Paxson Spring 2017 — CS 161 Computer Security — Final Exam

Solutions updated May 2021 by CS161 SP21 course staff.

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

I am aware of the Berkeley Campus Code of Student Conduct and acknowledge that any academic misconduct on this exam will be reported to the Center for Student Conduct and may lead to a "F"-grade for the course.

SIGN your name: _____

PRINT your class account login: `cs161-`_____ and SID: _____

Your TA's name: _____

Number of exam of person to your left: _____

Number of exam of person to your right: _____

You may consult two sheets of notes (each double-sided). You may not consult other notes, textbooks, etc. Calculators, computers, and other electronic devices are not permitted. Please write your answers in the spaces provided in the test.

You have 180 minutes. There are 9 questions, of varying credit (600 points total). The questions are of varying difficulty, so avoid spending too long on any one question. Parts of the exam will be graded automatically by scanning the **bubbles you fill in**, so please do your best to fill them in somewhat completely. Don't worry—if something goes wrong with the scanning, you'll have a chance to correct it during the regrade period.

If you have a question, raise your hand, and when an instructor motions to you, come to them to ask the question.

Do not turn this page until your instructor tells you to do so.

Problem 1 ***True/False*** **(80 points)**

For each of the following, FILL IN THE BUBBLE next to **True** if the statement is correct, or next to **False** if it is not. Each correct answer is worth 4 points. Incorrect answers are worth 0 points. Answers left blank are worth 1 point.

(a) Thanks to strong cryptography, a TLS connection to your bank is secure even if their web server's TCP/IP implementation has a buffer overflow vulnerability.

○ **True** ● **False**

Solution: False. TLS secures communication between you and your bank so that nobody except you and the bank can read or modify messages sent in the connection. However, TLS does not secure the bank's server. If the bank's server is vulnerable, someone can hack into the bank and read or modify your communications.

(b) Thanks to strong cryptography, a TLS connection to your bank is secure even if your home router's TCP/IP implementation has a buffer overflow vulnerability.

● **True** ○ **False**

Solution: True. A key property of TLS is how it provides *end-to-end* security: two systems can communicate using TLS without having to trust any of the intermediaries that forward their traffic. Thus, even if an attacker completely pwns your home router, the worst they can do to you is deny you service to your bank.

Note that this answer is different from the last question because the home router is not one of the ends in the end-to-end secure communication. The two ends are your computer/browser and the bank server.

(c) To protect against Kaminsky blind spoofing attacks requires servers to implement a new version of the DNS protocol.

○ **True** ● **False**

Solution: False. There is no new version of the DNS protocol that defends against the Kaminsky attack. (DNSSEC does defend against the Kaminsky attack, but it's more like a variation of the DNS protocol than a new version of DNS.)

(d) Using DNSSEC to resolve `example.com` guarantees authenticity and integrity on subsequent HTTP connections to `example.com`, but not confidentiality.

○ **True** ● **False**

Solution: False. DNSSEC provides authenticity and integrity for *DNS results*, but *not* for any subsequent use of those results. The subsequent connections will have to achieve security separately, such as by using TLS.

In other words, DNSSEC helps you be confident that you have the right IP address of `example.com`, but it does not help secure any actual communication with `example.com`.

(e) A properly configured firewall can prevent any DDoS attack from disrupting the ability of remote users to access your network.

○ **True** ● **False**

Solution: False. A firewall has limited computational resources, and a DDoS attack could overwhelm the firewall.

(f) Using a prepared statement to feed user input to an SQL query ensures that nothing the user enters will be treated as an SQL command.

● **True** ○ **False**

Solution: True. Prepared statements (parameterized SQL) pre-compiles a query before substituting user input, so nothing the user enters will be interpreted as SQL syntax.

(g) VPN can enable you to safely connect to your company when using an untrusted public WiFi network.

● **True** ○ **False**

Solution: True. Recall that with a VPN, you encrypt your requests and send them to the VPN, who decrypts them and forwards them to the company for you. The VPN then encrypts the company's reply and sends it back to you. Even if you're on an untrusted wifi network, your communications with the company will be safely encrypted.

(h) When configuring a firewall, it's safer to use a whitelisting approach than it is to use a blacklisting approach.

● **True** ○ **False**

Solution: True. Whitelisting is default-deny, so any unspecified or unfamiliar input will be rejected by default. This is safer than blacklisting, which is default-

allow and would allow any unspecified or unfamiliar input.

(i) A malicious website can execute a successful clickjacking attack even if the victim website uses HTTPS and the user's browser correctly implements the same origin policy.

● **True** ○ **False**

Solution: True. HTTPS defends against network attackers and secures communications between the victim and the malicious website, but it does not defend against the malicious website sending malicious data to the victim. The same-origin policy prevents two different websites from communicating with each other, but in this scenario, there's only one relevant website (the malicious website).

As an example, the malicious website might be a file downloading website that has a legitimate download button and a second, larger, more visible fake download button. If the user clicks on the fake download button, the clickjacking attack has been executed, and HTTPS and the same-origin policy did not defend against the attack.

(j) A secure hash function will not produce any collisions.

○ **True** ● **False**

Solution: False. Hashes map infinitely many input to finitely many outputs– any arbitrary-length string can be hashed, but the output is restricted to the finite space of fixed-length strings. Since the input space is larger than the output space, the hash function must produce collisions. (Think pigeonhole principle.)

A secure hash function instead has the guarantee that it is computationally infeasible to find a collision. The collisions exist, but it's impossible to find one in any reasonable amount of time.

(k) Recall that secure-cookies are cookies which the browser will only transmit over HTTPS connection. Using HTTPS and secure-cookies is one way to prevent clickjacking attacks.

○ **True** ● **False**

Solution: False. As explained above, HTTPS does not defend against clickjacking attacks. Clickjacking does not necessarily need cookies to succeed, so secure cookies aren't a defense against clickjacking. The scenario above (fake

download button) doesn't involve cookies at all, for example.

(l) Suppose Alice has signed up for text-message two factor authentication on `bank.com`. If `bank.com` randomly generates a long number (e.g., a 16-digit number) for its 2FA codes and an attacker doesn't hijack Alice's phone number, then Alice's `bank.com` account is secure against phishing attacks.

○ **True** ● **False**

Solution: False. Consider the following attack: Alice clicks on a malicious website impersonating `bank.com` and types her password into the website. The attacker opens the real `bank.com` and enters Alice's password. The legitimate website will send Alice a code in response to the attacker's login request. At the same time, the malicious website now asks Alice to enter her 2FA code. Alice sees the code on her phone and enters it into the malicious website. The attacker now has Alice's code and can enter it into the real `bank.com`, completing the login as Alice.

(m) For AES-CBC encryption, the IV does not need to be kept secret.

● **True** ○ **False**

Solution: True. The IV is sent in plaintext as part of the ciphertext.

(n) For AES-CTR encryption, the IV does not need to be kept secret.

● **True** ○ **False**

Solution: True. The IV is sent in plaintext as part of the ciphertext.

(o) If all messages are the same length and a message is never repeated, then it is secure to re-use the same one-time-pad for encryption.

○ **True** ● **False**

Solution: False. Reusing the one-time pad leaks information. For example, consider pad k and two messages m_1 and m_2 that are the same length and never repeated. If the attacker sees the encryption of the two messages with the same key, $m_1 \oplus k$ and $m_2 \oplus k$, then they can deduce the XOR of the two messages: $(m_1 \oplus k) \oplus (m_2 \oplus k) = m_1 \oplus m_2$.

(p) To securely store user passwords, a server should use AES to encrypt each user's password and only store the ciphertexts in its database.

○ **True** ● **False**

> **Solution:** False. AES (without any chaining mode) is deterministic, so an attacker can see if two users have the same password (the resulting ciphertext will the same). AES also only supports fixed-length inputs, so it would not be very useful unless the website forced passwords to be exactly a certain number of characters long.
>
> Password hashing (with random salts) is a better solution for securely storing user passwords.

(q) If *Website A* loads a website from another domain (*Website B*) inside of an iframe, the same origin policy prevents Javascript from *Website A* from accessing any of the other website's content in the iframe.

● **True** ○ **False**

> **Solution:** True. Website A and Website B have different origins, so by the same-origin policy, the inner and outer frames cannot access each other.

(r) A certificate authority that issues a TLS certificate for `example.com` can also passively decrypt TLS traffic to `example.com`.

○ **True** ● **False**

> **Solution:** The certificate authority only knows `example.com`'s public key, not its private key, which it would need for passive decryption. A certificate binds a public key to an identity (in this case the domain name `example.com`).

(s) Consider a worm that spreads by each infected instance uniform randomly selecting a 32-bit IP address. We would expect the worm to initially spread exponentially fast, but then slow down its spread during the later part of its propagation.

● **True** ○ **False**

> **Solution:** True. The initial exponential spread is because the increased number of infected computers also means more computers are working to infect other computers. (The first computer infects 2 other computers. Then these 2 computers each infect 2 more computers. Then these 4 computers each infect 2 other computers, and so on.)
>
> The worm later slows down because it runs out of computers to infect, so the rate of new computers getting infected (and spreading the infection) decreases.

(t) The Slammer worm spread extra-fast because each infected instance of the worm kept increasing its scanning speed.

○ **True** ● **False**

Solution: False. The Slammer worm was extra-fast because the malicious code of the worm was extremely small. The entire worm fit in one UDP packet, so it took only a single UDP packet transmission to infect another computer.

Problem 2 ***Multiple Choice*** **(74 points)**

(a) (6 points) Suppose an attacker steals the private key of a website that uses TLS, and remains undetected. What can the attacker do using the private key? **Mark ALL that apply.**

- ● Decrypt recorded past TLS sessions that used RSA key exchange.
- ● Successfully perform a MITM attack on future TLS sessions.
- ○ Decrypt recorded past TLS sessions that used Diffie–Hellman key exchange.
- ○ None of these.

Solution: RSA TLS does not have forward secrecy, so an attacker who steals the private key can decrypt recorded past RSA TLS sessions. (The attacker uses the private key to decrypt the premaster secret sent over the handshake. Then they use the decrypted premaster secret and the two random nonces sent over the handshake to derive the symmetric keys. Then they can use the symmetric keys to decrypt communications.)

Diffie-Hellman TLS does have forward secrecy, so an attacker who steals the private key cannot decrypt recorded past Diffie-Hellman TLS sessions. (The premaster secret $g^{ab} \bmod p$ is never sent over the handshake in Diffie-Hellman TLS. Only $g^a \bmod p$ and $g^b \bmod p$ are sent, and an attacker who knows the private key and those two values cannot learn the premaster secret because the discrete log problem is hard.)

The attacker can perform a MITM attack on future TLS sessions. In RSA TLS, the attacker decrypts the premaster secret as it's sent over the handshake, as described above. In Diffie-Hellman TLS, the attacker performs a Diffie-Hellman MITM to force both the client and server to derive premaster secrets that the attacker knows. In both cases, the attacker now knows the premaster secret and can use it with the nonces (sent in plaintext over the handshake) to derive the symmetric keys. With the symmetric keys, the attacker can decrypt and modify communications.

(b) (6 points) DNSSEC provides which of the following security properties for DNS responses? **Mark ALL that apply.**

- ○ Confidentiality
- ● Integrity
- ○ None of these
- ● Authentication
- ○ Availability

Solution: DNSSEC provides integrity and authenticity on responses because it signs all DNS records. (Recall that signatures provide integrity and authenticity.)

DNSSEC does not provide confidentiality because it does not encrypt DNS records. (DNS records are public information anyway, so confidentiality isn't needed.)

DNSSEC does not provide availability. A MITM attacker could always drop packets, and there's nothing DNSSEC can do to defend against that.

(c) (8 points) "Mixing program control and user data" is a class of vulnerabilities where a program/application accidentally treats user input as code and executes it. Which of the following attacks exploit this class of vulnerabilities? **Mark ALL that apply.**

- ● Buffer overflows
- ○ CSRF
- ● SQL Injection
- ○ None of these
- ● Stored XSS
- ● Reflected XSS
- ○ Clickjacking

Solution: In buffer overflows, user input is treated as C code. In SQL injection, user input is treated as SQL code. In XSS (both types), user input is treated as JavaScript code.

In CSRF, the attacker tricks the victim into making a malicious request with the user's cookies attached. User input isn't ever treated as code during the attack.

In clickjacking, the attacker tricks the victim into clicking an unintended link or button. User input isn't ever treated as code during the attack.

(d) (6 points) To verify that she is visiting the correct website, Alice is told to make sure to check that the URL in the browser's address bar is the URL she actually wants to visit. Which of the following statements are true? **Mark ALL of the following statements that apply.**

- ○ Of relevance for this situation is the principle of Least Privilege
- ● Of relevance for this situation is the principle of Consider Human Factors
- ○ This will help Alice defend herself against CSRF attacks
- ○ This will help Alice defend herself against some DNS spoofing attacks

- ● This will help Alice defend herself against some phishing attacks
- ○ None of these

Solution: The relevant security principle is considering human factors–visiting URLs like `g00gle.com` that are maliciously designed to look like legitimate URLs is a human mistake. Least privilege is not relevant, because Alice isn't given any special privileges in this scenario.

Checking the URL carefully prevents against phishing attacks, where the attacker tries to impersonate a legitimate website (possibly by using a similar URL).

Checking the URL does not directly prevent CSRF attacks. It can help defend against some CSRF attacks (because they often require the victim to visit an attacker's website to generate the forged request), but it does not prevent the main vulnerability of CSRF attacks, which is that the server can't distinguish between legitimate and forged requests from the victim.

DNS spoofing is a networking attack and is unrelated to carefully checking URLs (a web security defense).

(e) (6 points) Alice is trying to visit `maps.google.com` and neither her machine nor her local resolver have any entries in their DNS caches. In the following, assume that `google.com` subdomains use HTTPS and are on the predefined HSTS (HTTP Strict Transport Security) list in Alice's browser. You do not need to worry about attacks on availability, nor attacks based on stealing private keys, malware infections, or obtaining a fraudulent `google.com` certificate. **Mark ALL that apply.**

- ○ For DNSSEC to work securely, the root and .com zones will need to sign their NS and glue/additional records.
- ○ For DNSSEC to work securely, the root and .com zones will need to encrypt their NS and glue/additional records.
- ● Because google.com subdomains are on the predefined HSTS list, Alice's visit to `maps.google.com` is secure against DNS spoofing attacks.
- ● Because google.com subdomains are on the predefined HSTS list, Alice's visit to `maps.google.com` is secure against MITM attacks.
- ● Because google.com subdomains are on the predefined HSTS list, Alice's visit to `maps.google.com` is secure against *ssl-strip* attacks.
- ○ None of these apply.

Solution: Many students found this problem difficult.

DNSSEC does not protect NS and glue/additional records in any manner. It doesn't need to because its focus is on assuring the correctness of the final result (i.e., *object security*), not *how* the client gets the result.

The presence on the HSTS list means that the visit to `maps.google.com` will definitely occur using HTTPS. Given that, no DNS spoofing attack can fool Alice into visiting a different site unless the attacker has obtained a fraudulent certificate (which is ruled out in the framing of the problem).

Similarly, the guaranteed use of HTTPS secures the Alice's visit against MITM attacks.

Finally, *ssl-strip* attacks rely on the user's visit initially using HTTP rather than HTTPS. The use of HSTS will prevent such an initial visit from occurring.

Note on object security (compared to channel security):

Channel security: secure the letter while it is in transit between post offices (but not while it is stored)

Object security: the letter has a digital signature to make sure it hasn't been modified (either in transit or in storage)

Depending on how it's used, signing/verifying helps with both defending against tampering while the data is in transit along a network link and tampering while the data is stored or processed on a particular machine. The signing/verifying procedure from the crypto section ensures some channel security because no one can tamper with the message without being detected (though you need some more encryption to guarantee confidentiality), as well as protection against modification while the data is in storage because you can use Alice's public key to verify that the message is actually from Alice. This didn't quite give us full object security in DNSSEC because we couldn't verify Alice's public key in the first place, which is why we create a chain of trust that ensures that the nameservers we're talking to aren't compromised and are providing the correct public keys.

Notice that if you only use channel security between each pair of post offices, and a letter goes through multiple post offices before reaching its destination, there are two threats: the letter might be tampered with while it is being processed (or stored) at a particular office, and one of the post offices might be malicious and might replace the letter with an entirely different letter. Channel security doesn't protect against either of those threats. Object security does.

(f) (8 points) Gandalf is surfing the web and visits the URL `http://gondor.berkeley.edu`. Assume that neither his machine nor his local resolver have any entries in their DNS caches, and that `berkeley.edu` is the authoritative name server for all `berkeley.edu`

subdomains. Assuming global deployment and use of DNSSEC, and that DNS zones use Key Signing Keys (KSKs) and Zone Signing Keys (ZSKs), which of the following are True? **Mark ALL that apply.**

- ○ Gandalf's machine can use the `berkeley.edu` KSK to encrypt the query it sends to the `berkeley.edu` DNS server.
- ○ `berkeley.edu`'s ZSK will be signed by the root's KSK.
- ● `berkeley.edu`'s ZSK will be signed by `berkeley.edu`'s KSK.
- ● Gandalf's machine will receive a final A record for `gondor.berkeley.edu` that is signed with `berkeley.edu`'s ZSK.
- ○ Gandalf's machine will receive a final A record for `gondor.berkeley.edu` that is encrypted with a public key that Gandalf provides in his DNS query.
- ● The final A record for `gondor.berkeley.edu` will have object security.
- ○ If zones correctly implement DNSSEC, then Gandalf is secure against a MITM attacker who attempts to modify content retrieved from the `gondor.berkeley.edu` web site.

Solution: Left column (top-to-bottom):

False. KSKs are only used to sign and verify values. They are never used to encrypt values.

False. `berkeley.edu`'s ZSK is signed by `berkeley.edu`'s KSK, not the root's KSK.

True. A given name server's ZSK is always signed by that same name server's own KSK.

True. The final A record for `gondor.berkeley.edu` comes from the `berkeley.edu` name server, and `berkeley.edu` uses its ZSK to sign anything that's not a ZSK (including A records).

Right column (top-to-bottom):

False. Gandalf (the client) never provides a public key. In DNSSEC, the public keys are provided by the name servers.

True. We can be sure the A record itself (the object) has not been tampered with. See the previous subpart for a definition of object security.

False. DNSSEC guarantees that Gandalf has gotten the correct IP address for `gondor.berkeley.edu`. However, it does not secure communications between Gandalf and `gondor.berkeley.edu`. For example, if Gandalf talks to

`gondor.berkeley.edu` with HTTP (no HTTPS), the MITM could modify content.

(g) (6 points) A border firewall's primary purpose is (**Mark ONE**):

- ○ Block incoming VPN connections.
- ○ Prevent CSRF attacks.
- ○ Detect buffer overflows.
- ○ Prevent XSS attacks.
- ○ Prevent a network intruder inside the network from spreading internally.
- ● None of these.

Solution: A border firewall's primary purpose is to control which systems and which types of connections are allowed across the site's border. It does not focus on attack detection; generally allows, rather than hinders, incoming VPN connections; and cannot contain a network intruder already inside the network from spreading internally, since that spread does not transit the site's border.

(h) (8 points) Which of the following attacks might allow an attacker to steal one of your browser cookies (**Mark ALL that apply**):

- ● Reflected XSS
- ● Stored XSS
- ○ Clickjacking
- ○ None of these
- ● Buffer overflow
- ● TLS downgrade
- ○ DDoS

Solution: Both types of XSS allow the attacker to run JavaScript in your browser. JavaScript can access your cookies (assuming the HTTPonly attribute is not set) and steal them.

A buffer overflow would allow the attacker to execute arbitrary code on your machine. Depending on where the overflow is located, it may be able to access your cookies. For example, a buffer overflow in the implementation of your browser would probably be able to steal your cookies.

A TLS downgrade attack would cause you to use an outdated, possibly insecure version of TLS in your connection. This would let the attacker decrypt your TLS communications, which contains cookies.

A common mistake was to select Clickjacking. While clickjacking can cause user input (clicks or keystrokes) to go where the user did not intend it to go, these misdirections still result in HTTP requests that follow the usual rules regarding whether and which cookies to send.

DDoS attacks availability (e.g. makes some resource unavailable), but this doesn't help the attacker steal cookies.

(i) (6 points) Alice and Bob want to communicate over an insecure channel using one of the following schemes, where M is the message in plaintext. Which scheme should they use in order to avoid padding oracle attacks? Assume that (1) all of the algorithms are secure, and (2) MAC and Sign do not leak anything about M. **Mark ALL that apply.**

- ○ $\text{Enc}(M), \text{MAC}(M)$
- ● $\text{Enc}(M), \text{MAC}(\text{Enc}(M))$
- ○ $\text{Enc}(M \mid\mid \text{MAC}(M))$
- ○ $\text{Enc}(M), \text{Sign}(M)$
- ○ None of these

Solution: Recall from Project 2 that in padding oracle attacks, the attacker modifies the ciphertext in some cleverly chosen fashion, asks the client to decrypt it, and then observes whether the decryption process caused an invalid-padding error. If the attacker can observe whether such an error occurred, then this leaks partial information; after repeating this many times, an attacker can piece together all of these clues to deduce what the original message must have been.

To defend against padding oracle attacks, the recipient must be able to verify the integrity of the ciphertext *before* decrypting it. That is, the MAC / signature must be computed over the ciphertext, and not the plaintext. Hence, only the option on the top right is correct; in the others, the integrity of the message can only be verified *after* decrypting the ciphertext.

(j) (6 points) Let S be a publicly available trusted service that knows the public keys of all users. Alice communicates with S to obtain Bob's public key using the following protocol:

1. $A \longrightarrow S \;:\; A, B$
2. $S \longrightarrow A \;:\; [K_B, B]_{K_S^{-1}}$

In step 1, Alice sends along her identity A and asks S for Bob's public key. In step 2, S responds by returning Bob's public key K_B along with his identity B, and signs the message.

Which of the following attacks is this protocol vulnerable to? **Mark ALL that**

apply.

- ○ Mallory can tamper with S's response so as to substitute her own public key K_M instead of K_B.
- ○ Since S's response is not encrypted, Mallory can use K_B to decrypt any messages Alice sends to Bob in the future.
- ● Mallory can tamper with S's response so as to substitute an older key K'_B that Bob might have revoked.
- ○ None of these.

Solution: Option 1 (upper left) doesn't work because it requires Mallory to forge a signature on (K_M, B).

Option 2 is a valid attack (assuming Mallory has recorded some the older value K'_B and its corresponding signature).

Option 3 (upper right) is factually incorrect – the public key K_B is used for encrypting messages, not decrypting messages. Mallory requires the private key K_B^{-1} to decrypt the messages.

(k) (8 points) For the same situation as in the previous question, which of the following modifications to step 2 would defend against the attacks that the protocol in that question is vulnerable to? **Mark ALL that apply.**

- ○ $S \longrightarrow A : [K_B, A, B]_{K_S^{-1}}$
- ● $S \longrightarrow A : [K_B, B, T]_{K_S^{-1}}$ where T is a timestamp
- ○ $S \longrightarrow A : [K_B, T]_{K_S^{-1}}$ where T is a timestamp
- ○ $S \longrightarrow A : [K_B, B, A, N]_{K_S^{-1}}$, where N is a nonce randomly selected by S
- ○ $S \longrightarrow A : [K_B, B, N]_{K_S^{-1}}$, where N is a nonce randomly selected by S
- ○ None of these

Solution: The server can prevent a replay attack by including a timestamp in its response, as in Option 2. Option 1 (upper left) does not prevent replays. Option 3 (lower left) doesn't work because it doesn't bind the public key to Bob's identity; Mallory can request her own key from the server and then forward the message to Alice. Using a nonce instead of a timestamp in S's response (Options 4 and 5) only works if A sends the nonce to S in step 1.

A student pointed out on Piazza that the problem asks for defending against attacks applicable to the protocol in the previous question, and Option 3 does

so, even though it's flawed in a different way. We allowed full credit for answers that included Option 3 providing no other options (other than Option 1+2) were selected, and only the correct option was selected for the previous question.

Problem 3 ***Bypassing ASLR*** **(48 points)**

Mallory is trying to perform a return-to-libc attack on a simple stack buffer overflow vulnerability. She wants to overwrite the return address of the vulnerable function with the address of the `system` function, and pass it an arbitrary command argument. But the system she wants to attack has ASLR enabled, so `&system` (the address of `system`) is different every time.

Wanting to explore this further, Mallory writes the simple program:

```
#include <stdio.h>
void main() {
    printf("system is at 0x%x\n", &system);
}
```

She runs this five times, with ASLR enabled, and gets the following output:

```
system is at 0xbf9d7f14
system is at 0xbf9d7f99
system is at 0xbf9d7f88
system is at 0xbf9d7f36
system is at 0xbf9d7f08
```

(a) (16 points) She shouts "Eureka! It won't work every time, but I can easily break this now!". What did Mallory learn? How can she use it to successfully exploit the buffer overflow with a return-to-libc attack?

Solution: The implementation of ASLR has insufficient entropy — `&system` appears to have no more than one byte of randomness, so Mallory learned that just by repeatedly trying a manageable number of times, she can eventually luck upon the correct address of `system`.

Full credit required conveying the notion of exploiting repeated attempts to eventually make a correct guess.

(b) (8 points) What is the probability that Mallory will succeed if she has 1 chance to perform her return-to-libc attack?

Solution: If there is one byte of randomness, than the probability for a single attack to succeed is $\frac{1}{256}$. There might even be less randomness; in the output Mallory saw, the high bit in the bottom byte of the address never varied, so the probability might be as much as $\frac{1}{128}$.

If the probability is $\frac{1}{256}$, then it will take Mallory an expected 128 tries to guess correctly if her failed guesses do not cause re-randomization due to crash-and-restart; or an expected 256 tries if it does.

(c) (24 points) Suppose Mallory is able to control the input (i.e. `argv[1]`) to the following silly backup program, written by programmers from Junior University (assume headers necessary for this code to compile have been included):

```
// Protect our data by making 2 copies!
void double_copy(char *data) {
    char buf1[16];
    char buf2[16];

    strcpy(buf2, data);
    strcpy(buf1, data);
}

int main(int argc, char *argv[]) {
    // recall: argv[0] is the name of the program
    if (argc != 2)
        return -1;

    double_copy(argv[1]);
}
```

Give an input that will cause "`sudo rm -rf /`" to be run on the victim machine with probability equal to what you answered in the previous part.

Use the following assumptions about the victim system:

1. It is an IA-32 platform with 4-byte words (recall it's also little endian).
2. The stack is aligned at word granularity.
3. Local variables of each function are placed on the stack in the order they appear in the source code.
4. ASLR is enabled for the stack segment.
5. `argv[1] == 0x07070707` will always evaluate to true.

Hint: `#` is the shell comment character.

You can use `\x**` (where the `*`s are replaced by hex digits) to represent a character in hexadecimal form. **Fill in** the answer below:

Solution: Note 1: the write to `buf2` doesn't matter for the purposes of this problem. Any overflow will be immediately rewritten by the write to `buf1`.

Note 2: we don't use any 0 characters in our solution since these would be interpreted as NUL-terminators and cause the `strcpy()` to stop.

```
"sudo rm -rf / ##AAAA\x14\x7f\x9d\xbfDUMM\x07\x07\x07\x07"
```

Fill the 16 byte buffer. Note the shell comment character `#` at the end so that subsequent bytes are ignored	`sudo rm -rf / ##`
Overwrite `SFP` in order to get to RIP (next line).	`AAAA`
Guess at `&system`. (reversed to account for little endianness)	`\x14\x7f\x9d\xbf`
Dummy value in the stack position where a proper `IA-32 call` would have pushed `$eip` to become the stored `$rip`. The `system` function's assembly code will expect 4 bytes to proceed the arguments on the stack (by "proceed", we mean appear in lower memory addresses). This is the address that `system()` will return to. For this attack, it doesn't matter that the value is bogus since `system()` waits for the shell to finish executing, at which point Mallory has presumably completed her attack.	`DUMM`
Address of memory storing the attack string argument (`argv[1]`). This is what `system()` will read as its argument.	`\x07\x07\x07\x07`

Getting this answer fully correct was quite difficult. We awarded partial credit for specifying various parts.

Problem 4 ***Attacks on TLS*** **(72 points)**

Recall the TLS protocol, depicted in the figure below. We use the following notation: $\{M\}_K$ denotes a message M encrypted using the key K. $[M]_K^{-1}$ denotes a message M along with a signature over M using the key K^{-1}.

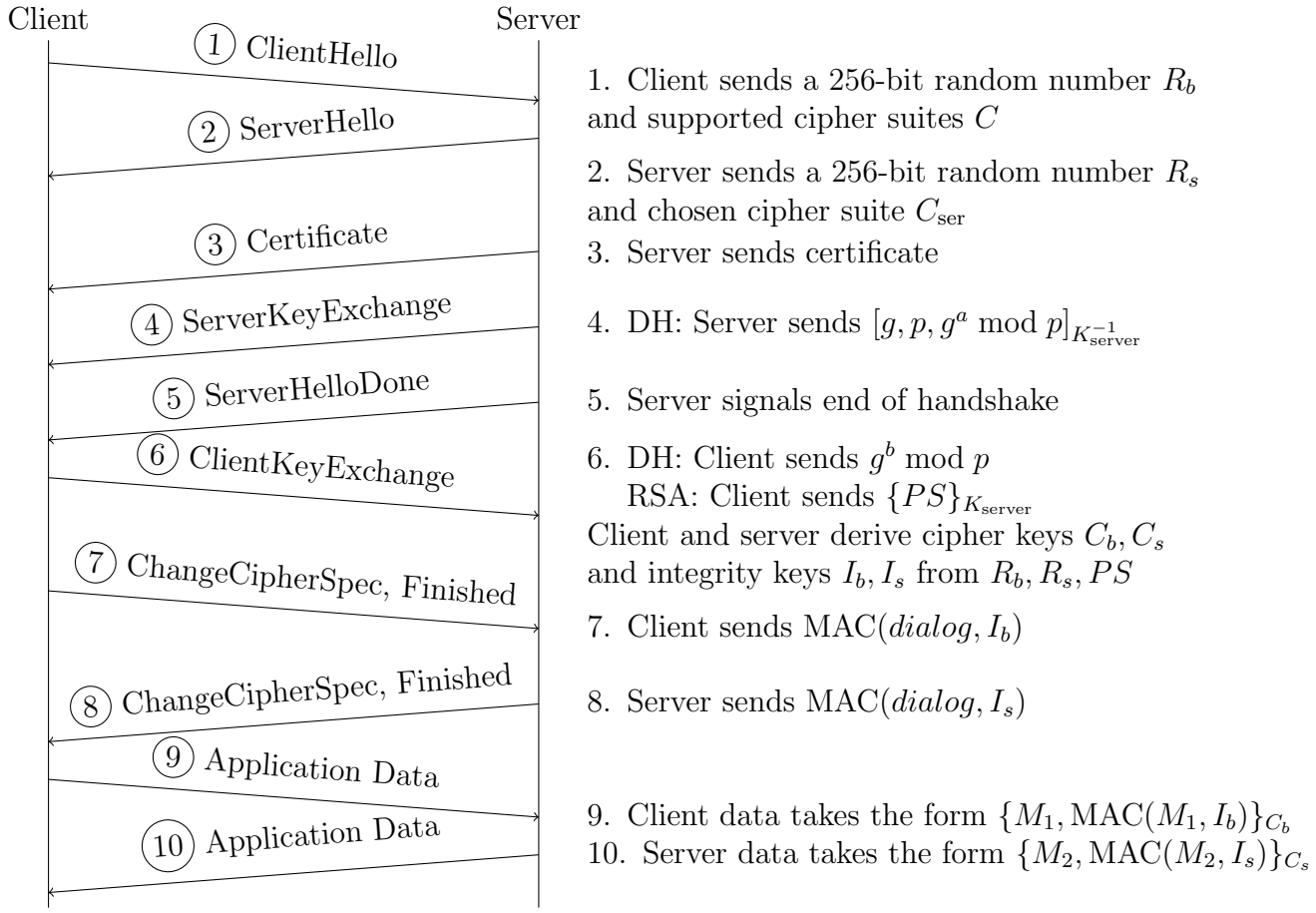

Figure 1: TLS 1.2 Key Exchange

(a) (24 points) Suppose the client and server use RSA to exchange the premaster secret. Mallory intercepts the ClientKeyExchange message and replaces PS with a fake value PS'. Assume that Mallory can modify the messages *after* ClientKeyExchange as well, if required. Which of the following are true? **Mark ALL that apply.**

- ○ Mallory will be able to decrypt the application data sent by the client to the server.
- ○ Mallory will be able to decrypt the application data sent by the server to the client.
- ○ The server will detect the tampering when it receives ClientKeyExchange.
- ○ Mallory can avoid detection until the server receives Finished from the client, at which point she'll be detected.
- ● Mallory can avoid detection until the client receives Finished from the server, at which point she'll be detected.
- ○ None of these

Solution: Mallory can pick an arbitrary PS', encrypt it with the server's public key to obtain a valid ciphertext $\{PS'\}_{K_{\text{server}}}$, and replace $\{PS\}_{K_{\text{server}}}$ with $\{PS'\}_{K_{\text{server}}}$ in ClientKeyExchange. The server will be unable to detect the tampering, and Option 3 (lower left) is therefore incorrect.

Mallory can then obtain the symmetric keys C'_b, C'_s, I'_b, I'_s as derived by the server from PS'. She can use I'_b to forge a valid MAC over the modified dialog in Step 7 and avoid detection by the server. Option 4 (upper right) is therefore incorrect.

However, Mallory will still be unable to obtain the keys C_b, C_s, I_b, I_s as derived by the client, since she does not know the value of PS. In Step 8, the server will send the client a MAC over the dialog using the fake key I'_s. In order to avoid detection, Mallory would need to re-compute this MAC using the key I_s as expected by the client. She will be unable to do so (since she does not know I_s), and will necessarily be detected (Option 5).

Once the client detects the tampering, it will terminate the handshake. Options 1 and 2 (top and middle left) are therefore incorrect as well. Of the two of these, Option 2 is better than Option 1. Mallory is trying to get the server to accept a bogus PS, which would allow Mallory to know what keys the server uses, and therefore to potentially decrypt the data it sends. But Mallory has no way to change the *client*'s view of PS, nor to recover the original PS, so even if Mallory's subterfuge goes undetected, she has no way to read data that the client sends. Therefore, answers that included Option 1 received no credit. Answers that included Option 2 but not Option 1 potentially received some partial credit, depending on which other options were selected.

(b) Now suppose the client and server use Diffie-Hellman for exchanging the premaster secret. Mallory wants to decrypt the data sent by the server to the client by *downgrading* the cipher suites. She doesn't care about the data sent by the client to the server. If the server always picks the strongest cipher suite and parameters available, specify whether Mallory's attack will succeed in the following scenarios **(Yes/No)**.
If yes, then list the handshake messages Mallory will need to *necessarily* modify. If not, explain why.
Assume that unless specified, all cryptographic algorithms supported by the client and server are secure.

i. (12 points) Suppose the client and server support 3DES in addition to AES. Mallory is aware of an attack on 3DES that allows her to learn any message encrypted using it. She therefore wishes to force the client and server to use 3DES instead of AES as the encryption algorithm.

Solution: No. Mallory will still be unable to learn the master secret, and hence will not be able to forge the MAC over the dialog. She will thus get detected.

ii. (12 points) Suppose the client and server support a weak variant of Diffie-Hellman (DH_{weak}). Mallory is aware of an attack on DH_{weak} that allows her to learn the exchanged secret. She therefore wishes to force the client and server to use DH_{weak} instead of standard Diffie-Hellman.

Solution: Yes. Mallory needs to modify ClientHello + both Finished messages. Since she can extract the exchanged secret, she can derive the integrity keys necessary to modify the MACs.

(c) (24 points) Recall that ClientHello contains a nonce R_b, along with C, the cipher suites supported by the client. ServerHello contains a nonce R_s along with C_{ser}, the cipher suite chosen by the server. Which of the following modifications to the TLS protocol would prevent Mallory from conducting *any* downgrade attacks on the cipher suites? **Mark ALL that apply.**

- ○ ServerKeyExchange includes $[R_b]_{K_{\text{server}}^{-1}}$, $[C]_{K_{\text{server}}^{-1}}$
- ○ ServerKeyExchange includes $[C]_{K_{\text{server}}^{-1}}$, $[C_{\text{ser}}]_{K_{\text{server}}^{-1}}$
- ○ ServerKeyExchange includes $[R_b, C]_{K_{\text{server}}^{-1}}$
- ● ServerKeyExchange includes $[C \,||\, C_{\text{ser}}]_{K_{\text{server}}^{-1}}$
- ○ ServerKeyExchange includes $[C]_{K_{\text{server}}^{-1}}$
- ● ServerKeyExchange includes $[R_b \,||\, C \,||\, C_{\text{ser}}]_{K_{\text{server}}^{-1}}$
- ○ ServerKeyExchange includes $[C_{\text{ser}}]_{K_{\text{server}}^{-1}}$
- ○ None of these

Solution: The attack won't work if the client can verify that the cipher suites C received by the server were altered by Mallory. To this end, the server can send the client a signature over C after binding it with C_{ser} (as in Options 6 and 7). The client can then verify the signature, validate C, and be assured that the server chose C_{ser} after receiving the correct cipher suites. (Including R_b as in Option 7 is unnecessary, but doesn't cause any problems.)

Options 1, 2 and 3 (lefthand column) don't work because Mallory can obtain the necessary signature by launching her own separate TLS session with the server and sending C in the ClientHello message. Including R_b in the signature doesn't help because Mallory can still MITM the connection as follows:

1. Pause the client's ClientHello message
2. Launch a separate TLS session using the client's R_b and C, obtaining a valid signature over these

3. Resume the client's ClientHello after downgrading C to C'

4. Replace the server's signature over C' and R_b using the one obtained in (2).

Option 4 (lower left) doesn't contain a signature over C, so the client can't tell that the server never received the client's original cipher suite offerings.

Option 5 (upper right) doesn't work because it contains *separate* signatures over C and C_{ser}, allowing Mallory to selectively replace the signature over C, as described above.

Problem 5 ***Software vulnerability*** **(64 points)**

Here is a fragment of Python source code for a fictitious email-based spell-checker service:

```
def process_incoming_email(msg):
    return_addr = msg.get("From")
    search_term = msg.get("Subject")
    status = os.system("fgrep " + search_term + " /usr/share/dict/words")
    if status == 0: # exit code 0 means success
        response = "The word " + search_term + " is spelled right!"
    else:
        response = "Sorry, " + search_term + " is not a word."
    send_response(return_addr, response)
```

In this service, users submit a word to check as the `Subject:` header field of an email. For example:

```
To: spellcheck@example.com
From: user@berkeley.edu
Subject: phenommenon
```

The `process_incoming_email` function is responsible for checking the spelling of the word, generating an appropriate response message, and sending the response back to the original sender via email. The function works by extracting a search term from the `Subject` header field, then using the `fgrep` command to search for the term in `/usr/share/dict/words`, a file containing a list of English words. The `fgrep` command searches a file for a fixed text pattern; its syntax is '`fgrep` ***`pattern filename`***'.

`os.system` is a Python function that accepts a single string and executes the string using the command shell. It works the same as the C `system()` library routine that we discussed in lecture.

(a) (16 points) The `process_incoming_email` function has a vulnerability. What email `Subject:` could you send that would cause the server to pause for 10 seconds before replying?

`Subject:`

Solution: There are several possibilities. Here are some examples (the last two only work if the shell used is Bash):

```
Subject: ; sleep 10; fgrep word
Subject: word /dev/null; sleep 10; echo
Subject: & sleep 10 #
Subject: $(sleep 10)
```

We accepted various ways of running a delay, like `sleep 10`, `wait 10`, and `sleep(10)` (even though the parentheses are shell metacharacters and wouldn't actually work in a shell command). We also accepted `//` and `--` as comment markers in place of `#`.

We show what command each of these `Subject:` lines will run. The general theme is to first do something that escapes the context of the `fgrep` command, and at the end do something to take care of the `/usr/share/dict/words` that will be appended.

`fgrep & sleep 10 # /usr/share/dict/words`
: `fgrep` is immediately put into the background with '`&`' and ignored. The `sleep` command runs, then '`#`' starts a comment that continues to the end of the line.

`fgrep ; sleep 10; fgrep word /usr/share/dict/words`
: The '`;`' ends the `fgrep` command. Running `fgrep` without arguments is an error, but the error is ignored and the shell next runs the `sleep` command. The injected `fgrep word` combines with the code's `/usr/share/dict/words` to finish with a valid command, as if the user had done a request for 'word'.

`fgrep word /dev/null; sleep 10; echo /usr/share/dict/words`
: The first `fgrep` command searches `/dev/null`, a pseudo-file with no contents (basically turning the `fgrep` into a no-op. The `echo` command at the end harmlessly consumes the terminating filename.

`fgrep $(sleep 10) /usr/share/dict/words`
: The `$(...)` syntax spawns a new subshell and substitutes the result of the subcommand into the top-level command line. You can also write it using backquotes: `` `sleep 10` ``. Because `sleep` does not produce any output, the command will first sleep for 10 seconds, then run the command `fgrep /usr/share/dict/words`.

(b) (16 points) What email `Subject:` could you send that would tell you whether or not there is a user called `dbadmin` on the spellcheck server? The list of users is stored in the file `/etc/passwd`.

`Subject:`

Solution:

`Subject: dbadmin /etc/passwd #`

The above search term will cause `process_incoming_email` to build this command:

`fgrep dbadmin /etc/passwd # /usr/share/dict/words`

Instead of searching `/usr/share/dict/words`, `fgrep` will search `/etc/passwd`. The '`#`' turns the `/usr/share/dict/words` at the end of the command into a comment. The exit code of `fgrep` will reflect whether the search term `dbadmin` was found in `/etc/passwd`. If `dbadmin` was found, the email reply will be

`The word dbadmin /etc/passwd # is spelled right!`

If `dbadmin` was not found, the email reply will be

`Sorry, dbadmin /etc/passwd # is not a word.`

Some ideas that do not work completely:

`dbadmin /etc/passwd; cd`
: This doesn't work because the return value of `os.system` is the return value of the last command. Putting a `cd` (or other command) at the end will cause the return value of `os.system` to reflect that command, not `fgrep dbadmin /etc/passwd`.

`dbadmin /etc/passwd;`
: This doesn't work because the `/usr/share/dict/words` that gets appended will be interpreted as a command name by the shell. Because the file is not executable, trying to execute it will always result in a nonzero exit status.

  The slight variant `dbadmin /etc/passwd` (no semicolon at the end) actually *does* work, because the `/usr/share/dict/words` that gets appended will be interpreted as an additional filename argument to `fgrep`, not a separate command. The search will work correctly as long as `dbadmin` is not present in `/usr/share/dict/words`.

(c) (32 points) State **one** way that you could fix the vulnerability? (If you name more than one, we will only grade the first.)

Solution: Possible solutions:

- Use `subprocess.Popen` in place of `os.system`. (Or some other API that allows you to specify a vector of command arguments, rather than a shell string.)

- Carefully escape `search_term` to remove shell metacharacters. (This is tricky because it requires detailed knowledge of how the shell will interpret each character.)
- Implement the search in Python rather than using a `fgrep` subprocess.

Problem 6 ***Coffee Shop Worries*** **(54 points)**

Alice and Bob just arrived at Brewed Awakening, the local coffee shop. Eve is already there, enjoying a cup of tea.

(a) (6 points) Alice wants to connect to Brewed Awakening's WiFi network. Under which protocols would her connections be safe from sniffing attacks by other coffee shop visitors, such as Eve? **Mark all that apply.**

- ○ WEP
- ● WPA2 - Enterprise mode
- ○ WPA2 - Personal mode
- ○ None of these

Solution: Only "WPA2 - Enterprise mode" provides per-connection secret keys with the WiFi access point to secure each connection separately.

(b) (24 points) Turns out that Brewed Awakening's network has no encryption. Alice warns Bob that its not safe to use this connection, but Bob disagrees. Bob connects to the WiFi, and tests that he has Internet connectivity by going to `https://kewlsocialnet.com`. It loads without issues. Bob says the Alice: "See, no problem! That access was totally safe!"

If Bob is correct and the access to `kewlsocialnet.com` was safe, explain why he is correct. If he is not correct, provide a network attack against Bob.

Answer:

Solution: Bob is correct.

Bob is visiting an HTTPS website, which uses TLS to provide an end-to-end secure channel. As Bob's browser did not encounter any certificate warnings, then unless there's been a CA breach or some other CA issue, the network connection has confidentiality, authentication, and integrity.

We allowed full credit for solutions that specified that Bob was incorrect and provided a valid approach for undermining his HTTPS connection to the site, including the threat of obtaining fraudulent certs from misbehaving CAs.

We allowed only partial credit for solutions that framed attacks that would work in the situation if TLS did not provide all of the strong security properties that it

does. These solutions received more credit if they clearly stated that the attack is relevant for Bob's *subsequent* connections, rather than his test connection. These solutions received less credit if they were simply stating that because the WiFi network is unencrypted, an attacker could read Bob's private information, since use of HTTPS prevents that.

(c) (24 points) Now that he has tested his WiFi access, Bob then tells Alice: "I want to buy that last muffin at the counter. Let me check if I have enough money in my bank account." Eve hears this and panics! She wants the last muffin too but is waiting for her friend Mallory to bring enough cash to buy it. She is now determined to somehow stop Bob from buying that last muffin by preventing him from checking his bank account. Through the corner of her eye, Eve sees Bob start to type `https://bank.com` in his browser URL bar ...

Describe two network attacks Eve can do to prevent Bob from checking his bank account. For each attack, describe clearly in one or two sentences how Eve performs the attack.

Attack #1:

Attack #2:

Solution:

Note that Eve cannot do an ARP or DHCP spoofing attack as Bob has already connected to the WiFi network, so already knows the IP and hardware addresses of the local network's gateway and DNS resolver. (This assumes that extraneous ARPs are not accepted by Bob's system. ARP spoofing is a viable answer for this problem if accompanied by specific mention of this consideration.)

1. TCP RST injection attack — Eve can sniff Bob's transmitted (and received) packets, so she can observe the sequence numbers of TCP packets. Thus, Eve can send a valid TCP RST packet to Bob's browser (or to the bank website), resetting the TCP connection.
2. DNS response spoofing — When Bob tries to load the bank website, his browser will generate a DNS request for the bank's domain. Eve can spoof a response with an incorrect answer, preventing Bob from loading the bank website properly.
3. DoS attack on either Bob's system or the coffee shop network. This can be done through various means, such as DNS amplification attacks directed at Bob.

DoS attacks on the bank itself only received partial credit, as it assumes that `bank.com` is a site that is as easy to overwhelm as Bob's system or the local network.

Kaminsky's DNS cache poisoning attack is inappropriate because it assumes the Bob's system remains vulnerable to a vulnerability for which we discussed how it has been subsequently fixed; and it is not necessary because Eve is on-path, not off-path.

Problem 7 ***The Great Cannon*** **(96 points)**

In 2015, Github experienced a DoS attack orchestrated by China using the so-called "Great Cannon" (GC). It worked as follows. (Some details of the attack have been simplified or modified for this problem.)

Many websites include a fetch for a script for analytics from Baidu, a large Internet service in China somewhat similar to Google. The script would be retrieved via `http://hm.baidu.com/h.js`. The GC operated in-path at the border between China and the rest of the Internet. Upon seeing a request for this script, the GC would prevent the original HTTP request from being forwarded, and would instead return a different script, which instructed clients to repeatedly load `http://github.com/cn-nytimes`.

You can assume that Baidu served its traffic using servers in China; Github did so from servers in the USA; and websites using the analytics script were hosted all over the world.

(a) (6 points) For which of the following layers would the GC need to **guess** or **infer** header values it could not directly determine in order to carry out the attack? **Mark ALL that apply.**

- ○ Physical
- ○ Transport
- ○ Link
- ○ Application
- ○ Network
- ● None of these

Solution: Because the GC operated in-line, it can see all of the packet header values it needed in order to construct its bogus reply. Because the fetch used HTTP and not HTTPS, it could in addition see the germane application-layer values.

(b) (6 points) Which layer was this attack meant to particularly stress regarding Github's servers? **Mark the BEST choice.**

- ○ Physical
- ○ Network
- ○ Link
- ○ Transport

● Application ○ None of these

Solution: The queries redirected to Github all used fully established TCP connections. This means they did not particularly stress the Transport layer or any lower layers. The attack imposed load on Github's web server process, i.e., at the Application layer.

(c) (4 points) Whose traffic contributed to the DDOS attack? **Mark the BEST choice.**

○ Web browsers inside China ○ Both of these

● Web browsers outside China ○ Neither of these

Solution: The DDoS attack is happening as a result of the malicious script provided by the GC.

The question states that websites all over the world are requesting the analytics script, which requires a request to Baidu in China. For web browsers outside China, this request passes through the GC, and the GC returns the malicious script to the web browsers outside China.

For web browsers inside China, the request never passes through the GC, so the GC won't be able to replace the request with the malicious script.

(d) (4 points) Which packets would the implementers of this attack need to inspect? **Mark the BEST choice.**

● Packets going into China ○ Both of these

○ Packets going out of China ○ Neither of these

Solution: As described in the solution to the previous part and the question text, the GC is inspecting requests from the rest of the world to Baidu in China. These are packets going into China.

(e) (12 points) Why doesn't the Same Origin Policy prevent this attack? (Limit your answer to no more than 2 sentences.)

Solution: The SOP prevents one origin from accessing another's DOM. (The DOM is the browser's internal representation of an HTML page.)

Here, the issue is instead that an origin can still *load* another site (e.g., in an iframe), which in this case will trigger a request to Github and increase load on the Github server.

(f) (12 points) For this and the next question, suppose that after the attack began, Github installed a NIPS to deal with this particular attack. Assume the NIPS is deployed on the Ethernet link connecting the `github.com` server to the public Internet. What kind of detection is MOST LIKELY to be effective under the circumstances? **Mark the BEST choice** and provide a **short explanation**.

- ● Signature-based
- ○ Anomaly-based
- ○ Specification-based
- ○ Behavioral
- ○ Honeypots
- ○ Vulnerability scanning

Explanation:

Solution: As described in the problem, instances of the attack traffic will all look similar: particular `github.com` page requests with the same `Referer`. That situation fits with the general notion of signature-based detection looking for activity that reflects a known attack.

Anomaly-based could potentially work if accompanied by a discussion of the Github operators having trained their detector on past access patterns (such as the order in which a given client visits different `github.com` pages).

Specification-based approaches would be difficult to manage. It's not clear what sort of specification Github could use that would allow legitimate traffic but for which this traffic would be in violation.

Behavioral detection generally applies to detecting the *consequences* of an attack rather than the attack itself. It thus is not a direct fit here. That said, an approach based on a behavior such as "visited `http://github.com/cn-nytimes` as the first-visit-ever to `github.com`" could work, accompanied by discussion that the visit is evidence of a successful attack on the client (namely, the GC manipulating the client's operation).

(g) (12 points) Suppose that the attack caused Github to receive 50 times as many bogus requests as legitimate requests, and that Github will consider a defense successful if it reduces the volume of flooding requests by at least a factor of 50, so the flooding is no larger than the volume of legitimate requests. Suppose further that Github found that their NIPS had a precision of 0.999 and a recall of 0.99 when detecting this attack. To what degree would this represent a successful defense?

Mark ONE of the following and **BRIEFLY explain** ($\leq$ 2 sentences) your answer.

- ● Yes, the NIPS provided a successful defense.
- ○ No, the NIPS did not provide a successful defense.
- ○ Additional information is needed to tell whether the NIPS provided a successful defense.
- ○ Such a combination of precision and recall values is not possible under these circumstances.

Explanation:

Solution: A reminder that Precision = P[attack|alert] and Recall = P[alert|attack]. Github's success criterion is a reduction of the flooding traffic by a factor of at least 50. This means that at most 2% of the flooding requests can escape detection. A recall value of 0.99 means that 99% of the flooding requests result in an alert (and thus in blockage, since Github is using an network IPS), so only 1% of the flooding requests will escape detection.

While not part of what we expected in a solution, there's a somewhat subtle additional consideration. The problem's wording states that "flooding is no larger than the volume of legitimate requests." If the blocking included dropping many legitimate requests, then it could be difficult to tell whether this requirement is met. However, the precision of 0.999 means that this is not a significant effect. Consider the fate of 10,000 requests, of which about 9,800 are bogus and about 200 are legitimate (so about a 50:1 ratio). We expect about 9,702 alerts for the attack traffic (we miss 1% of them, per the recall value). A precision of 0.999 means that we could have in addition at most about (in expectation) 9.7 additional alerts that correspond to legitimate activity. This constitutes only 5% of the legitimate requests.

While not part of the problem framing, one could wonder whether the defense might fail to be "successful" due to excessive false positives. In this situation, the concerns of the Base Rate Fallacy do *not* apply: what's being detected is something that is very common (more so than legitimate activity), rather than something that is much more rare than legitimate activity. That said, while the value for precision tells us that nearly all of the alerts correspond to actual attacks, it does not tell us whether the occasional blocking of legitimate traffic due to the (rare) incorrect alerts is inexpensive enough (in terms of impact to the site's operation) so as to not constitute an operational problem.

Answers that stated that more information was necessary received partial credit if framed in terms of false positives. (This credit was reduced if the answer also discussed the need to know about false negatives, since the value of Recall already provides this.) They did not receive full credit, however, because the

problem explicitly framed what Github considered to be successful, namely a 50x reduction in attack traffic.

(h) (8 points) This attack occurred for sets of HTTP requests. Which of the following changes would have prevented the attack? Consider each choice in isolation (i.e., assess whether it prevents the attack assuming none of the other choices are in effect). **Mark ALL that apply.** For each choice, assume that the content that the site serves remains the same.

- ○ Every website that uses Baidu's analytics switches to serve its content using HTTPS instead of HTTP.
- ○ Baidu's analytics server redirects any incoming HTTP connection to a corresponding HTTPS URL.
- ○ None of these.
- ● Baidu switches its analytics server over to only be accessible using an HTTPS URL.
- ○ Github's server redirects any incoming HTTP connection to a corresponding HTTPS URL.
- ○ Github switches its server over to only be accessible using HTTPS.

Solution: If Baidu switches its analytics server over to only be accessible using an HTTPS URL, then the requests that the GC sees will (1) be encrypted, and (2) have their integrity protected due to use of TLS. This will prevent the GC from manipulating them.

If websites using Baidu's analytics switch to HTTPS, that simply means that the server's response that includes the instruction to fetch the analytics script enjoys TLS's protection. However, the browser receiving the response will still issue an HTTP request to get the analytics script, which the GC can manipulate.

If Baidu's analytics server redirects incoming HTTP requests to an HTTPS URL, that action occurs too late in the process. The GC will see the incoming HTTP request first, and can manipulate it at that point, before there's an opportunity for the request to switch over to HTTPS.

Github changing to HTTPS (either option) doesn't prevent the attack: what the GC is manipulating is the reply to the request for the analytics script, not the request sent to Github.

(i) (8 points) Which of the following techniques could Github have used to make the attack ineffective? **Mark ALL that apply.**

- ○ Blacklist any packets from Chinese IP addresses
- ○ Use SYN cookies for all new connections

- ● None of these
- ○ Move the affected Github server to a new IP address
- ○ Remove all use of Baidu analytics from Github web pages

Solution: None of these solutions will be effective:

1. The attack traffic seen by Github does *not* come from Chinese IP addresses. The GC is manipulating browsers *outside* of China to send traffic to Github.
2. The attack does not involve spoofed TCP SYN requests, so SYN cookies will not provide any benefit.
3. If Github moves their server to a new IP address, they will need to update the DNS binding of github.com to point to resolve to the new address; because the GC directs browsers to retrieve a URL that uses that domain name, rather than a direct IP address, the attack traffic will follow the move.
4. The Baidu analytics scripts are used on numerous *other* websites; the attack will continue as before regardless of whether Github itself uses Baidu analytics.

(j) The remainder of this problem concerns a Web security feature called Subresource Integrity (SRI). It works by adding an attribute to the `script` tag for externally loaded scripts:

`<script src="http://example.com/script.js" integrity="[`*CRYPTOGOOP*`]">`

Browsers then validate the integrity of the script retrieved from the given `src=` location.

i. (8 points) What should *CRYPTOGOOP* contain for it to achieve its goal of assuring integrity, while minimizing the effort required by web developers to adopt it? **Mark the BEST answer.**

- ○ An encryption of the script being loaded
- ○ A digital signature of the script being loaded
- ○ A MAC of the script being loaded
- ○ A hash of the URL of the script
- ● A hash of the script being loaded

Solution: If a browser visits website A and receives from it a script directive to fetch a script from website B, then all that is needed to ensure

that the *fetch* has integrity is a hash, because an attacker who messes with the fetch would have to also mess with the original reply from A in order to alter the CRYPTOGOOP. If the attacker can do that, they can change A's reply in arbitrary ways anyway, so there's no need for the attacker to instead only adjust A's reply to enable the attacker to supply a bogus value for the fetch from B.

While in principle MACs or digital signatures could provide integrity, they both introduce unnecessary complexity. For MACs, website A would need to provide the associated key; doing so reduces the MAC to essentially a hash-based approach. For digital signatures, the browser would have to use some sort of PKI to verify that the script it receives matches the signature. This adds complexity without providing any benefit to offset that.

ii. (8 points) Suppose every website with Baidu's analytics starts using SRI. Given GC's capabilities, could it still redirect some Baidu analytics traffic to Github?

● Yes ○ No

Explanation (1 sentence):

Solution: If some of the websites were hosted within China and over HTTP, the Great Cannon could potentially replace their SRI response with one that included an SRI tag that matches the malicious script.

iii. (8 points) Name **ONE** drawback to a website's owner from deploying SRI. (If you name more than one, we will only grade the first.)

Drawback:

Solution: Website owners will have to manually update the SRI tag if the external script changes. This will require coordination between the website and the external site to pick up any new features, bug fixes, or security updates. We allowed partial credit for answers that did not identify this consideration but did flag that the website owner would have to do some initial work to securely gather the CRYPTOGOOP.

We allowed partial credit for answers stating that the use of SRI would slow down web page load times. This is a very modest effect, since hashes are quick to compute.

No credit was given for answers that discussed the cost of servers having to generate CRYPTOGOOP in response to incoming requests for a script. The generation would be done offline, since it's only needed one time per change to a script.

No credit was given for answers that discussed leveraging the cost of cryptography for DoS. That would not be a drawback for the website owner, since it would only be relevant for them if they themselves wanted to use SRI to induce a DoS on other sites.

Problem 8 ***Computing on encrypted data*** **(56 points)**

Recall the El Gamal scheme: The El Gamal public key is (p, g, h), x is the private key, and $h = g^x \bmod p$. The encryption of a message M is $\mathsf{Enc}(M) = (g^r \bmod p, M \times h^r \bmod p)$, for a random r.

(a) (24 points) Say function F can be computed over El Gamal ciphertexts. This means given only $C_1 = \mathsf{Enc}(M_1) = (s_1, t_1)$, $C_2 = \mathsf{Enc}(M_2) = (s_2, t_2)$, and the El Gamal public key, anyone can compute a ciphertext $C_3 = \mathsf{Enc}(F(M_1, M_2))$

Which of the following arithmetic operations can be computed over El Gamal ciphertexts? **Mark ALL that apply.**

- ○ Modular Addition: $F(M_1, M_2) = M_1 + M_2 \bmod p$
- ○ Modular Exponentiation: $F(M_1, M_2) = {M_1}^{M_2} \bmod p$
- ● Modular Multiplication: $F(M_1, M_2) = M_1 \times M_2 \bmod p$
- ○ None of the above

For each arithmetic operation you select, write down the equation that someone can use to compute C_3 using the components of C_1, C_2 (i.e., s_1, t_1, s_2, t_2), and the public key. Or if none of the computations is possible, explain why not.

Equation(s) or Explanation:

Solution: Modular Multiplication: $C_3 = (s_1 s_2 \bmod p, t_1 t_2 \bmod p)$

(b) (24 points) Suppose Alice sends Bob a message M_0 after encrypting it with Bob's El Gamal public key. Let $C_0 = (s_0, t_0)$ be the corresponding ciphertext. Mallory wants to learn the message M_0. Bob agrees to decrypt a single ciphertext $C_1 = (s_1, t_1)$ of Mallory's choice, as long as $C_1 \neq C_0$. Explain how Mallory can take advantage of Bob's offer in order to learn M_0.
Hint: Mallory observes that she can manipulate C_0 in a way that allows her to obtain another valid ciphertext that also decrypts to M_0.

Solution: El Gamal is vulnerable to chosen ciphertext attacks, as this question aims to illustrate. A couple of ways in which Mallory can learn M_0 are as follows:

1. Let $C_0 = (g^r \bmod p, M_0 \times h^r \bmod p)$. Mallory picks an arbitrary number s, and computes $C_1 = (g^{r+s} \bmod p, M_0 \times h^{r+s} \bmod p)$ (since g, h, and p are known). She then asks Bob to decrypt C_1. The corresponding plaintext M_1 will be equal to M_0.
2. Let $C_0 = (g^r \bmod p, M_0 \times h^r \bmod p) = (s_0, t_0)$. Mallory picks a message M_1 and computes its ciphertext $C_1 = (g^s \bmod p, M_1 \times h^s \bmod p) = (s_1, t_1)$ by encrypting M_1 with Bob's public key and using a random number s.

She then computes:

$$\begin{aligned} C_2 &= C_1 \times C_0 \\ &= (s_0 s_1 \bmod p, t_0 t_1 \bmod p) \\ &= (g^r g^s \bmod p, M_0 M_1 \times h^r h^s \bmod p) \\ &= (g^{r+s} \bmod p, M_0 M_1 \times h^{r+s} \bmod p) \end{aligned}$$

and asks Bob to decrypt C_2. The corresponding plaintext will be $M_2 = M_0 \times M_1 \bmod p$. Mallory can then obtain $M_0 = M_2 \times M_1^{-1} \bmod p$.

Students came up with innovative additional approaches. A simpler version of the second option above is for Mallory to encrypt $M_1 = 1$, which eliminates the need to multiply by $M_1^{-1} \bmod p$.

Even simpler is to construct $C_1 = C_0 + p$. Given that both components of the ciphertext are processed mod p, this ciphertext will decrypt to the same value as what C_0 decrypts to. The ciphertext is arguably not "well-formed" (which we didn't specify for this problem). However, it's certainly within the spirit of conducting attacks to look for ways to bend the rules!

We did not provide any partial credit for solutions that were based on a misconception in the previous problem regarding which functions can be computed on El Gamal ciphertexts. Such approaches are arithmetically flawed. In addition, it is possible to solve this problem even if a student did not correctly solve the previous problem.

(c) (8 points) Which of the following best describes the attack in the previous question?

- ○ Ciphertext-only attack
- ○ Known plaintext attack
- ○ Chosen plaintext attack
- ● Chosen ciphertext attack

Solution: This is a chosen ciphertext attack, because Bob (the victim) is agreeing to decrypt some ciphertext of Mallory (the attacker)'s choice. In other words, the attacker is *choosing* the *ciphertext* for the victim to decrypt.

Problem 9 ***Password Cracking*** **(56 points)**

Mallory has an account on `www.lamesec.com`, a hot new social networking site. So does her rival, Alice. Mallory desperately wants to break into Alice's account (username "alice") to read Alice's private messages. `www.lamesec.com` specifies that account passwords must:

1. Be no longer than 7 characters.
2. These characters must be either lowercase letters or one of the following symbols: `+`, `-`, `_`, `$`, `*`, `!`.
3. Should be randomly chosen given these constraints.

One of Alice's many traits that Mallory finds annoying is that Alice will always comply with rules like these.

Mallory has observed that if she tries to use her browser to guess a possible password for Alice's account, she receives a reply from the `www.lamesec.com` web server that looks like:

No Dice

Your attempt to authenticate as **alice** *failed*.

Apache/2.4.25 (Fedora) Server at www.lamesec.com Port 80

Job status: started 10:39:32 May 12 2017, completed 10:39:32 May 12 2017
Total # instructions executed 21333427
Total memory required 1522667 bytes
Total disk storage 0 bytes
Have a nice day

Mallory in addition notices that the site uses a framework for which the below Python code at the server validates authentication attempts:

```
def CheckPassword(account, submitted_password):
  if len(submitted_password) != len(account.password):
    return False
  for i in range(len(submitted_password)):
    if submitted_password[i] != account.password[i]:
      return False
  return True
```

Assume that the code is compiled without any optimization, and that all comparison operators take a single instruction to execute. Also assume that `len(x)` always takes

the same number of instructions to execute regardless of how long `x` is, and access to `account.password` likewise takes constant time.

Knowing that this is the specific code that is used, Mallory analyzes the information returned for a number of failed authentication attempts she makes to *her own* account. In doing so, she is free to repeatedly change her password to new values if she wishes. After analyzing this information, Mallory feels ready to try to attempt to infer information about Alice's password.

(a) (12 points) How many authentication attempts will suffice for Mallory to determine the *length* of Alice's password? Choose the **MINIMUM** such number of attempts that *guarantees* success for Mallory:

- ○ 1 attempt
- ● 10^3 attempts
- ○ 10^6 attempts
- ○ 10^9 attempts
- ○ 10^{12} attempts
- ○ Mallory can do this but will need more than 10^{12} attempts
- ○ Mallory cannot do this

Solution: The key observation to this problem is that the web server's answer leaks information about the amount of computation done to validate the password for the account. This provides a *side channel*. The server's reply provides 5 types of such potential information:

1. The server's software version. This won't change between answers, so does not provide any additional information.
2. The time the job began and finished, to one-second granularity. As indicated in the example, the resolution of these times (here, resulting in an elapsed time of 0 seconds) is too coarse to provide a useful timing channel.
3. The number of instructions executed. This provides a very fine-grained indication of how much computation was done, analogous to (and in fact even better than) the timing channel discussed in lecture.
4. The amount of memory required. This *could* provide fine-grained information about the amount of computation that's done, but the checking routine (and the one in the next part of the problem) doesn't vary its memory consumption based on the progression of the password-matching process. (But see below.)
5. The disk storage required. As indicated in the example, this does not appear to provide any useful information.

Note that the problem is framed in terms of Mallory using her browser to access the server: thus, Mallory is not operating on the same system as the server,

and cannot use techniques such as leveraging page-alignment to induce a timing signal.

Thus, Mallory will need to exploit the side channel of the number of instructions executed. Given that, if Mallory tries a password with a different number of characters than in Alice's password, then the first conditional will fail, resulting in a lower instruction count than if Mallory's guess has the same number of characters. Given that, by trying 7 passwords of distinct lengths, she can see which one executes more instructions, revealing the length of Alice's password. In fact, trying 6 passwords will suffice, since if the instruction counts for all of them match the not-the-right-length count, then the length not tried must be the correct length.

Mallory cannot however guess the length of Alice's password in 1 guess, because for her one guess she might pick a candidate with the wrong length, which will not then provide information about which of the 6 other possible lengths is the correct one.

When grading, however, we realized that some students assumed that the amount of memory consumed would depend on the length of Alice's password. While the code we provided does not have this property, we agree that that is a reasonable possibility. That side channel could enable determining Alice's password in a single query (by Mallory first constructing a dictionary of memory sizes seen for different-length passwords for her own account). Accordingly, we also provided full credit for solutions that stated that the length could be determined in just one attempt.

(b) (16 points) How many authentication attempts will suffice for Mallory to determine the *exact value* of Alice's password? Choose the **MINIMUM** such number of attempts that *guarantees* success for Mallory:

- ○ 1 attempt
- ● 10^3 attempts
- ○ 10^6 attempts
- ○ 10^9 attempts
- ○ 10^{12} attempts
- ○ Mallory can do this but will need more than 10^{12} attempts
- ○ Mallory cannot do this

Solution: Mallory first determines the length of Alice's password. This requires at most 6 attempts, each of a distinct length. (Because even if all 6 fail, then she knows that the length she didn't try is the correct one.)

Because the checking routine exits immediately upon finding a mismatch, Mallory can know how many instructions correspond to a mismatch in the 1st, 2nd,

3rd, etc. characters. (She can experiment on her own account to determine these values.) Thus, to determine the 1st character, she can try all 32 possible initial values and see which one progresses further into the matching than the others. Once she has determined that, she can repeat this process for the 2nd character, and so on.

In total, this requires at most $7 \cdot 32 = 224$ additional guesses. (Actually, $7 \cdot 31$, since if all of the first 31 attempts yield the same timing, then she knows that the untried 32nd value is correct.) Thus, the total effort is well under 10^3 attempts.

The size-of-memory side channel discussed above does not apply in this case. While it potentially leaks length information, it does not leak content information.

(c) (12 points) Suppose `www.lamesec.com` instead uses the following code to validate authentication attempts:

```
def CheckPassword(account, submitted_password):
  if len(submitted_password) != len(account.password):
    return False
  num_correct = 0
  num_incorrect = 0
  for i in range(len(submitted_password)):
    if submitted_password[i] == account.password[i]:
      num_correct = num_correct + 1
    if submitted_password[i] != account.password[i]:
      num_incorrect = num_incorrect + 1
  return num_incorrect == 0
```

Given this change, now how many authentication attempts will suffice for Mallory to determine the *length* of Alice's password? Choose the **MINIMUM** such number of attempts that *guarantees* success for Mallory:

- ○ 1 attempt
- ● 10^3 attempts
- ○ 10^6 attempts
- ○ 10^9 attempts
- ○ 10^{12} attempts
- ○ Mallory can do this but will need more than 10^{12} attempts
- ○ Mallory cannot do this

Solution: For the revised code, the check for length at the beginning hasn't changed. Thus, the same logic applies as before; it takes 6 attempts to determine the length.

We again also gave full credit to answers stating it could be done in 1 attempt, on the assumption that these reflect use of a size-of-memory side channel.

(d) (16 points) Continuing with the new version of `CheckPassword`, now how many authentication attempts will suffice for Mallory to determine the *exact value* of Alice's password? Choose the **MINIMUM** such number of attempts that *guarantees* success for Mallory:

- ○ 1 attempt
- ○ 10^3 attempts
- ○ 10^6 attempts
- ○ 10^9 attempts
- ● 10^{12} attempts
- ○ Mallory can do this but will need more than 10^{12} attempts
- ○ Mallory cannot do this

Solution: Once the execution has proceeded beyond the length test, the revised code completely eliminates the instruction-counter side channel: no matter whether any of the characters match or do not match, the same number of instructions are executed.

Thus, all Mallory can do at this point is resort to brute force. This requires $32^7 = (2^5)^7 = 2^{35}$ attempts. Approximating $2^{10} \approx 10^3$, this corresponds to $2^5 \cdot 10^9$. So 10^9 attempts won't suffice, but 10^{12} will.