CS 161 Fall 2022 — Introduction to 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).

Question:	1	2	3	4	5	6	7	8	9	10	11	Total
Points:	4	38	14	19	18	21	12	24	17	10	23	200

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)

Pre-exam activity (not graded, just for fun): Our GSIs are striking for better staffing and wages! If EvanBot went on strike, what would Bot demand?

Solution:

Q1 *Honor Code* (4 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.

SIGN your name: _____

Q2 *True/False* **(38 points)**

Each true/false is worth 2 points.

Q2.1 In GDB, you run `x/2wx &pancake` and see this output:

```
0xffff12a0: 0xffff13b2 0xffff14c4
```

`0xffff13b2` is the address of the variable `pancake`, and `0xffff12a0` is the value of the variable `pancake`.

○ (A) True ● (B) False

Solution: False. The GDB `x` command takes in an address and prints out memory at that address. In this output, `0xffff12a0` is the address, not the value.

Q2.2 If the `printf` function were updated so that `%n` (which writes values to memory) is no longer a valid format specifier, then format string vulnerabilities would no longer be possible.

○ (A) True ● (B) False

Solution: False. Other format specifiers like `%s` can still be used to leak values in memory.

Q2.3 On average, an attacker with 2^{32} tries will be able to brute-force an address when ASLR is enabled on a 32-bit system.

● (A) True ○ (B) False

Solution: True. In the worst case, the attacker could just try all 2^{32} addresses.

Q2.4 In a return-oriented programming (ROP) attack, the attacker needs to write machine code bytes in their exploit.

○ (A) True ● (B) False

Solution: False. ROP is designed to defeat non-executable pages, where any code written into memory wouldn't be executed anyway.

Q2.5 Return-oriented programming (ROP) is a technique used to circumvent stack canaries.

○ (A) True ● (B) False

Solution: False. ROP is used to circumvent non-executable pages by utilizing snippets of code (gadgets) already loaded in executable memory.

Q2.6 Enabling stack canaries only has minimal impact on the program's performance.

● (A) True ○ (B) False

Solution: True. Stack canaries add a few assembly instructions per function, and the performance impact is negligible on pretty much all modern systems.

Q2.7 PRNGs are deterministic, so they cannot be used to build an IND-CPA secure scheme.

○ (A) True ● (B) False

Solution: False. Consider stream ciphers, which use PRNGs (and introduce some randomness) to produce an IND-CPA secure scheme.

Q2.8 If Bitcoin used a non-cryptographic hash like MD5, users would be able to spend other users' coins.

○ (A) True ● (B) False

Solution: False. The hash algorithm makes it difficult to append to the blockchain. Signatures prevent users from spending other users' coins. Even if it were easy to append to the blockchain, it wouldn't be possible to forge signatures for other users.

Q2.9 JavaScript is usually sent by the server and executed in the browser.

● (A) True ○ (B) False

Solution: True. JavaScript is a client-side language because the code runs in the browser.

Q2.10 Cookies can be created by either the browser or the server.

● (A) True ○ (B) False

Solution: True. The server can send a response setting a cookie in the browser, or the browser could create cookies itself through Javascript or through user action.

Q2.11 Without cookies, it would be impossible for users to authenticate themselves in a request.

○ (A) True ● (B) False

Solution: False. A user could still send their credentials in every HTTP request that they make. This would be really tedious, though, which is why we use cookies to store and automatically send session tokens!

Q2.12 CSRF attacks allow an attacker to make requests that look like they're coming from the victim, but are actually being sent by the attacker.

○ (A) True ● (B) False

Solution: False. In a CSRF attack, the requests are actually being sent by the victim. The attacker tricks the victim into making the requests and attaching the victim's own cookies.

Q2.13 An attacker tricks the victim into visiting the attacker's website, which serves malicious JavaScript to the victim. This is an example of an XSS attack.

○ (A) True ● (B) False

Solution: False. The main vulnerability in XSS is attacker JavaScript being sent to the victim with the origin of a legitimate website. In this scenario, the JavaScript is still running the attacker's website's origin.

Q2.14 Phishing attacks rely on a malicious server pretending to be a legitimate website.

● (A) True ○ (B) False

Solution: True. In a phishing attack, the malicious server presents itself as a legitimate website, in order to trick the user into providing secret information to the attacker.

Q2.15 Layer 2 protocols are considered "end-to-end" protocols, unlike Layer 3 protocols, which may change from hop to hop as a packet is transmitted across the Internet.

○ (A) True ● (B) False

Solution: False. The Layer 3 protocol (which is only ever IP) is the end-to-end protocol, while the Layer 2 protocol may change across hops.

Q2.16 There is no difference between using UDP and using IP to send messages.

○ (A) True ● (B) False

Solution: False. UDP adds port numbers. Also, UDP and IP are on different layers; UDP is built on top of IP.

Q2.17 To send a message to someone on the same local network, you need to send the message to the router, which will forward the message to the recipient.

○ (A) True ● (B) False

Solution: False. If you're on the same local network as the recipient, you can directly send the message to them over the link layer. The router is mainly necessary for sending messages to other local networks.

Q2.18 A MITM attacker will always be able to successfully perform an ARP spoofing or DHCP spoofing attack on the first try.

● (A) True ○ (B) False

Solution: True. The MITM attacker can drop the legitimate response and replace it with their own malicious response with an incorrect IP/MAC mapping or configuration.

Q2.19 The checksum field in the TCP header stops random errors, but not malicious tampering.

● (A) True ○ (B) False

Solution: True. Checksums are not cryptographic functions (they don't have any secret keys involved), so they're mainly used to detect and fix random errors in transit.

Q2.20 (0 points) `EvanBot is a real bot.`

● (A) True ○ (B) False

Solution: `True.`

Q3 *EvanBART* (14 points)

Let's think about how fare gates are implemented in subway systems around the world! Select the security principle most relevant to each story.

Q3.1 (2 points) In Staten Island, there are only fare gates at the first stop. Some passengers avoid paying by walking to the second stop and getting on the train there.

- ○ (A) Security is economics
- ○ (B) Least privilege
- ○ (C) Separation of responsibility
- ○ (D) Defense in depth
- ○ (E) Consider human factors
- ● (F) Ensure complete mediation
- ○ (G) Know your threat model
- ○ (H) Detect if you can't prevent
- ○ (I) Don't rely on security through obscurity
- ○ (J) Design security in from the start

Solution: This is a failure of ensuring complete mediation; not all ways to enter the system are equally secured.

Q3.2 (2 points) All the transit systems in San Francisco agree to accept Clipper cards as payment in order to make it easier for passengers to pay the fare without having to worry about different types of tickets.

- ○ (A) Security is economics
- ○ (B) Least privilege
- ○ (C) Separation of responsibility
- ○ (D) Defense in depth
- ● (E) Consider human factors
- ○ (F) Ensure complete mediation
- ○ (G) Know your threat model
- ○ (H) Detect if you can't prevent
- ○ (I) Don't rely on security through obscurity
- ○ (J) Design security in from the start

Solution: Consider human factors: If every transit system had its own different ticketing system, paying fares would be more complicated, and people might be more incentivized to not pay the fare. By keeping the fare payment system simple, we're encouraging passengers to pay the fare.

Q3.3 (2 points) London's system was built and originally operated by many different companies. Even though the systems have been unified now, it's still hard to standardize fare gates at all the stations.

- ○ (A) Security is economics
- ○ (B) Least privilege
- ○ (C) Separation of responsibility
- ○ (D) Defense in depth
- ○ (E) Consider human factors
- ○ (F) Ensure complete mediation
- ○ (G) Know your threat model
- ○ (H) Detect if you can't prevent
- ○ (I) Don't rely on security through obscurity
- ● (J) Design security in from the start

Solution: Security was not designed in from the start, which makes it difficult to build security on top of an existing system.

Q3.4 (2 points) Caltrain doesn't have fare gates, but the conductor on the train will occasionally check that you have a valid ticket.

- ○ (A) Security is economics
- ○ (B) Least privilege
- ○ (C) Separation of responsibility
- ○ (D) Defense in depth
- ○ (E) Consider human factors
- ○ (F) Ensure complete mediation
- ○ (G) Know your threat model
- ● (H) Detect if you can't prevent
- ○ (I) Don't rely on security through obscurity
- ○ (J) Design security in from the start

Solution: Detect if you can't prevent. Even though there are no fare gates to prevent passengers from taking the train without a ticket, they try to detect passengers without tickets by checking tickets sometimes.

Q3.5 (2 points) The emergency exit doors in the New York City subway are supposed to make a loud noise if you open them to enter the station, but most stations leave the alarm off and hope that nobody notices.

- ○ (A) Security is economics
- ○ (B) Least privilege
- ○ (C) Separation of responsibility
- ○ (D) Defense in depth
- ○ (E) Consider human factors
- ○ (F) Ensure complete mediation
- ○ (G) Know your threat model
- ○ (H) Detect if you can't prevent
- ● (I) Don't rely on security through obscurity
- ○ (J) Design security in from the start

Solution: This is security through obscurity/Shannon's maxim. Assume that passengers know and can figure out that the alarm has been turned off.

Q3.6 (2 points) Los Angeles's subway system was losing $1 million per year in passengers not paying the fare. To avoid this, they spent $30 million installing new fare gates.

- ● (A) Security is economics
- ○ (B) Least privilege
- ○ (C) Separation of responsibility
- ○ (D) Defense in depth
- ○ (E) Consider human factors
- ○ (F) Ensure complete mediation
- ○ (G) Know your threat model
- ○ (H) Detect if you can't prevent
- ○ (I) Don't rely on security through obscurity
- ○ (J) Design security in from the start

Solution: This is a cost-benefit analysis (security is economics). Is the cost of the defense more than the cost of the attack here?

Q3.7 (2 points) Some cities have run studies on why passengers try to avoid the fare. Then, they try to solve the root causes of fare evasion by implementing reduced-fare programs, or making public transit free entirely.

- ○ (A) Security is economics
- ○ (B) Least privilege
- ○ (C) Separation of responsibility
- ○ (D) Defense in depth
- ○ (E) Consider human factors
- ○ (F) Ensure complete mediation
- ● (G) Know your threat model
- ○ (H) Detect if you can't prevent
- ○ (I) Don't rely on security through obscurity
- ○ (J) Design security in from the start

Solution: Think about your threat model–why are attacks happening in the first place? Can we stop attacks by addressing the root cause behind the attacks?

Author's editorial: Fare evasion on transit is an interesting case study where security meets ethics. By treating fare evaders as attackers and trying to implement defenses, we actually end up with inefficient and inequitable systems, where lots of effort is spent on unsuccessful security strategies that disproportionately hurt people from lower socioeconomic classes. (For example, transit cops are expensive to hire and are often racially biased when enforcing fares. Also, the companies behind transit cards like Clipper often have non-transparent privacy polices and a vested interest in making money off selling data about passengers' transit habits.) Making public transit free can be more cost-effective and equitable than trying to implement complicated and problematic security schemes.

Q4 *PACMAN* (19 points)

Relevant function definitions for this question:

- `char *fgets(char *s, int size, FILE *stream)` reads in at most one less than `size` characters from `stream` and stores them into the buffer pointed to by `s`. Reading stops after an EOF or a newline. If a newline is read, it is stored into the buffer. A terminating null byte (`'\0'`) is stored after the last character in the buffer.
- `char *gets(char *s)` reads a line from stdin into the buffer pointed to by `s` until either a terminating newline or EOF, which it replaces with a null byte (`'\0'`).

Consider a *PAC oracle*. An attacker can send an address and a PAC to the oracle, and the oracle will report whether the PAC is valid for the provided address.

Consider the following vulnerable C code, run on a little-endian, **64-bit** system with 16-bit pointer authentication codes (PACs):

```
1 void vulnerable() {
2     int authenticated;
3     char buf[24];
4     gets(buf);
5 }
```

Assume that the program will verify the PACs on all saved registers containing a memory address before they are restored.

Q4.1 (3 points) An attacker wants to overwrite the `authenticated` variable. In the worst case, how many calls to the PAC oracle would the attacker need to make in order to successfully perform this attack without crashing the program?

● (A) 0 ○ (B) 1 ○ (C) 2^{16} ○ (D) 2^{17} ○ (E) 2^{32} ○ (F) 2^{64}

Solution: 0

`buf` and `authenticated` are not pointers, so there's no need to guess any PACs to overwrite these local variables.

Q4.2 (3 points) An attacker wants to redirect execution to shellcode. In the worst case, how many calls to the PAC oracle would the attacker need to make in order to successfully perform this attack without crashing the program?

○ (A) 0 ○ (B) 1 ● (C) 2^{16} ● (D) 2^{17} ○ (E) 2^{32} ○ (F) 2^{64}

Solution: 2^{16} or 2^{17}

The attacker needs to brute-force the 16-bit PAC on the RIP in order to overwrite the RIP with a valid address of shellcode and then brute-force the 16-bit PAC on the SFP, since the SFP will be overwritten before the RIP by the `gets` function.

When this exam was administered, we didn't talk about any of the defenses against re-using a pointer value across two different addresses or domains. In practice, the SFP and RIP pointers would be authenticated with two different keys, and the addresses they're stored at would be included as part of the PAC to defend against trivially copying a pointer. Since students weren't expected to know this information, 2^{16} also received credit as an answer, if you reuse the same authenticated pointer for both the SFP and the RIP.

Assume that the attacker has learned that all addresses have PAC value of `0xFFFF`, and that shellcode exists at address `0x0000055CAFE00000`.

Assume that on a 64-bit system, the size of every variable type except pointers stay the same.

Write an exploit that executes shellcode without crashing the program.

Q4.3 (4 points) First, input a single byte repeated a number of times. Provide your answer in the template below (e.g. `'\x61' * 8`):

```
'\x______________' * ______________
```

Q4.4 (3 points) Then, input these 4 bytes:

● (A) `\x00\x00\xE0\xAF` ○ (C) `\xFF\xFF\xE0\xAF` ○ (E) `\xF0\xFF\xEF\xAF`

○ (B) `\xAF\xE0\x00\x00` ○ (D) `\xAF\xE0\xFF\xFF` ○ (F) `\xAF\xEF\xFF\xF0`

Q4.5 (3 points) Finally, input these 4 bytes:

○ (A) `\x5C\x05\x00\x00` ● (C) `\x5C\x05\xFF\xFF` ○ (E) `\x5C\xF5\xFF\x0F`

○ (B) `\x00\x00\x05\x5C` ○ (D) `\xFF\xFF\x05\x5C` ○ (F) `\x0F\xFF\xF5\x5C`

Solution: `\xFF * 36 + '\x00\x00\xE0\xAF\x5C\x05\xFF\xFF'`

At a high level, the exploit here is the same as a simple buffer overflow. We have to write enough garbage bytes to reach the RIP, and then overwrite the RIP with the address of shellcode. However, we also have to account for the fact that it's a 64-bit system and PACs are enabled.

We need 36 bytes of garbage in total. The first 24 bytes overwrite the buffer and the next 4 bytes overwrite the `authenticated` variable. Then the next 8 bytes overwrite the SFP (remember that pointers are 8 bytes in a 64-bit system).

However, since one of the values we're overwriting (the SFP) is a pointer, we need to make sure it has a valid PAC when it's checked during the function return. Since all PACs are `0xFFFF` here, we can ensure that a valid PAC is written back by using `\xFF` as our garbage byte.

The address of the RIP should be replaced with `0xFFFF055CAFE00000`, which is the address of shellcode with the PAC replacing the top 16 zero bits. We have to convert the address to little-endian before writing it into memory too.

Q4.6 (3 points) Suppose Line 4 is replaced with `fgets(buf, 24)`. Which of the following statements is true?

○ (A) An attacker can use the PAC oracle to execute shellcode, without crashing the program.

○ (B) An attacker can use the PAC oracle to execute shellcode, but may cause the program to crash while trying.

○ (C) An attacker can use the PAC oracle to crash the program, but not execute shellcode.

● (D) An attacker cannot use the PAC oracle to execute shellcode.

Solution: Even if the attacker is able to guess valid PACs, they can't write outside the buffer, so they won't be able to redirect program execution to shellcode.

Q5 *Hulk Leftover* **(18 points)**

Relevant function definitions for this question. (`fgets` is defined at the top of the previous question.)

- `size_t fwrite(const void *ptr, size_t size, size_t nmemb, FILE *stream)` writes `nmemb` items of data, each `size` bytes long, to the stream pointed to by `stream`, obtaining them from the location given by `ptr`.

In this question, your goal is to delete the `targets.txt` file, which Hulk uses to smash his targets!

- For your inputs, you may use `SHELLCODE` as a 16-byte shellcode that, if executed, will delete the `targets.txt` file.
- If needed, you may use standard output as `OUTPUT`, slicing it using Python syntax.
- You run GDB once, and discover that the address of the RIP of the `hulk` method is `0xffffcd84`.

```
1  void smash(char *buf) {
2      char name[20];
3      if (fgets(name, 20, stdin) == NULL) {
4          return;
5      }
6      FILE *targets = fopen("targets.txt", "w");
7      if (fwrite(name, sizeof(char), 20, targets) != 20) {
8          return;
9      }
10     name[strlen(name)] = '!';
11     printf("Target: %s", name);
12     fclose(targets);
13 }
14
15 void hulk(char *eyes) {
16     char anger[16];
17     smash(eyes);
18     fgets(anger, 28, stdin);
19 }
```

In this question, **stack canaries are enabled**, but all other memory safety defenses are disabled.

Suppose the program is paused at Line 2. Fill in the following stack diagram.

Stack

1
2
3
`anger`
4
RIP of `smash`
5
6

Q5.1 (2 points) Fill in blanks 1-3 on the stack diagram:

- ● (A) (1) - `eyes`; (2) - RIP of `hulk`; (3) - SFP of `hulk`
- ○ (B) (1) - `eyes`; (2) - SFP of `hulk`; (3) - RIP of `hulk`
- ○ (C) (1) - SFP of `hulk`; (2) - RIP of `hulk`; (3) - `eyes`
- ○ (D) (1) - RIP of `hulk`; (2) - SFP of `hulk`; (3) - `eyes`

Q5.2 (2 points) Fill in blanks 4-6 on the stack diagram:

- ○ (A) (4) - SFP of `smash`; (5) - `buf`; (6) - `name`
- ● (B) (4) - `buf`; (5) - SFP of `smash`; (6) - `name`
- ○ (C) (4) - `name`; (5) - SFP of `smash`; (6) - `buf`
- ○ (D) (4) - `name`; (5) - `buf`; (6) - SFP of `smash`

Solution:

Stack

`eyes` (argument to `hulk`)
RIP of `hulk`
SFP of `hulk`
`anger` (local variable of `smash`)
`buf` (argument to `smash`)
RIP of `smash`
SFP of `smash`
`name` (local variable of `hulk`)

Note: We forgot to add the canary to our stack diagram. This was a mistake on the exam, though it doesn't change the grading for Q5.1 and Q5.2 (the marked answers are the only ones that have the non-canary values in the right order).

Provide an input to each of the boxes below in order to delete `targets.txt`. For each box below:

- If any input would result in a working exploit, you must write "Anything".
- If you need garbage characters (i.e. any character would work), you must use the letter "A" repeated. For example, `'A'*5` would be 5 garbage characters.

Q5.3 (4 points) Provide a string value for `eyes` (argument to `hulk`):

Solution: Anything

Anything works here, because the `buf` argument is actually never used in `smash`.

Q5.4 (4 points) Provide an input to the `fgets` in `smash`:

Solution: `'A'*19`

This fills `name` with 19 garbage bytes, plus the null byte at the end. Then, line 10 will overwrite index 19 of `name` (the last byte of the buffer) with an exclamation point, which clobbers out the null byte.

Note that writing 20 or more garbage bytes also works, since `fgets` would just stop reading after the 19th byte.

Q5.5 (6 points) Provide an input to `gets` in `hulk`:

Solution:

```
SHELLCODE + OUTPUT[28:32] + 'A'*4 + '\x6c\xcd\xff\xff'
```

As a result of the null byte in `name` getting deleted, line 11 will print out the canary. To slice the canary, skip over the 8 characters of `Target:` that get printed, plus the 20 characters of `name` that get printed, to get bytes 28-31.

Once the canary is leaked, you can perform a standard buffer overflow attack, though note that the 16-byte shellcode will only fit at the start of the `anger` buffer here, since you can only write 28 bytes into memory.

To get the address of shellcode, start with the RIP of `hulk`, which ends in `0x84`. Then the SFP of hulk is at `0x80`. Then the canary is at `0x7c`. Then `anger` is a 16-byte buffer, so 16 bytes below `0x7c` is `0x6c`.

Note: This question should have said input to `fgets`, not `gets`. When grading, we allowed inputs longer than 28 bytes (i.e. allowed for the interpretation that the function was `gets` instead of `fgets`).

Q6 *WEP: Wasn't Even Protected* **(21 points)**

WEP (Wired Equivalent Privacy) was an insecure network protocol eventually replaced by WPA. In this question, we'll walk through an attack for tampering messages sent with WEP.

Assumptions:

- Alice is trying to send a message M to Bob, but Mallory can read and tamper with their communication.
- Alice and Bob share a key k that Mallory doesn't know.
- Enc and Dec denote AES-CTR encryption and decryption.

For confidentiality, WEP used a stream cipher. In this question, we'll choose AES-CTR with randomly-generated IVs as our stream cipher.

For integrity, WEP used the CRC checksum algorithm, which takes in one input and outputs a checksum on that input. CRC has the special property that for any bitstrings X and Y, $\mathsf{CRC}(X \oplus Y) = \mathsf{CRC}(X) \oplus \mathsf{CRC}(Y)$.

Alice sends $C_1 = \mathsf{Enc}(k, M)$ and $C_2 = \mathsf{CRC}(C_1)$. Mallory does not know M, and wants to tamper with this data so that Bob receives $M' = M \oplus 1$.

Q6.1 (4 points) Mallory should change C_1 to these two values, XORed together. Select exactly two options:

- ☐ (A) 0
- ■ (C) C_1
- ☐ (E) $\mathsf{CRC}(0)$
- ■ (B) 1
- ☐ (D) C_2
- ☐ (F) $\mathsf{CRC}(1)$

Solution: $C_1 \oplus 1$.

Remember that AES-CTR is a stream cipher, so flipping a bit of the ciphertext will also flip the bit of the plaintext. By flipping the last bit of the ciphertext, we've created the $\mathsf{Enc}(k, M \oplus 1)$. Bob will decrypt this and see $M \oplus 1 = M'$.

Q6.2 (4 points) Mallory should change C_2 to these two values, XORed together. Select exactly two options:

☐ (A) 0 ☐ (C) C_1 ☐ (E) $\mathsf{CRC}(0)$

☐ (B) 1 ■ (D) C_2 ■ (F) $\mathsf{CRC}(1)$

Solution: $C_2 \oplus \mathsf{CRC}(1)$

Substituting in the definition of C_2, this is equal to $\mathsf{CRC}(C_1) \oplus \mathsf{CRC}(1)$.

Using the special property from above, this is equal to $\mathsf{CRC}(C_1 \oplus 1)$.

This is a valid checksum/tag on the ciphertext from the previous part, so Bob will not detect the tampering.

Repeated from the previous part, for your convenience: Alice sends $C_1 = \mathsf{Enc}(k, M)$ and $C_2 = \mathsf{CRC}(C_1)$.

Now, suppose Mallory knows M, and wants to tamper with this data so that Bob receives M' (where M' is any value chosen by Mallory).

Q6.3 (5 points) Mallory should change C_1 to these values, XORed together. Select as many options as you need.

☐ (A) 1 ☐ (D) $\mathsf{CRC}(1)$ ■ (G) C_1

■ (B) M ☐ (E) $\mathsf{CRC}(M)$ ☐ (H) C_2

■ (C) M' ☐ (F) $\mathsf{CRC}(M')$ ☐ (I) $\mathsf{CRC}(0)$

Solution: $C_1 \oplus M \oplus M'$

When Bob decrypts this, he'll see M'.

We can prove this by treating AES-CTR as a stream cipher, which computes the ciphertext by XORing some pseudorandom bits B with the input (like a one-time pad). In equations, $C_1 = B \oplus M$.

Let's replace C_1 with $C_1 \oplus M \oplus M'$. Using the encryption formula, we know this is $(B \oplus M) \oplus M \oplus M'$. The Ms cancel to give $B \oplus M'$, which is exactly the encryption of M'.

Q6.4 (5 points) Mallory should change C_2 to these values, XORed together. Select as many options as you need.

- ☐ (A) 1
- ☐ (B) M
- ☐ (C) M'
- ☐ (D) $\mathsf{CRC}(1)$
- ■ (E) $\mathsf{CRC}(M)$
- ■ (F) $\mathsf{CRC}(M')$
- ☐ (G) C_1
- ■ (H) C_2
- ☐ (I) $\mathsf{CRC}(0)$

Solution: $C_2 \oplus \mathsf{CRC}(M) \oplus \mathsf{CRC}(M')$

This is equivalent to $\mathsf{CRC}(C_1 \oplus M \oplus M')$, a valid tag on the modified value from the previous part.

Applying the special property twice: $\mathsf{CRC}(C_1 \oplus M \oplus M') = \mathsf{CRC}(C_1) \oplus \mathsf{CRC}(M \oplus M') = \mathsf{CRC}(C_1) \oplus \mathsf{CRC}(M) \oplus \mathsf{CRC}(M')$.

Finally, note that $C_2 = \mathsf{CRC}(C_1)$ to get the desired result.

Q6.5 (3 points) Which of these modifications, if made individually, would stop the attack from the previous two subparts? Select all that apply.

- ■ (A) Use AES-CBC instead of AES-CTR
- ☐ (B) Use the MAC-then-encrypt pattern instead of the encrypt-then-MAC pattern
- ■ (C) Use HMAC instead of CRC
- ☐ (D) None of the above

Solution: Using AES-CBC would stop Mallory from changing the ciphertext to any message of her choosing.

If Alice used MAC-then-encrypt, she would send $C_1 = \mathsf{Enc}(k, M \| \mathsf{CRC}(M))$. Mallory could still tamper with the first part of the ciphertext to change M, then tamper with the second part of the ciphertext to change the tag.

Using HMAC would stop tampering.

Q7 *Hello, Is This EvanBot?* **(12 points)**

One day, you receive a mysterious message M from someone claiming to be EvanBot. In each subpart, select whether you're able to verify whether this message actually came from EvanBot.

Assumptions:

- Prime p and generator g are public values known to everyone.
- Enc is an IND-CPA symmetric encryption scheme.
- EvanBot's private key has not been compromised.

Q7.1 (3 points) You receive $g^a \bmod p$.

Then, you choose some value b and send back $g^b \bmod p$.

Finally, you receive $\mathsf{Enc}(g^{ab} \bmod p, M)$.

○ (A) You can verify that M is from EvanBot.

○ (B) You can verify that M is from EvanBot, but only if a and b are randomly generated.

○ (C) You cannot verify that M is from EvanBot, because you cannot derive g^{ab}.

● (D) You cannot verify that M is from EvanBot, but not for the reason above.

Solution: This is a standard Diffie-Hellman key exchange, followed by sending an encrypted message using the shared secret. However, nothing about this key exchange validates EvanBot's identity; anybody could have sent you $g^a \bmod p$.

Because this is standard Diffie-Hellman, you can take $g^a \bmod p$ and raise it to the b power to derive $g^{ab} \bmod p$, so the third option is incorrect.

Q7.2 (3 points) You receive $g^a \bmod p$.

Then, you choose some value b and send back $g^b \bmod p$.

Finally, you receive M and $\mathsf{HMAC}(g^{ab} \bmod p, M)$. You verify that the HMAC is valid.

- ○ (A) You can verify that M is from EvanBot.
- ○ (B) You can verify that M is from EvanBot, but only if a and b are randomly generated.
- ○ (C) You cannot verify that M is from EvanBot, because you cannot derive g^{ab}.
- ● (D) You cannot verify that M is from EvanBot, but not for the reason above.

Solution: The HMAC could be generated by whoever initiated the Diffie-Hellman exchange with you. Just like in the previous part, there's nothing here associating the exchange with EvanBot.

Q7.3 (3 points) You receive a certificate for EvanBot's public key, signed by a trusted CA.

Then, you exchange the same messages as in the previous subpart:

You receive $g^a \bmod p$.

Then, you choose some value b and send back $g^b \bmod p$.

Finally, you receive $\mathsf{Enc}(g^{ab} \bmod p, M)$.

- ○ (A) You can verify that M is from EvanBot.
- ○ (B) You can verify that M is from EvanBot, but only if you already knew EvanBot's public key.
- ● (C) You cannot verify that M is from EvanBot, because other people can provide EvanBot's certificate too.
- ○ (D) You cannot verify that M is from EvanBot, because certificates should contain private keys, not public keys.

Solution: Certificates are public, and anybody could have requested a copy of EvanBot's certificate and provided it to you.

Certificates do not contain private keys, so the fourth option is incorrect.

Q7.4 (3 points) You receive a certificate for EvanBot's public key, signed by a trusted CA.

You receive $g^a \bmod p$ and a signature over $g^a \bmod p$. You successfully verify the signature with the public key in the certificate.

Then, you choose some value b and send back $g^b \bmod p$.

Finally, you receive $\mathsf{Enc}(g^{ab} \bmod p, M)$.

- ○ (A) You can verify that M is from EvanBot.
- ○ (B) You can verify that M is from EvanBot, but only if you already knew EvanBot's public key.
- ○ (C) You cannot verify that M is from EvanBot, because Diffie-Hellman is vulnerable to MITM attacks.
- ● (D) You cannot verify that M is from EvanBot, because M was never signed.

Solution: The key exchange portion is exactly how TLS verifies the server. The server sends a certificate so that you know its public key. Then, the server signs its half of the Diffie-Hellman exchange so that you know the person you're talking to knows the server's private key. The signature over the Diffie-Hellman exchange also guarantees that the exchange is not vulnerable to MITM attacks.

However, even though the key exchange guarantees that you're talking to EvanBot, the message itself was never signed, so someone could tamper with the encryption and cause you to decrypt a different M (that was not from EvanBot).

Q8 *BotFlix* (24 points)

EvanBot wants to watch movies on BotFlix without paying, so EvanBot tries to hack into CodaBot's BotFlix account.

In this entire question, BotFlix does not perform any input sanitization.

The BotFlix database contains a `users` table with three string fields: `username`, `password_hash`, and `salt`.

There are U different users, P different possible passwords, and S different possible salts.

Users log in by making a GET request to `https://www.botflix.com/login?username=X&password=Y`, replacing `X` and `Y` with their username and password. The server first executes a query on the database using `username` inputted by the user:

```
db.Execute(fmt.Sprintf("SELECT * FROM users WHERE username = '%s'", username))
```

If this query returns 0 records or more than 1 record, the server displays all of the records returned by the query for debugging purposes.

If this query returns exactly 1 record, the server computes $\mathsf{H}(\texttt{password} \| \texttt{salt})$, where H is a slow, cryptographic hash, `password` is inputted by the user, and `salt` is from the database. If the result matches the hash from the database, the server authenticates the user. Otherwise, the server displays "Incorrect password."

In the next two subparts, EvanBot wants to perform an **online** brute-force attack to learn only CodaBot's password.

Q8.1 (2 points) How many GET requests does EvanBot need to submit?

- ○ (A) 0
- ● (D) P
- ○ (G) US
- ○ (B) 1
- ○ (E) UP
- ○ (H) UPS
- ○ (C) U
- ○ (F) PS
- ○ (I) Not possible

Solution: EvanBot submits one login request for each password.

Q8.2 (2 points) How many hashes does EvanBot need to compute?

- (●) (A) 0
- ( ) (B) 1
- ( ) (C) U
- ( ) (D) P
- ( ) (E) UP
- ( ) (F) PS
- ( ) (G) US
- ( ) (H) UPS
- ( ) (I) Not possible

Solution: In an online brute-force attack, the computation is done by the server, not the attacker; EvanBot doesn't compute any hashes at all.

In the next three subparts, EvanBot wants to perform an **offline** brute-force attack to learn CodaBot's password, but does not have access to the full list of password hashes and salts.

Q8.3 (5 points) Assume that EvanBot is not in the `users` table, but PintoBot and CodaBot are.

Which of the following usernames could EvanBot submit to help with this attack? Select all that apply.

- [ ] (A) `CodaBot`
- [■] (B) `CodaBot' OR 1=1--`
- [■] (C) `EvanBot' OR 1=1--`
- [■] (D) `PintoBot' OR username='CodaBot`
- [ ] (E) `EvanBot' OR username='CodaBot`
- [ ] (F) None of the above

Solution: To perform an offline brute-force attack, we need to first obtain CodaBot's salt and password hash. Note that if our injection returns exactly 1 record, we don't get to see the output of the SQL query, so we need to force the server to return more than 1 record.

`CodaBot` causes the server to return 1 record, so this isn't useful.

`CodaBot" OR 1=1--` and `EvanBot" OR 1=1--` both cause the server to return all the records.

`PintoBot" OR username="CodaBot` causes the server to return 2 records, including the one we care about.

`EvanBot" OR username="CodaBot` causes the server to return 1 record (because EvanBot isn't a valid user in the table), so this isn't useful.

Q8.4 (3 points) How many GET requests does EvanBot need to submit?

○ (A) 0	○ (D) P	○ (G) US
● (B) 1	○ (E) UP	○ (H) UPS
○ (C) U	○ (F) PS	○ (I) Not possible

Solution: EvanBot submits a single SQL injection query to leak CodaBot's password hash.

Q8.5 (2 points) How many hashes does EvanBot need to compute?

○ (A) 0	● (D) P	○ (G) US
○ (B) 1	○ (E) UP	○ (H) UPS
○ (C) U	○ (F) PS	○ (I) Not possible

Solution: Note that EvanBot is only trying to learn one password, not all of the passwords, so EvanBot only needs hash all the different passwords combined with CodaBot's salt to see which one is CodaBot's password.

The remaining subparts are independent of all the previous subparts. The remaining subparts are also independent of each other.

Q8.6 (3 points) Suppose EvanBot has already learned the value of CodaBot's current record in the `users` table.

EvanBot uses SQL injection to change CodaBot's record in the `users` table. EvanBot's goal is to be able to log in as CodaBot, using a password that EvanBot knows.

Is EvanBot able to achieve this goal?

● (A) Yes, by changing only `password_hash`

○ (B) Yes, but both `password_hash` and `salt` must be changed

○ (C) No, because a cryptographic hash is being used

○ (D) No, because a slow hash is being used

Solution: EvanBot can choose a new password, hash the new password with the existing salt, and replace `password_hash` in the database with the newly-hashed result. This is possible because anybody can compute hashes.

Q8.7 (3 points) Consider this modification: If the SQL query returns 0 records, or more than 1 record, the server displays "Invalid username: `X`", where `X` is the username inputted in the GET request.

What attack is this modification vulnerable to?

○ (A) CSRF

○ (B) Stored XSS

● (C) Reflected XSS

○ (D) Clickjacking

○ (E) TCP hijacking

○ (F) TCP spoofing

Solution: The server is now taking an arbitrary input string from the URL parameter and displaying it in the response. This is a reflected XSS vulnerability.

Q8.8 (4 points) Consider this modification: The login GET request is made to `http://www.botflix.com/login?username=X&password=Y` (HTTP instead of HTTPS).

Assume that there exists an on-path attacker between CodaBot and BotFlix, and the attacker observes CodaBot logging in.

With this modification, what attacks can the attacker perform? Select all that apply.

- ■ (A) Denial of service (prevent a user from logging in)
- ☐ (B) Learn CodaBot's password, but only after brute-forcing some hashes
- ■ (C) Learn CodaBot's password without computing any hashes
- ☐ (D) Learn every user's password without computing any hashes
- ☐ (E) None of the above

Solution: Without HTTPS, an on-path attacker can inject messages into the connection between CodaBot and BotFlix.

The attacker could perform a DoS attack by using TCP RST injection.

The attacker can learn CodaBot's password without any hash computation, as the password is sent in plaintext over TCP as part of the login request.

The attacker cannot learn every user's password. Other users' passwords will not be sent in the connection between CodaBot and BotFlix.

Q9 *Lost and Found* (17 points)

EvanBot is cleaning up the lecture hall after the last final exam, and finds a bunch of ID cards left behind! EvanBot designs a networking protocol to return the ID cards to their rightful owners.

1. EvanBot sends the SID (the ID number on the card) to all students.
2. The student with that SID sends their address to EvanBot.

Q9.1 (4 points) Assume that all students are in the same local network. Which of the following are valid ways to send the SID to all students? Select all that apply.

- ■ (A) Send each student a packet with the SID.
- ■ (B) Broadcast the SID over the local network.
- ☐ (C) In a local network secured by WPA, broadcast the SID, encrypted with the most recently derived PTK.
- ■ (D) In a local network secured by WPA, broadcast the SID, encrypted with the GTK.
- ☐ (E) None of the above

Solution: Broadcasting sends the message to all students on the local network. In WPA, everyone knows the GTK, but the PTK is different per student.

Q9.2 (4 points) Mallory is a malicious student. Which of the following must be true for Mallory to trick EvanBot into giving Mallory an ID card that doesn't belong to her? Select all that apply.

- ☐ (A) Mallory is an on-path attacker.
- ☐ (B) Mallory is able to fill in fake data in the sender field of a packet.
- ☐ (C) Mallory is able to fill in fake data in the recipient field of a packet.
- ■ (D) Mallory is able to send a packet to EvanBot before any legitimate packets.
- ☐ (E) None of the above

Solution: When EvanBot broadcasts the SID request, Mallory will see it, and can send her address back to EvanBot. Mallory doesn't need to spoof any packets; she can just pretend to be the student with that SID, since EvanBot doesn't know which SID maps to which student.

EvanBot wants to modify the scheme, with the following requirements:

- The legitimate student can still send their address to EvanBot.
- A student should not be able to learn other students' SIDs and addresses.
- An **on-path** attacker should not be able to learn any SIDs or addresses.
- An **off-path** attacker cannot trick EvanBot into associating an SID with the wrong address.

You can assume that:

- There are too many SIDs to list them all.
- H refers to a secure cryptographic hash.
- Enc refers to a semantically secure **public-key** encryption function.
- Everyone knows EvanBot's public key PK.

Q9.3 (4 points) In Step 1, what should EvanBot send to all students?

○ (A) SID

● (B) $\mathsf{H}(\text{SID})$

○ (C) $\mathsf{Enc}(PK, \text{SID})$

Solution: Sending the SID in plaintext would leak it to other students.

If we sent the SID encrypted with EvanBot's public key, only EvanBot would be able to decrypt the SID. Other students would not be able to check whether their SID matched the one sent out. (If we were using a deterministic encryption scheme, other students might be able to encrypt the message and check that the encryption matches, but the question uses RSA-OAEP, which introduces randomness into the encryption.)

A hash is the best option here. Students can hash their own SID and check whether it matches the hash that EvanBot sent out. The hash doesn't leak any information about other students' SIDs.

Q9.4 (5 points) In Step 2, which value should the student send back to EvanBot?

- ○ (A) $(\text{SID}, \text{address})$
- ● (B) $\text{Enc}(PK, (\text{SID}, \text{address}))$
- ○ (C) $(\text{H}(\text{SID}), \text{address})$
- ○ (D) $\text{Enc}(PK, \text{address})$
- ○ (E) $\text{Enc}(PK, \text{SID})$
- ○ (F) $\text{H}((\text{SID}, \text{address}))$

Solution: First, we need to think of how to send the SID. (Note that if we didn't send an SID, the attacker could just send back their own address.)

Sending the SID in plaintext would leak it to an on-path attacker.

Sending the hash of the SID leaves the scheme vulnerable to spoofing attacks. An attacker could spoof a response with the same hash that EvanBot sent out. Similarly, if we sent nothing about the SID at all, an attacker could spoof a response without knowing the SID.

Sending the SID encrypted with EvanBot's public key is the best option. An attacker cannot perform this encryption without knowing the correct SID.

Then, we need to think of how to send the address:

Sending the address in plaintext would leak it to an on-path attacker.

Sending the hash of the address doesn't allow EvanBot to receive the student's address, since EvanBot can't reverse the hash.

Sending the address encrypted with EvanBot's public key is the best option. Only EvanBot can decrypt this and learn the actual address.

Q10 *Teaching Lonely (because) Strike* **(10 points)**

Each subpart is independent.

Q10.1 (5 points) Consider this modification to TLS: The exchange of MACs in the handshake is removed. Instead, the server signs every message they send with their private key. Also, the client generates a public/private key pair, and signs every message they send with their private key. No other modifications are made (i.e. no extra information is sent).

Select all true statements about this modified version of TLS.

- ■ (A) An attacker can tamper with client-to-server messages without being detected.
- ☐ (B) An attacker can tamper with server-to-client messages without being detected.
- ☐ (C) An attacker can replay messages from a past connection.
- ☐ (D) An attacker can replay messages from the current connection.
- ☐ (E) An attacker can impersonate the server.
- ☐ (F) None of the above

Solution: In the existing handshake, the server never learns the client's public key, so there is no way for the server to verify that messages are correctly signed by the client. Therefore, an attacker can tamper with messages sent by the client.

Q10.2 (5 points) Suppose the client and server already share a symmetric master key MK, not known to the attacker. For each connection, the client and server each increment a shared counter i, stored locally, and derive the premaster secret as $PS = \mathsf{HKDF}(MK, i)$. As before, the TLS encryption and MAC keys are derived from the client random R_B, server random R_S, and PS.

Which of the following elements of the TLS handshake are no longer needed and may be removed without affecting the security of TLS? Select all that apply.

- ■ (A) The client random value
- ■ (B) The server random value
- ■ (C) The server's certificate
- ☐ (D) The MAC exchange at the end of the handshake
- ☐ (E) The MACs over the TLS application data
- ☐ (F) None of the above

Solution: Neither the client nor server random values are needed anymore, since derivation of PS from a monotonically increasing counter i guarantees that the PS will always be unique per session, which is usually guaranteed by including a client and server random value in the handshake.

The server's certificate is no longer needed, since the client and the server already have a shared secret. Someone impersonating the server would not have this secret.

The MAC exchange is still needed, because someone could have tampered with some part of the handshake (e.g. consider an attacker tampering with the ServerHello and ClientHello that help the client and server agree on a cipher suite).

The MACs over the data are still needed to ensure integrity on the data sent in the TLS connection.

Q11 *Everyone Trusts EvanBot* **(23 points)**

CodaBot makes a DNSSEC request for `actually.isevanbotreal.com`. After the request, CodaBot's DNS cache is filled with the following records.

Each row lists a record type and who sent that record. For example, row 2 says that the root name server sent an A type record.

	Received From	Type
1	root name server	NS
2	root name server	A
3	root name server	DS
4	root name server	RRSIG
5	root name server	DNSKEY
6	`.com` name server	NS
7	`.com` name server	A
8	`.com` name server	DS
9	`.com` name server	RRSIG
10	`.com` name server	DNSKEY
11	`isevanbotreal.com` name server	A
12	`isevanbotreal.com` name server	RRSIG
13	`isevanbotreal.com` name server	DNSKEY

Q11.1 (3 points) Which record contains the IP address of `actually.isevanbotreal.com`?

- ○ (A) 1
- ○ (B) 2
- ○ (C) 3
- ○ (D) 4
- ○ (E) 5
- ○ (F) 6
- ○ (G) 7
- ○ (H) 8
- ○ (I) 9
- ○ (J) 10
- ● (K) 11
- ○ (L) 12
- ○ (M) 13
- ○ (N) None

Solution: A type records contain mappings of domains to IP addresses. The IP address of `actually.isevanbotreal.com` is returned by the `isevanbotreal.com` name server.

Q11.2 (3 points) Which record contains the IP address of `www.isevanbotreal.com`?

○ (A) 1	○ (E) 5	○ (I) 9	○ (M) 13
○ (B) 2	○ (F) 6	○ (J) 10	● (N) None
○ (C) 3	○ (G) 7	○ (K) 11	
○ (D) 4	○ (H) 8	○ (L) 12	

Solution: We never requested the IP address of the `www.isevanbotreal.com` server, so this IP address is not in any of the records.

Be careful to distinguish the `isevanbotreal.com` name server from the `www.isevanbotreal.com` web server. Record 7 returns the IP address of the name server, not the IP address of the web server.

Q11.3 (3 points) What would happen to the query if record 2 was not sent?

● (A) CodaBot is unable to learn the IP address of `actually.isevanbotreal.com`.

○ (B) CodaBot learns the IP address of `actually.isevanbotreal.com` but cannot verify its integrity.

○ (C) CodaBot learns the IP address of `actually.isevanbotreal.com`, and is able to verify its integrity.

Solution: Without record 2, CodaBot would not be able to locate the `.com` name server.

Q11.4 (4 points) Suppose the `.com` name server sends another new record, with a signature on that record. Which records in the cache are needed to verify this new record? Select all that apply.

- ☐ (A) 1
- ■ (E) 5
- ☐ (I) 9
- ☐ (M) 13
- ☐ (B) 2
- ☐ (F) 6
- ■ (J) 10
- ☐ (N) None
- ■ (C) 3
- ☐ (G) 7
- ☐ (K) 11
- ■ (D) 4
- ☐ (H) 8
- ☐ (L) 12

Solution: We need the `.com` name server's public key (record 10) to verify the integrity of the signature.

Then, we need to make sure that the public key in record 10 is trustworthy. The root sent a DS record (3) and a signature on that DS record (4), which endorse the public key in record 10. Finally, we need the root's public key (5) to verify the signature in record 4.

We implicitly trust the root's public key, so this completes the chain of trust from the new record back to the root.

Recall that in DNS, everyone implicitly trusts the root name server. For the rest of the question, suppose that the root name server is no longer trusted. Instead, the `isevanbotreal.com` name server's public key is known and trusted by everybody.

Q11.5 (3 points) Which of the following name servers, if compromised, would allow the attacker to trick CodaBot into accepting a malicious IP address in the final answer record? Select all that apply.

- ☐ (A) root
- ■ (C) `isevanbotreal.com`
- ☐ (B) `.com`
- ☐ (D) None of the above

Solution: Because the `isevanbotreal.com` name server is now implicitly trusted, we only need a chain of trust back to `isevanbotreal.com`, not a chain of trust all the way back to root.

Q11.6 (4 points) Suppose that CodaBot makes another DNSSEC request for `fa22.cs161.org`. CodaBot's cache now contains additional records from the `.org` and `cs161.org` name servers.

Which of the following DS records could the `isevanbotreal.com` name server send to help CodaBot verify the integrity of the IP address of `fa22.cs161.org`? Assume that an RRSIG record over the DS record is also sent. Select all that apply.

■ (A) DS record with hash of root's public key

☐ (B) DS record with hash of the `.com` name server's public key

■ (C) DS record with hash of the `.org` name server's public key

■ (D) DS record with hash of `cs161.org` name server's public key

■ (E) None of the above

Solution: We need to create a new chain of trust from `cs161.org` back to the root of trust, which is now `isevanbotreal.com`. To do so, we need `isevanbotreal.com` to endorse one of the name servers in the original chain (root, `.org`, `cs161.org`).

An alternative answer for this question involved reasoning that `fa22.cs161.org` is not in bailiwick for `isevanbotreal.com`, so it can't issue any records that will be accepted by other users. "None of the above" also received full credit as a solution.

Q11.7 (3 points) CodaBot suggests that instead of starting by contacting the root name server, all future DNS requests should now start by contacting the `isevanbotreal.com` name server. Then, the `isevanbotreal.com` can redirect users to the correct top-level domain name server (e.g. the `.org` name server).

Which of the following features in existing DNS would need to be changed to support this modification?

○ (A) Source port randomization

● (B) Bailiwick checking

○ (C) ID field

○ (D) Using UDP instead of TCP

Solution: Under bailiwick checking, the `isevanbotreal.com` name server would only be authorized to provide records for subdomains of `isevanbotreal.com`, not other domains like the domain of the `.org` name server.

Clarifications

We aren't issuing clarifications during the exam. However, if you have any clarification questions, or made any assumptions while solving a question, you can note it here and we'll account for it during grading.

Doodle

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