Popa and Weaver
Fall 2021

CS 161
Computer Security

Final Exam

PRINT your name: ______________________________, ______________________________
(last) (first)

PRINT your student ID: ____

You have 170 minutes. There are 11 questions of varying credit (200 points total).

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

○ Unselected option (completely unfilled)

● 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).

It's EvanBot versus the Caltopian Space Agency! Who do you decide to align with?

This activity will not affect your grade in any way.

EvanBot ○

CSA ○

Q1 *Honor Code* (2 points)

Read the following honor code and sign your name.

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.

SIGN your name: ____

Q2 *True/false* (28 points)

Each true/false is worth 2 points.

Q2.1 True or False: Alice and Bob meet in-person to agree on a shared key K. After they separate, they use K to secure their communications using common cryptographic primitives. This is an example of security through obscurity.

○ True [✓ False]

Solution: Recall that Kerckhoff's principle states that everything is known to the adversary *except* for the secret key. Agreeing offline on a secret key is not security through obscurity and is a fairly standard practice.

Q2.2 A company hires an intern responsible for downloading and analyzing data from a database. The company grants this intern read/write access to the database containing relevant data.

True or False: This is a violation of the principle of least privilege.

[✓ True] ○ False

Solution: True. This intern shouldn't be granted read/write access to the database and should only have read access.

Q2.3 True or False: A secure PRNG is initialized with a high-entropy seed, then used to generate pseudorandom bits. It is computationally easy for an attacker who sees the pseudorandom bits to learn the seed.

○ True [✓ False]

Solution: False. By definition, it is computationally hard for an attacker to learn the PRNG input just by looking at the PRNG output.

Q2.4 True or False: In Bitcoin, if someone has 51% compute power, they can forge transactions.

○ True [✓ False]

Solution: False. The Bitcoin protocol uses public/private keypairs to manage identities. All transactions are signed with a user's private key. Because digital signatures are unforgeable, an attacker who doesn't know a user's secret signing key will be unable to impersonate that user (and thus, will be unable to forge transactions on behalf of that user). Having 51% control of the network won't allow attackers to bypass this!

Q2.5 TRUE or FALSE: The same origin-policy (SOP) determines which cookies a browser will include in an HTTP request.

○ TRUE [✓ FALSE]

Solution: False. Cookie policy determines which cookies are included in HTTP requests, not the same-origin policy.

Q2.6 TRUE or FALSE: If Mallory successfully executes a DHCP spoofing attack, Alice can no longer rely on the end-to-end security guarantees provided by TLS.

○ TRUE [✓ FALSE]

Solution: False. Defending against low-layer attacks like DHCP spoofing is hard, because there is no trusted party to rely on when we're first connecting to the network. As such, we rely on higher layers to defend against DHCP spoofing attacks (e.g. end-to-end encryption guarantees provided by TLS). Even if an attacker acquires MiTM privileges through DHCP spoofing, TLS will still protect the confidentiality and integrity of any messages sent between Mallory and her destination.

Q2.7 TRUE or FALSE: UDP guarantees that packets which are delivered are delivered in order, though it does not guarantee that any given packet is actually delivered.

○ TRUE [✓ FALSE]

Solution: False. UDP doesn't guarantee in-order delivery *or* delivery. One could add a counter to the payload of what they are sending with UDP, but UDP itself does not provide this information.

Q2.8 TRUE or FALSE: The TLS layer is responsible for retransmitting TLS packets if they get dropped.

○ TRUE [✓ FALSE]

Solution: False. TLS runs over a bytestream abstraction, and the underlying TCP layer is responsible for retransmitting data that gets dropped.

Q2.9 TRUE or FALSE: RSA TLS and Diffie-Hellman TLS would both still be safe from replay attacks, even if all clients used the same, fixed value as the client random.

[✓ TRUE] ○ FALSE

Solution: True. Both RSA TLS and Diffie-Hellman TLS include another random value from the client. In RSA TLS, the premaster secret is chosen randomly by the client, and in Diffie-Hellman TLS, the client's private Diffie-Hellman component is chosen randomly.

Q2.10 TRUE or FALSE: RSA TLS provides the property of forward secrecy, but Diffie-Hellman TLS does not.

○ TRUE [✓ FALSE]

Solution: False. DH-TLS provides forward secrecy, but RSA-TLS does not.

An adversary who records the network traffic and later compromises the RSA private key associated with the TLS certificate would be able to obtain the plaintext of all values during the TLS handshake required to derive the symmetric keys used for encryption (confidentiality). Thus, they could then decrypt all of the recorded HTTP request/response traffic.

Q2.11 TRUE or FALSE: If MD5 is used to hash the domain names, then domain enumeration is possible in NSEC3.

[✓ TRUE] ○ FALSE

Solution: True. MD5 is a broken hash, so the attacker can reverse the MD5 hash to learn the domain names.

Q2.12 TRUE or FALSE: Specification-based detection is easier to manage than signature-based detection, because you can easily reuse specifications collected from the security community.

○ TRUE [✓ FALSE]

Solution: False. Specification-based detection usually requires manually specifying allowed behavior. Signature-based detection is easier to manage because you can easily reuse signatures collected from the security community.

Q2.13 TRUE or FALSE: One method of determining if some code is malware is running it in a sandbox (isolated environment) and detecting malicious behavior.

[✓ TRUE] ○ FALSE

Solution: True. There is no algorithm to perfectly classify whether code is bad or not. For example, the malware could check if it's being run in a sandbox and only perform malicious actions if it's not being run in a sandbox.

Q2.14 TRUE or FALSE: After Alice chooses three nodes in the Tor network to form a circuit, it is possible to establish a connection to all three nodes in parallel.

○ TRUE [✓ FALSE]

Solution: False. Because Alice needs to conduct each DH exchange *through* the previous node, it is not possible to do these handshakes in parallel.

Q2.15 (0 points) `EvanBot is a real bot.`

[✓ True] False

Solution: `True.`

Q3 *Echo, Echo, Echo* **(16 points)**

Consider the following vulnerable C code:

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

char name[32];

void echo(void) {
    char echo_str[16];
    printf("What do you want me to echo back?\n");
    gets(echo_str);
    printf("%s\n", echo_str);
}

int main(void) {
    printf("What's your name?\n");
    fread(name, 1, 32, stdin);
    printf("Hi %s\n", name);

    while (1) {
        echo();
    }

    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 questions. For the first 4 parts, assume that **no memory safety defenses** are enabled.

Q3.1 (2 points) Assume that execution has reached line 8. Fill in the following stack diagram. Assume that each row represents 4 bytes.

Stack
1
2
RIP of `echo`
SFP of `echo`
3
4

○ (A) (1) - RIP of `main`; (2) - SFP of `main`; (3) - `echo_str[0]`; (4) - `echo_str[4]`

○ (B) (1) - SFP of `main`; (2) - RIP of `main`; (3) - `echo_str[0]`; (4) - `echo_str[4]`

[✓ (C)] (1) - RIP of `main`; (2) - SFP of `main`; (3) - `echo_str[12]`; (4) - `echo_str[8]`

Solution: The first two items on the stack are the RIP and SFP of `main`, respectively. Since the stack grows down, lower addresses are at the bottom of the diagram, and arrays are filled from lower addresses to higher addresses and are zero-indexed. As such, row (3) contains `echo_str[12]`, and row (4) contains `echo_str[8]`.

Q3.2 (3 points) Using GDB, you find that the address of the RIP of `echo` is `0x9ff61fc4`.

Construct an input to `gets` that would cause the program to execute malicious shellcode. Write your answer in Python 2 syntax (like in Project 1). You may reference `SHELLCODE` as a 16-byte shellcode.

Solution: Where to put the `SHELLCODE` does not matter. This is a simple stack-smashing attack: we want to redirect execution to `SHELLCODE` when the `echo` function returns.

Approach 1: Place the Shellcode in the Buffer

```
SHELLCODE + 'A' * 4 + '\xb0\x1f\xf6\x9f'
```

Approach 2: Place the Shellcode above the RIP

```
'A' * 20 + '\xc8\x1f\xf6\x9f' + SHELLCODE
```

There may be a few other correct answers here (with the shellcode placed at slightly different offsets within the buffer or above the RIP), but these are the most common.

Q3.3 (4 points) Which of the following defenses on their own would prevent an attacker from executing the exploit above? Select all that apply.

■ (A) Stack canaries

■ (B) Pointer authentication

■ (C) Non-executable pages

■ (D) ASLR

☐ (E) None of the above

Solution: Stack canaries defend against this attack because we are consecutively writing from the local variables to the RIP. The canary would be checked when the `echo` function returns, and because we don't have a way to leak the value of the canary in our exploit, canaries effectively stop our exploit from succeeding.

Non-executable pages defend against our exploit by preventing the shellcode on the stack (a write-not-execute region of memory) from being executed.

If ASLR were enabled, we wouldn't be able to reliably find the address of the RIP - it would change every time! We'd have to use one of our special attacks specifically for ASLR to bypass this (e.g. ROP). As such, ASLR stops our original exploit from succeeding.

Pointer authentication would require us to forge a valid pointer authentication code along with our new RIP. We don't have a way to do this, so pointer authentication stops our exploit from succeeding.

Note: we received a handful of questions asking if the "Pointer Authentication" answer choice was referring to a 32-bit system, which is what was stated in the prologue of this question, or a 64-bit system, which is what we originally intended. As such, we awarded credit for both answer choices.

Q3.4 (5 points) Assume that non-executable pages are enabled so we cannot execute SHELLCODE on stack. We would like to exploit the `system(char *command)` function to start a shell. This function executes the string pointed to by `command` as a shell command. For example, `system("ls")` will list files in the current directory.

Construct an input to `gets` that would cause the program to execute the function call `system("sh")`. Assume that the address of `system` is `0xdeadbeef` and that the address of the RIP of `echo` is `0x9ff61fc4`. Write your answer in Python 2 syntax (like in Project 1).

Hint: Recall that a return-to-libc attack relies on setting up the stack so that, when the program pops off and jumps to the RIP, the stack is set up in a way that looks like the function was called with a particular argument.

Solution: Our goal is to make `echo` return to the system function by changing the `RIP of echo` to the address of `system`. When `echo` returns to `system`, the stack should look like the stack diagram below, because by calling convention the callee expects its arguments and its RIP to be pushed onto the stack by the caller. It's the callee's responsibility to push the SFP onto the stack as its first step.

Therefore we need to first place garbage bytes from the beginning of `name` up to the `RIP of echo` (`'A' * 20`) and replace the `RIP of echo` with the address of `system` (`'\xef\xbe\xad\xde'`) so that `echo` will return to `system`. Now, we want to create the stack diagram above to make the stack in line with what the `system` method expects. Thus, we add four bytes of garbage where the `system` method expects `RIP of system` to be. Note that, `RIP of system` is the address that `system` method will return to. Then, we place the address of "sh" at the location where `system` expects an argument, and place the string "sh" at that location (which is 8 bytes above the `RIP of system`).

Stack

`command` (pointer to `"sh"`)
(Expected) RIP of `system`

As such, our exploit may look something like the following:

```
'A' * 20 + '\xef\xbe\xad\xde' + 'B' * 4 + '\xd0\x1f\xf6\x9f'
+ 'sh' + '\x00'
```

NOTE: Since the stack below the `RIP of echo` will get invalidated (because it's below the ESP) after `echo` returns, we cannot make any assumptions about whether the values placed there would remain as is. Therefore, you should not place the string "sh" in `name`.

Q3.5 (2 points) Assume that, in addition to non-executable pages, ASLR is also enabled. However, addresses of global variables are not randomized.

Is it still possible to exploit this program and execute malicious shellcode?

○ (A) Yes, because you can find the address of both `name` and `system`

○ (B) Yes, because ASLR preserves the relative ordering of items on the stack

○ (C) No, because non-executable pages means that you can't start a shell

[✓ (D) No, because ASLR will randomize the code section of memory]

Solution: If ASLR is enabled, the address of `system`, a line of code in the *code* section of memory, will be randomized each time the program is run. Because our exploit uses this address, ASLR will effectively prevent us from using our approach!

Q4 *Challenge-Response* (28 points)

Clarification during exam: Weak unforgeability and unforgeability mean that the attacker is trying to directly authenticate to Bob, while Alice is offline.

Suppose that Alice wants to authenticate herself to Bob, such that Bob knows that he is talking to Alice. Formally, Bob wants to verify that Alice knows some fixed secret K (which Bob may not necessarily know). One common method of authentication is *challenge-response authentication*, which has two main phases:

- **Challenge**: Bob generates a challenge C, and sends C to Alice. C may be different each time Alice wants to authenticate herself.
- **Response**: Alice uses the challenge C and her secret K in order to construct a response R, and she sends R to Bob. If Bob successfully verifies the response, Bob knows that he is talking to Alice.

Consider the following properties of challenge-response authentication:

- **Weak unforgeabilty**: An adversary that does not know K should not be able to authenticate to Bob. Assume that the adversary **has not** observed any past authentication rounds between Alice and Bob.
- **Unforgeabilty**: An adversary that does not know K should not be able to authenticate to Bob. Assume that the adversary **has** observed (but not been a MITM in) past, successful authentication rounds between Alice and Bob.
- **Weak key secrecy**: A **passive** adversary that observes any number of authentication rounds between Alice and Bob cannot recover K.
- **Key secrecy**: An **active** adversary that is a MITM for any number of authentication rounds between Alice and Bob cannot recover K.

For each challenge-response authentication scheme, select the property or properties that the scheme possesses. Assume that:

- Enc/Dec is a semantically secure asymmetric encryption scheme.
- Sign/Verify is a secure, unforgeable digital signature scheme.

Q4.1 (4 points) **Scheme 1**: K is a 128-bit random key known to both Alice and Bob.

Challenge: Bob generates a new asymmetric key pair (SK_B, PK_B) and sends the challenge $C = PK_B$.

Response: Alice sends the response $R = \mathsf{Enc}(PK_B, K)$. Bob verifies that $\mathsf{Dec}(SK_B, R)$ is equal to K.

■ (A) Weak unforgeabilty ■ (C) Weak key secrecy □ (E) None of the above

■ (B) Unforgeabilty □ (D) Key secrecy

Solution: Notice that this scheme requires knowledge of the key K in order to compute $\mathsf{Enc}(PK_B, K)$. Because PK_B changes for every round, a replay attack is ineffective against this scheme, and both unforgeability requirements are satisfied. Additionally, because K is only ever sent as ciphertext, an observer cannot learn the value of K. However, an active, MITM attacker would be able to replace PK_B with their own, malicious public key, tricking Alice into encrypting K using a public key corresponding to a private key owned by the MITM attacker.

Q4.2 (4 points) **Scheme 2**: $K = SK_A$ is a private key known only to Alice. Bob knows Alice's public key PK_A.

Challenge: Bob sends a challenge $C = r$, where r is a randomly chosen, 256-bit number.

Response: Alice sends the response $R = \mathsf{Sign}(SK_A, C)$. Bob verifies that $\mathsf{Verify}(PK_A, R)$ returns true.

■ (G) Weak unforgeabilty ■ (I) Weak key secrecy □ (K) None of the above

■ (H) Unforgeabilty ■ (J) Key secrecy

Solution: Because this scheme relies on a signature over a random message, an adversary must have the secret key SK_A in order to forge a valid response. This secret key is never exposed to an adversary (only the signatures, which reveal nothing about the key), so this scheme fulfills all requirements.

Q4.3 (4 points) **Scheme 3**: K is a 128-bit random key known to both Alice and Bob.

Challenge: Bob sends a challenge $C =$ “” (empty message).

Response: Alice sends the response $R = K$. Bob verifies that the response is equal to K.

■ (A) Weak unforgeabilty ☐ (C) Weak key secrecy ☐ (E) None of the above

☐ (B) Unforgeabilty ☐ (D) Key secrecy

Solution: Without having seen the key sent in a previous authentication round, an adversary can't forge a valid response. However, once the adversary observes the key, they can authenticate themselves. Both the passive and active adversaries learn the key K when they see the response, since it is sent in plaintext.

Q4.4 (4 points) **Scheme 4**: $K = SK_A$ is a private key known only to Alice. Bob knows Alice's public key PK_A.

Challenge: Bob sends a challenge $C = \mathsf{Enc}(PK_A, r)$, where r is a randomly chosen, 128-bit number.

Response: Alice sends the response $R = \mathsf{Dec}(SK_A, C)$. Bob verifies that R is equal to r.

■ (G) Weak unforgeabilty ■ (I) Weak key secrecy ☐ (K) None of the above

■ (H) Unforgeabilty ■ (J) Key secrecy

Solution: Similar to the signature scheme, an adversary must have SK_A in order to learn the value of r to forge the response. This key is never exposed, only ciphertexts generated from the keys, so all requirements are fulfilled.

Q4.5 (4 points) **Scheme 5**: K is a 128-bit random key known to both Alice and Bob.

Challenge: Bob sends a challenge $C =$ “” (empty message).

Response: Alice sends the response $R = \mathsf{H}(K)$, where H is a secure, cryptographic hash function. Bob verifies that R is equal to $\mathsf{H}(K)$.

■ (A) Weak unforgeabilty ■ (C) Weak key secrecy ☐ (E) None of the above

☐ (B) Unforgeabilty ■ (D) Key secrecy

Solution: This scheme is marginally better than sending K in the clear, since an adversary will never be able to recover K from $\mathsf{H}(K)$. Without knowing K, it is impossible to forge $\mathsf{H}(K)$ as a valid response. However, once an adversary possesses $\mathsf{H}(K)$ from a previous authentication round, they can replay it to the server as a forged response.

Q4.6 (5 points) Assume that Alice and Bob perform an ephemeral Diffie-Hellman key exchange to derive a symmetric key and form an encrypted channel. Alice authenticates herself to Bob **through** the encrypted channel, using one of the schemes above. Afterwards, Bob receives the message "Send $10 to Charlie," also through the encrypted channel. For which of these schemes can Bob be sure that Alice sent this message? Select all that apply.

☐ (G) Scheme 1
☐ (H) Scheme 2
☐ (I) Scheme 3
☐ (J) Scheme 4
☐ (K) Scheme 5
■ (L) None of the above

Solution: The answer here is actually the same as in the previous question. An adversary could act as a MITM during the initial Diffie-Hellman and forward Alice's authentication messages. After this, Mallory can forge the "Send $10 to Charlie." message to Bob, since there is nothing that binds the message to the authentication.

Q4.7 (3 points) Notice that password-based authentication is very similar to Scheme 3: The user uses their password as the secret K and sends it to the server to authenticate themselves. Suppose that Alice authenticates herself their username and password to a bank website that uses secure HTTPS, using Diffie Hellman TLS.

In the same HTTPS channel, Alice makes another request to the bank: "Send $10 to Charlie." Can the bank be sure that Alice made this request? Assume that no one else knows Alice's password. Briefly justify your answer (1–2 sentences).

Solution: The key distinction between standard authentication over a Diffie-Hellman channel and password authentication over HTTPS is that HTTPS uses TLS, which authenticates the *server* during the TLS handshake. By authenticating the server, the client can be sure that they are communicating with the server, rather than an adversary performing a MITM attack on Diffie-Hellman. Because the server is authenticated, the user can guarantee that sending the plaintext of the password is safe.

Q5 *Bob's Birthday* (11 points)

It's Bob's birthday! Alice wants to send an encrypted birthday message to Bob using ElGamal.

Recall the definition of ElGamal encryption:

- b is the private key, and $B = g^b \bmod p$ is the public key.
- $\mathsf{Enc}(B, M) = (C_1, C_2)$, where $C_1 = g^r \bmod p$ and $C_2 = M \times B^r \bmod p$
- $\mathsf{Dec}(b, C_1, C_2) = C_1^{-b} \times C_2 \bmod p$

Q5.1 (2 points) Mallory wants to tamper with Alice's message to Bob. In response, Alice decides to sign her message with an RSA digital signature. Bob receives the signed message and verifies the signature successfully. Can he be sure the message is from Alice?

[✓ (A) Yes, because RSA digital signatures are unforgeable.]

○ (B) Yes, because RSA encryption is IND-CPA secure.

○ (C) No, because Mallory could have blocked Alice's message and replaced it with a different one.

○ (D) No, because Mallory could find a different message with the same hash as Alice's original message.

Solution: RSA digital signatures, when paired with a secure hash function, are believed to be unforgeable. See the textbook for a game-based definition of what exactly we mean by unforgeable.

As we discussed in class, ElGamal is malleable, meaning that a man-in-the-middle can change a message in a *predictable* manner, such as producing the ciphertext of the message $2 \times M$ given the ciphertext of M.

Q5.2 (3 points) Consider the following modification to ElGamal: Encrypt as normal, but further encrypt portions of the ciphertext with a block cipher E, which has a block size equal to the number of bits in p. In this scheme, Alice and Bob share a symmetric key $K_{\textsf{sym}}$ known to no one else.

Under this modified scheme, C_1 is computed as $E_{K_{\textsf{sym}}}(g^r \bmod p)$ and C_2 is computed as $E_{K_{\textsf{sym}}}(M \times B^r \bmod p)$. Is this scheme still malleable?

○ (G) Yes, because block ciphers are not IND-CPA secure encryption schemes

○ (H) Yes, because the adversary can still forge $k \times C_2$ to produce $k \times M$

[✓ (I) No, because block ciphers are a pseudorandom permutation]

○ (J) No, because the adversary isn't able to learn anything about the message M

Solution: While block ciphers aren't IND-CPA secure, they are secure when encrypting "random-looking" values because of their properties as pseudorandom permutations. As long as the values you encrypt are unique, the output of the block cipher will always be secure. ElGamal's C_1 and C_2 both appaer random.

Additionally, because block ciphers are a PRP, the scheme is no longer malleable, because modifying the ciphertext in any way causes an unpredictable change to the result of decrypting the block cipher with $D_{K_{\textsf{sym}}}$.

The remaining parts are independent of the previous part.

For Bob's birthday, Mallory hacks into Bob's computer, which stores Bob's private key b. She isn't able to read b or overwrite b with an arbitrary value, but she can multiply the stored value of b by a random value z known to Mallory.

Mallory wants to send a message to Bob that appears to decrypt as normal, but **using the modified key** $b \cdot z$. Give a new encryption formula for C_1 and C_2 that Mallory should use. Make sure you only use values known to Mallory!

Clarification during exam: For subparts 5.3/5.4, assume that the value of B is unchanged.

Q5.3 (3 points) Give a formula to produce C_1, encrypting M.

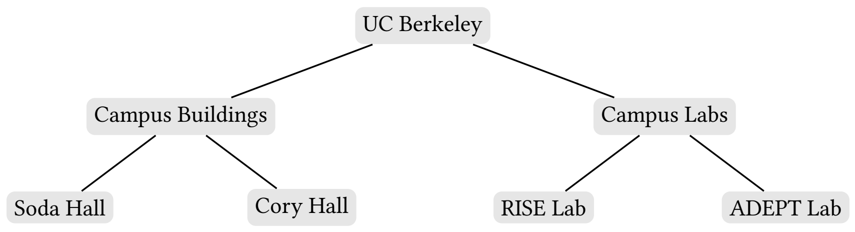
Solution: Mallory should send g^r with some randomly chosen r, as usual.

Q5.4 (3 points) Give a formula to produce C_2, encrypting M.

Solution: Mallory should send $C_2 = m \times B^{rz} \mod p$.

Q6 *RISELab Shenanigans* (16 points)

Certificate authorities of UC Berkeley are organized in a hierarchy as follows:



Alice is a student in RISELab at UC Berkeley and wants to obtain a certificate for her public key. Assume that only RISELab is allowed to issue certificates to Alice.

Q6.1 (2 points) Which of the following values are included in the certificate issued to Alice? Select all that apply.

■ (A) Alice's public key

☐ (B) Alice's private key

■ (C) A signature on Alice's *public* key, signed by RISELab's private key

☐ (D) A signature on Alice's *private* key, signed by RISELab's private key

☐ (E) None of the above

Solution: This follows from the definition of certificates: they include a user's public key, and a signature on the enclosed public key, signed by the issuer (which we state in the prologue is RISELab).

Q6.2 (2 points) Assume that the only public key you trust is UC Berkeley's public key. Which certificates do you need to verify in order to be sure that you have Alice's public key? Select all that apply.

■ (G) Certificate for Alice

☐ (H) Certificate for Soda Hall

■ (I) Certificate for RISELab

■ (J) Certificate for Campus Labs

☐ (K) None of the above

Solution: To validate Alice's public key, we can follow our way up to our root of trust (which is UC Berkeley's public key). As such, we need certificates for Alice, RISELab, and Campus Labs.

Q6.3 (4 points) RISELab issues a certificate to Alice that expires in 1 hour. Which of the following statements are true about using such a short expiration date? Select all that apply.

■ (A) It mitigates attacks where Alice's private key is stolen

☐ (B) It mitigates attacks where RISELab's private key is stolen

☐ (C) It mitigates attacks where Campus Labs' private key is stolen

■ (D) It forces Alice to renew the certificate more often

☐ (E) None of the above

Solution: Short expiration times only mitigate the situation where Alice's private key is stolen. If RISELab's private key is compromised, the attacker can issue certificates with any expiration date, and it is up to the parent CA to revoke RISELab's certificate, not RISELab itself. The same argument applies to Campus Labs' private key.

The following subparts are independent from the previous subparts.

Passwords on the RISELab website are six-digit codes, where each digit is one of 0–9 (repeat digits are allowed). An attacker steals the password database, which includes Alice's hashed password, and wants to learn Alice's password.

For each password storage scheme, *in the worst case*, how much time would it take for the attacker to brute-force Alice's password?

Assumptions:

- The attacker tries passwords one at a time.
- H is a hash function that takes 1 second to compute.
- The time required for all other operations is negligible.

Q6.4 (2 points) Passwords are stored as $H(\mathsf{pwd})$.

○ (G) $10^6 \cdot 2 \cdot 8$ seconds ○ (I) $10^6 \cdot 2^8$ seconds ○ (K) 2^8 seconds

○ (H) $6 \cdot 10 \cdot 2^8$ seconds [✓ (J) 10^6 seconds]

Solution: Since the password is six-digits, and there are 10 possibilities for each digit, the attacker must try 10^6 possible passwords in the worst case.

Q6.5 (2 points) Passwords are stored as $(\mathsf{salt}, H(\mathsf{salt}\|\mathsf{pwd}))$, where salt is an 8-bit random string.

○ (A) $10^6 \cdot 2 \cdot 8$ seconds ○ (C) $10^6 \cdot 2^8$ seconds ○ (E) 2^8 seconds

○ (B) $6 \cdot 10 \cdot 2^8$ seconds [✓ (D) 10^6 seconds]

> **Solution:** Since the attacker knows the salt—it's stored next to the password in plaintext—the worst-case number of tries the attacker must attempt doesn't change from the previous subpart: the answer is 10^6 once again.

Q6.6 (4 points) Assume that the attacker is conducting an **online** brute-force attack against Alice's account. Which of the following changes, if implemented individually, would make it more difficult for the attacker to access Alice's account? Select all that apply.

[✓ (G) Alice uses a random, alphanumeric, 32-character password instead of a 6-digit numeric password]

[✓ (H) Alice enables two-factor authentication on her account]

[✓ (I) RISELab imposes a timeout which doubles after each password attempt]

☐ (J) RISELab enables TLS for its login page

☐ (K) None of the above

> **Solution:** Using a longer password with more possibilities would make it more difficult to brute force.
>
> Enabling 2FA would prevent an attacker from compromising Alice's account even if the attacker managed to brute force Alice's password.
>
> Enabling a timeout would successfully mitigate an online brute force attack – especially one where the timeout doubles after each attempt!
>
> However, enabling TLS wouldn't make it more difficult to prevent an online brute-force attack; the attacker's sole goal in brute force is to access Alice's account by trying many, many times; we didn't specify that this attacker had any other information (e.g. we didn't say the attacker was on-path, etc.), so TLS wasn't a valid answer choice here.

Q7 *SQL Injection* (20 points)

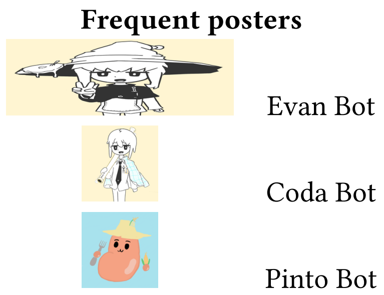
CS 161 students are using a modified version of Piazza to discuss project questions! In this version, the names and profile pictures of the students who answer questions frequently are listed on a side panel on the website.

The server stores a table of users with the following schema:

```
CREATE TABLE users (
    First TEXT,                 -- First name of the user.
    Last TEXT,                  -- Last name of the user.
    ProfilePicture TEXT,        -- URL of the image.
    FrequentPoster BOOLEAN,     -- Are they a frequent poster?
);
```

Q7.1 (3 points) Assume that you are a frequent poster. When playing around with your account, you notice that you can set your profile picture URL to the following, and your image on the frequent poster panel grows wider than everyone else's photos:

ProfilePicture URL: `https://cs161.org/evan.jpg" width="1000`



What kind of vulnerability might this indicate on Piazza's website?

[✓ (A) Stored XSS]

(B) Reflected XSS

(C) CSRF

(D) Path traversal attack

(E) Buffer overflow

Solution: Because the user seems to be able to inject arbitrary HTML through the image URL, this might indicate a stored XSS vulnerability. The user can submit an profile picture URL that escapes the `img` tag of the image and injects a malicious script into future users who attempt to load the profile picture.

Q7.2 (3 points) Provide a malicious image URL that causes the JavaScript `alert(1)` to run for any browser that loads the frequent poster panel. Assume all relevant defenses are disabled.

Hint: Recall that image tags are typically formatted as `<img src="image.png">`.

Solution: The input would look something like the following:

```
"><script>alert(1)</script><img src="
```

So when injected into the image, this would render as:

```
<img src="image.png"><script>alert(1)</script><img src="">
```

We assume that all relevant defenses (e.g. content security policy) are disabled, so this script will run when the frequent poster panel is loaded.

Q7.3 (4 points) Suppose your account is not frequent poster, but you still want to conduct an attack through the frequent posters panel!

When a user creates an account on Piazza, the server runs the following code:

```
query := fmt.Sprintf("
    INSERT INTO users (First, Last, ProfilePicture, FrequentPoster)
        VALUES ('%s', '%s', '%s', FALSE);
    ",
    first, last, profilePicture)
db.Exec(query)
```

Provide an input for `profilePicture` that would cause your malicious script to run the next time a user loads the frequent posters panel. You may reference `PAYLOAD` as your malicious image URL from earlier, and you may include `PAYLOAD` as part of a larger input.

Solution: There's a key insight here: your accout isn't a frequent poster, but you want it to show up in the frequent posters panel, so you need to set `FrequentPoster` to `TRUE` for that to happen! Because it's hardcoded as `FALSE` in the current injection, we need to do something like the following:

```
PAYLOAD', TRUE) --
```

As a result, the following SQL will be executed:

```
INSERT INTO users (First, Last, ProfilePicture, FrequentPoster)
    VALUES ('[some first name]', '[some last name]',
        'PAYLOAD', TRUE) --', FALSE);
```

Q7.4 (4 points) Instead of injecting a malicious script, you want to conduct a DoS attack on Piazza! Provide an input for `profilePicture` that would cause the SQL statement `DROP TABLE users` to be executed by the server.

Solution: Similar to the previous problem, we're going to construct a SQL injection attack. This time, we need to start a completely new statement, so we'll use a semicolon to start the `DROP TABLE users` statement:

```
', FALSE); DROP TABLE users --
```

This results in the following SQL being executed:

```
INSERT INTO users (First, Last, ProfilePicture, FrequentPoster)
    VALUES ('[some first name]', '[some last name]',
        '', FALSE); DROP TABLE users --', FALSE);
```

Suppose that session cookies are used to authenticate to Piazza. This token is checked whenever the user sends a request to Piazza.

Clarification during exam: "Your malicious script" refers to your exploit in 7.2.

Q7.5 (3 points) Your malicious script submits a GET request to the Piazza website that marks "helpful!" on one of your comments. Does the same-origin policy defend against this attack?

- (A) Yes, because the same-origin policy prevents the script from making the request
- (B) Yes, because the script runs with the origin of the attacker's website
- [✓ (C) No, because the same-origin policy does not block any requests from being made]
- (D) No, because the script runs with the origin of Piazza's website

Solution: The best answer here is that the SOP (in the context of how we teach it in this class) doesn't block any requests from being made – so if a request is being made from the Piazza homepage that makes a change on Piazza's webpage, then SOP doesn't block that request from occurring.

It is true that the script runs with the origin of Piazza's website, but even if it ran from the origin of a different website, SOP (again, in the context of how we teach it in class) wouldn't block the request from being made. So the third answer choice is the best answer here.

Q7.6 (3 points) Your malicious script submits a GET request to the Piazza website that marks "helpful!" on one of your comments. Does enabling CSRF tokens defend against this attack?

○ (G) Yes, because the attacker does not know the value of the CSRF token

○ (H) Yes, because the script runs with the origin of the attacker's website

○ (I) No, because GET requests do not change the state of the server

[✓ (J) No, because the script runs with the origin of Piazza's website]

Solution: Since the script runs in the origin of the Piazza website, the script can leak the value of the CSRF token presumably embedded in the HTML and make a GET request with a legitimate CSRF token.

GET requests *can* change the state of the server; it's only convention that they *usually* don't do this.

We don't usually talk about how CSRF tokens work with GET requests in this class, but we do give you enough information to reason this one out!

Q8 *Integration Testing* (28 points)

Note: We think this is the hardest question on the exam. Feel free to skip it for now and come back to it later.

Engineers at Gradescope have introduced a new scheme for managing session tokens. Here's how it works:

1. Alice visits `https://gradescope.com/login`, enters her email/password, and clicks "Submit".
2. The JavaScript on the login page makes a POST request to `https://gradescope.com/login` with Alice's email and password.
3. The server validates Alice's email and password and sends a randomly generated session token in the **response body** (not as a cookie).
4. The JavaScript on the login page extracts the token from the response and redirects to `https://gradescope.com/home?token=[sessiontoken]`

The Gradescope web server will check that any URL that requires the user to be signed in contains the `token=[sessiontoken]` parameter in the query string. If it is not present, the server will return an "Access Denied" page instead of responding with the page itself. If successful, the page will also contain the session token somewhere on the page:

```
<p id="token">[sessiontoken]</p>
```

The server ensures that all future requests contain the same token by including `?token=[sessiontoken]` in all Gradesscope links that are present in the HTML response. For example, the link to go to the `https://gradescope.com/grades` page will have its URL set to `https://gradescope.com/grades?token=[sessiontoken]`.

For all parts of this question, make the following assumptions:

- All TLS connections use Diffie-Hellman TLS.
- Alice's browser always sends a Referer header.
- The attacker has full control over the `evil.com` web server.
- All parts of this question are independent.

Q8.1 (5 points) After Alice completes the sign-in flow, she sees a number of links on the homepage she can click on. **An attacker has discovered a vulnerability on each linked page,** allowing them to execute arbitrary JavaScript on that page. If Alice clicks on these links, which of the following links may cause her session token to be leaked to the attacker? Select all that apply.

■ (A) `http://gradescope.com/assignments?token=[sessiontoken]`

■ (B) `https://gradescope.com:1234/?token=[sessiontoken]`

■ (C) `http://support.gradescope.com/?token=[sessiontoken]`

■ (D) `https://gradescope.com/assignments?token=[sessiontoken]`

■ (E) `http://evil.com/`

☐ (F) None of the above

Solution: For all Gradescope sites, the session token will be passed in a query string parameter. Because of this, the site will embed Alice's session token in the page response. We can inject malicious JavaScript onto the page to leak the token from the HTML body. As such, Alice should avoid clicking these links.

For `http://evil.com`, the previous page's session token is still included in the Referer header, which can be seen by the web server.

Q8.2 (5 points) After Alice completes the sign-in flow, she clicks on the following link from the homepage:

```
https://gradescope.com/assignments?token=[sessiontoken]
```

Eve, an on-path eavesdropper, observes a snapshot of the request and response corresponding to this query. Which of the following attackers could individually collude with Eve for Eve to be able to learn Alice's **session token**? Select all that apply.

■ (G) An attacker with control over the Gradescope web server

■ (H) An attacker who's capable of solving the discrete-log problem

■ (I) An attacker who's capable of exerting infinite computational power

■ (J) An attacker who has full control over a HIDS on Alice's device

☐ (K) An attacker who can see Alice's cookies

☐ (L) None of the above

Solution: The actual URL that a GET Request is made to in this case will look something like this:

```
https://gradescope.com/assignments/?token=[sessiontoken]
```

An attacker with control over Gradescope's web server would be able to trivially learn Alice's session token (e.g. by looking at the server-side database mapping tokens to users).

The message request/response will contain the session token in the query string parameter. However, the channel is protected over TLS, but TLS relies on Diffie Hellman, which in turn relies on the discrete-log problem! If an attacker breaks the discrete-log problem, then our security/confidentiality guarantees fall apart. Attacker with infinite computational power could solve the discrete log problem, so they fall into this category as well.

HIDS use TLS keys to look at decrypted network traffic, so an attacker with control over Alice's HIDS would be able to see the session token in the request URL.

Since session tokens aren't stored in cookies, there isn't much looking at cookies will help us with.

Q8.3 (3 points) After Alice completes the sign-in flow, she clicks on the following link from the homepage:

```
https://gradescope.com/assignments?token=[sessiontoken] (Version 1)
http://gradescope.com/assignments?token=[sessiontoken] (Version 2)
```

For this question part only, assume that Alice uses the Tor browser instead of her regular browser. Which of the compromised nodes could learn Alice's **session token**? Select all that apply.

☐ (A) A compromised exit node in a two-node circuit

☐ (B) A compromised exit node in a three-node circuit

☐ (C) The node in a one-node circuit

☐ (D) A compromised entry node in a two-node circuit

☐ (E) A compromised middle node in a three-node circuit

[✓ (F) None of the above]

Solution: The actual URL that a GET Request is made to in this case will look something like this:

```
https://gradescope.com/assignments/?token=[sessiontoken] (Version 1)
http://gradescope.com/assignments/?token=[sessiontoken] (Version 2)
```

As discussed in lecture and in notes, the Tor exit node can see the message plaintext (M)—which, in the case of HTTP, is the HTTP payload! However: if M is encrypted (which, in this case it is, since we're using HTTPS)—none of these adversaries will be able to learn any information about the contents of the message, including, in this case, the URL of the HTTP request. As such, no nodes will be able to learn Alice's session token!

Note: there were two versions of this question: One with `http:` and one with `https:`. For the versions that received HTTP, then the correct answer would require selecting all of the choices with a compromised exit node (A, B, and C).

Q8.4 (5 points) After Alice completes the sign-in flow, she sees a number of links on the homepage she can click on. An **on-path** attacker wants to set a cookie with the following attributes in Alice's browser:

```
Domain=gradescope.com
Path=/home
HttpOnly=true
```

If Alice clicks on these links, which of the following links would allow the attacker to set a cookie with these attributes in Alice's browser? Select all that apply.

■ (G) `http://gradescope.berkeley.edu/`

■ (H) `http://gradescope.com:8081/?token=[sessiontoken]`

■ (I) `http://support.gradescope.com/grades?token=[sessiontoken]`

■ (J) `http://evil.com`

☐ (K) `https://gradescope.com/?token=[sessiontoken]`

☐ (L) None of the above

Solution: Cookie policy states that **when a server sets a cookie, the server's URL must end in the cookie's Domain attribute**. Because all of these requests are made over HTTP, an on-path attacker effectively has the same power as the server—they're capable of setting the SET-COOKIE header to whatever they'd like. As such, enforcement of cookie policy occurs when the response is received by the browser.

However, there's an additional catch here—all of these on-path attackers can tamper with the response, effectively allowing them to run arbitrary JavaScript on the page that's returned. This JavaScript can then make more requests, e.g. an intentional request to any site that DOES satisfy cookie policy. As such, those the attacker could add a SET-COOKIE header to these requests' responses, which would pass cookie policy enforcement by the browser!

Q8.5 (5 points) After Alice completes the sign-in flow, she sees a number of links on the homepage she can click on. An attacker **who has control over the web server at each of these links** wants to set a cookie with the following attributes in Alice's browser:

```
Domain=gradescope.com
Path=/home
Secure=true
```

If Alice clicks on these links, which of the following links may allow the attacker to set a cookie with these attributes in Alice's browser? Select all that apply.

■ (A) `http://gradescope.com/?token=[sessiontoken]`

☐ (B) `https://gradescope.berkeley.edu`

■ (C) `https://support.gradescope.com/grades/?token=[sessiontoken]`

☐ (D) `https://evil.com/`

■ (E) `https://gradescope.com/?token=[sessiontoken]`

☐ (F) None of the above

Solution: This question strictly asks about cookie policy; as such, we must look for all sites that end with `gradescope.com`. The `secure` flag doesn't matter when setting a cookie.

Q8.6 (5 points) Gradescope uses a `language` cookie with the following attributes:

```
Domain=gradescope.com
Path=/grades
Secure=false
HttpOnly=true
```

After Alice completes the sign-in flow, she sees a number of links on the homepage she can click on. If Alice clicks on these links, which of the following links may allow the attacker to learn the cookie value?

Assume that the attacker can observe and modify the request/response information of **only** the request made by Alice clicking the link and nothing else. Select all that apply.

■ (G) `http://support.gradescope.com/grades?token=[sessiontoken]`

☐ (H) `http://berkeley.edu/grades/1234/`

■ (I) `http://gradescope.com/grades/?token=[sessiontoken]`

☐ (J) `http://gradescope.cs161.org/home/`

☐ (K) `http://evil.com/`

☐ (L) None of the above

Solution: For a browser to send a cookie value with a request, the URL domain should end in the cookie's Domain attribute, and the URL path should begin with the cookie's Path attribute.

What about our previously-mentioned attack vector of JS injection? Notice that the HttpOnly flag is present. Because of this, even if an attacker can run JavaScript on any of these pages, that JavaScript wouldn't be able to leak this cookie value. If an attacker tried to spawn a new request from the JS code, that attacker wouldn't be able to see that request/response information (they're not on-path—they simply have a snapshot in this case).

Q9 *Pancake Query Protocol* (19 points)

EvanBot is already prepared for the winter break but realizes that there are no more pancakes and needs to order more! To speed up the ordering process, EvanBot crafts a custom Pancake Query Protocol (PQP) and needs to ensure that it is secure.

PQP runs directly over IP, and a PQP packet contains the following information:

- A packet type
- The pancake query data (either a request for an order, or the order itself)

For now, assume that the only packet type supported by PQP is the ORDER type. For example, EvanBot might send the following PQP packet:

- EvanBot $\longrightarrow$ Restaurant: {Type: ORDER; Data: "I want 1 stack of blueberry pancakes!"}

For all parts, assume that EvanBot knows the IP address of the restaurant. All subparts of this question are independent.

Q9.1 (5 points) Which of the following statements are true about PQP? Select all that apply.

☐ (A) An off-path attacker can learn EvanBot's order

■ (B) An off-path attacker can trick the restaurant into cooking unwanted pancakes for EvanBot

☐ (C) An on-path attacker can conduct a RST injection attack

■ (D) An on-path attacker can learn EvanBot's order

☐ (E) EvanBot can be sure that the restaurant received the order

☐ (F) None of the above

Solution: A: False. EvanBot will always send their order to the restaurant (i.e. not visible to the off-path attacker), since Bot knows the restaurant's IP address.

B: True. An off-path attacker can forge an IP packet as if it were coming from EvanBot and place an malicious order inside the forged PQP packet.

C: False. RST injection attacks don't exist without TCP, since a RST packet is a construct of TCP.

D: True. An on-path attacker can see the contents of the PQP packet and learn the order.

E: False. Raw IP does not provide reliability.

Q9.2 (3 points) EvanBot adds an ACK packet type to PQP packets. After a restaurant receives an order, the restaurant sends an ACK packet acknowledging the order. If EvanBot does not receive the ACK, EvanBot re-sends the order until an ACK is received.

EvanBot tries to order 1 stack of pancakes from the restaurant and eventually receives an ACK. Assume that **no** network attackers are present. How many orders could the restaurant receive?

○ (G) Exactly 0, because IP is unreliable.

○ (H) Either 0 or 1, because IP is unreliable.

○ (I) 0 or more, because IP is unreliable.

○ (J) Exactly 1, because the restaurant ACKs any order it receives.

[✓ (K) 1 or more, because EvanBot might try more than once.]

○ (L) 2 or more, because the restaurant may send multiple ACKs.

Solution: EvanBot succeeds in receiving an ACK, sot he restaurant must have received at least one order. However, it is possible that the restaurant received more than one order: One possible sequence of events is that EvanBot tried sending their order multiple times without receiving an ACK. However, the restaurant received all of these orders, but all of the ACK packets they sent back were lost, because IP is unreliable. Eventually, one of the ACK packets got through to EvanBot.

Q9.3 (4 points) Consider the following modification to PQP: To order, the client generates a random order ID and sends it in the PQP ORDER packet along with the order. The server sends back the order ID in the PQP ACK packet.

Can an off-path attacker trick the restaurant into accepting a spoofed order appearing to come from EvanBot? Briefly justify your answer (1–2 sentences).

[✓ (A) Yes] ○ (B) No

Solution: While including the random order ID makes this seem secure, notice that the restaurant takes action immediately when the order is received. An off-path attacker can thus forge an ORDER packet containing a malicious order and random order ID.

Q9.4 (4 points) Which of the following modifications to PQP, if made individually, would prevent an off-path attacker from tricking the restaurant into accepting spoofed orders? Select all that apply.

■ (G) The restaurant generates a random order ID and sends it back in the PQP ACK. The restaurant must receive a PQP ACK-ACK packet from EvanBot containing the order ID to confirm the order.

☐ (H) The restaurant sends a fixed time-to-live (TTL) to EvanBot in the PQP ACK. The restaurant must receive a final, empty PQP ACK packet from EvanBot within the TTL to confirm the order.

☐ (I) PQP runs over UDP instead of IP, and EvanBot chooses a random source port.

■ (J) PQP runs over TCP instead of IP.

☐ (K) None of the above

Solution: Having the *restaurant* generate a random order ID, sending it back to the client, and then having the client send that order ID back to the server to confirm the order would work. This ensures that an off-path attacker must guess information that it cannot see, since it can't see the random order ID generated by the restaurant.

Q9.5 (3 points) EvanBot proposes an additional packet types for PQP: LISTORDERS. When a restaurant receives an LISTORDERS packet, it responds with a list of all orders that it has ever received from any customer. Name one security issue with this proposal and describe the steps an attacker should take to exploit this issue (1–2 sentence).

Solution: This creates a classic amplified DoS attack: An attacker can spoof an ORDER_LIST packet appearing to come from a victim IP address, and the restaurant will send a very large response to the victim. This is similar to DNSSEC response packets and the deprecated NTP MONLIST command.

Q10 *In Medias Res* (14 points)

Recall that TLS uses a handshake to form a new connection. One possible optimization for TLS is to store the symmetric keys associated with a TLS session and use those keys to "resume" a previous TLS session, instead of performing a new handshake for every connection.

One implementation of session resumption uses a session ticket. During the initial connection, the handshake is as follows:

1. The client sends a ClientHello with no associated ticket.
2. The rest of the standard TLS handshake proceeds as normal.
3. The server generates a **session ticket** and sends the ticket to the client through the encrypted TLS channel.
4. The client stores the ticket.

If the client wants to resume this connection later:

1. The client sends a ClientHello to the server with the ticket from the initial connection.
2. The server uses the ticket to "resume" a TLS connection.

For each scheme for generating and using the session ticket, select all statements that are true about the scheme. Assume that Diffie-Hellman TLS is used and that an on-path attacker has observed and recorded both the initial TLS handshake and a resumed handshake between the client and server.

Q10.1 (3 points) The ticket contains the TLS symmetric keys used in the initial connection. The resumed TLS session uses the symmetric keys present in the ticket.

■ (A) The attacker can trick the server into resuming a TLS connection that uses any symmetric keys of their choosing

■ (B) The attacker, if they compromise the server, can decrypt messages from the initial TLS connection

■ (C) The attacker, if they do not compromise the server, can decrypt messages from the initial TLS connection

☐ (D) None of the above

Solution: The server does not authenticate the ticket whatsoever, so any attacker can send an arbitrary ticket containing a key of their choosing to force the server to use symmetric keys of their choosing. Additionally, an attacker will see the keys in the ticket when the client sends the ticket in the resumed ClientHello, so they can decrypt the initial TLS connection regardless of whether they compromise the server.

Q10.2 (3 points) The server randomly generates a *new* set of symmetric keys and stores them in the ticket. The resumed TLS session uses the symmetric keys present in the ticket.

[✓ (G) The attacker can trick the server into resuming a TLS connection that uses any symmetric keys of their choosing]

☐ (H) The attacker, if they compromise the server, can decrypt messages from the initial TLS connection

☐ (I) The attacker, if they do not compromise the server, can decrypt messages from the initial TLS connection

☐ (J) None of the above

> **Solution:** Similar to the previous question, the server doesn't authenticate the ticket, so it can be tricked into using any arbitrary keys as part of an attacker-generated ticket. However, because the keys for the initial TLS connection are not preserved anywhere, the attacker will never be able to decrypt the initial TLS connection—it retains forward secrecy!

Q10.3 (3 points) The ticket contains the same symmetric keys as the initial connection. The ticket is then encrypted and MAC'd with a **ticket key** known only to the server. The server verifies and decrypts the ticket, and then the resumed TLS session uses the symmetric keys present in the ticket.

☐ (A) The attacker can trick the server into resuming a TLS connection that uses any symmetric keys of their choosing

[✓ (B) The attacker, if they compromise the server, can decrypt messages from the initial TLS connection]

☐ (C) The attacker, if they do not compromise the server, can decrypt messages from the initial TLS connection

☐ (D) None of the above

> **Solution:** Because the ticket includes a MAC, the attacker cannot trick the server into accepting arbitrary keys. Additionally, the keys are encrypted with the ticket key, so the attacker can no longer learn the symmetric keys, *if they don't compromise the server.* However, if the attacker does compromise the server, then they can learn the ticket key, decrypt the keys in the ticket, and decrypt the initial TLS connection.

Q10.4 (3 points) The ticket contains a session token randomly generated by the server, and the server stores the initial connection's symmetric keys associated with this token. When resuming a TLS session, it uses the received token to look up its associated keys, and resumes the TLS session uses those keys.

☐ (G) The attacker can trick the server into resuming a TLS connection that uses any symmetric keys of their choosing

■ (H) The attacker, if they compromise the server, can decrypt messages from the initial TLS connection

☐ (I) The attacker, if they do not compromise the server, can decrypt messages from the initial TLS connection

☐ (J) None of the above

Solution: The server cannot be tricked into using any arbitrary keys for a TLS connection, since it will only ever use keys that it has stored on the server itself. Even if the attacker can choose any arbitrary token, this does not correspond with any arbitrary keys. Additionally, this means that an attacker that sees the token can't learn any of the stored TLS keys from the initial connection, but it does mean that an attacker that compromises the server will learn the keys and, thus, be able to decrypt the initial TLS connection.

Q10.5 (2 points) EvanBot suggests storing parameters from a the initial TLS handshake in the ticket. Which of the following values would need to be included in the ticket for the client and server to be able to re-derive the same symmetric keys from the initial connection? Select all that apply.

■ (A) Client random

■ (B) Server random

■ (C) Premaster secret

☐ (D) The MAC of the dialog

☐ (E) None of the above

Solution: In the TLS handshake, the client random, server random, and premaster secret are used to derive the symmetric keys.

Q11 *DNSSEC* (18 points)

EvanBot wants to know the IP address of `security.csa.gov`.

Q11.1 (2 points) EvanBot performs a DNS lookup and receives the following records from the root name server:

#	Name	Type	Value
1	`.gov`	NS	`gov-ns.com`
2	`gov-ns.com`	A	`140.0.0.1`

EvanBot thinks the root name server might be broken, because it incorrectly endorsed `gov-ns.com` as a `.gov` name server. Is EvanBot right?

○ (A) Yes, because `gov-ns.com` is a `.com` name server

○ (B) Yes, because `gov-ns.com` is a domain, not a name server

○ (C) No, because `gov-ns.com` can answer both `.com` and `.gov` queries

[✓ (D) No, because the domain of the name server doesn't need to be in the name server's zone]

Solution: A name server is a machine on the network, and just like any other machine on the network, it has its own domain name and IP address. The name server's domain name does not need to match the zone that the name server is answering records for.

Q11.2 (3 points) Next, EvanBot receives the following records from the `.gov` name server:

#	Name	Type	Value
3	`csa.gov`	NS	`csa-ns.gov`
4	`csa.gov`	NS	`csa-ns.com`
5	`csa-ns.gov`	A	`150.0.0.1`
6	`csa-ns.com`	A	`160.0.0.1`
7	`csa.gov`	A	`170.0.0.1`

If EvanBot performs bailiwick checking, which A records will be stored into EvanBot's cache? Select all that apply.

[✓ (G) Record 5]

☐ (H) Record 6

[✓ (I) Record 7]

☐ (J) None of the above

Solution: According to bailiwick checking, EvanBot will only accept additional records whose domains are in the name server's zone. Since these records are coming from the `.gov` name server, EvanBot will only accept the records for `.gov`, and not the record for `.com`.

Q11.3 (4 points) Finally, EvanBot receives the following record from the `csa.gov` name server:

#	Name	Type	Value
8	`security.csa.gov`	A	`180.0.0.1`

Under which of these threat models can EvanBot be confident that this answer record is correct and not malicious? Assume that source port randomization is enabled, and EvanBot makes only one request. Assume that the recursive resolver is not malicious. Select all that apply.

☐ (A) There are no network attackers, but a name server might be malicious

☐ (B) No name servers are malicious, but there is a man-in-the-middle attacker present

☐ (C) No name servers are malicious, but there is an on-path attacker present

[✓ (D) No name servers are malicious, but there is an off-path attacker present]

☐ (E) None of the above

Solution: All of the listed threat models make DNS vulnerable, except for the off-path attacker. Because source port randomization is used and Kaminsky attacks are ruled out (since EvanBot makes only one request), it is almost impossible for the off-path attacker to guess both the query ID and the UDP source port correctly.

Q11.4 (2 points) EvanBot decides to perform the query with DNSSEC for extra security. EvanBot receives the following DNSSEC records from the root name server:

#	Type	Contents
9	DNSKEY	root's public key
10	DS	H(`.gov`'s public key)
11	RRSIG	Signature on record 10

EvanBot thinks the root name server might be broken. Is EvanBot right?

○ (G) Yes, because it never provided a signature on `.gov`'s public key

○ (H) Yes, because it never provided `.gov`'s public key

○ (I) No, because we can implicitly trust any records coming from the root name server

[✓ (J) No, because it provided enough information for us to trust the `.gov` name server]

Solution: In DNSSEC, each name server must provide its own public key and endorse the next name server by signing a hash of the next name server's public key. Record 9 provides the root name server's public key. Records 10 provides a hash of the next name server's public key, and record 11 signs the hash, which endorses the next name server.

Q11.5 (4 points) EvanBot receives the following DNSSEC records from the `.gov` name server:

#	Type	Contents
12	DNSKEY	`.gov`'s public key
13	DS	H(`csa.gov`'s public key)
14	RRSIG	Signature on record 13
15	A	IP address of `evil.csa.gov`
16	RRSIG	Signature on record 15

EvanBot validates that these records have a valid chain of trust back to root, but suspects that some of these records might have been added by an attacker to poison EvanBot's cache.

Which of the following attackers could have successfully poisoned EvanBot's cache with these records? Select all that apply.

☐ (A) A man-in-the-middle attacker who has not compromised any private keys

☐ (B) An on-path attacker who has not compromised any private keys

[✓ (C) A man-in-the-middle attacker who has compromised `.gov`'s private key]

[✓ (D) An on-path attacker who has compromised `.gov`'s private key]

☐ (E) None of the above

Solution: In DNSSEC, an attacker must compromise the name server's private key in order to create signatures for malicious records. A network attacker who has not compromised private keys will not be able to generate valid signatures for malicious records.

Q11.6 (3 points) EvanBot receives the following DNSSEC record from the `csa.gov` name server:

#	Type	Contents
17	RRSIG	Signature on record 8

How many signature verifications does EvanBot need to perform to ensure that the IP address in record 8 is correct?

○ (G) 0 ○ (I) 2 ○ (K) 4

○ (H) 1 [✓ (J) 3] ○ (L) More than 4

Solution: EvanBot needs to validate the signature in record 17 to validate that the final A record from `csa.gov` is correct. Then EvanBot needs to validate the signature in record 14 to validate that `csa.gov`'s public key is correct (as endorsed by `.gov`). Finally, EvanBot needs to validate the signature in record 11 to validate that `.gov`'s public key is correct (as endorsed by the root). This creates a chain of trust back to the root, so EvanBot can trust that the final A record is correct. In total, EvanBot validated 3 signatures.

Doodle Page

Nothing on this page will not affect your grade in any way.

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

Post-Exam Activity