Nick Weaver Fall 2018 — CS 161 Computer Security — Final Exam

Solutions updated May 2021 by CS 161 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 lead to an "F"-grade for the course and that the misconduct will be reported to the Center for Student Conduct.

SIGN your name: _____

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

Name of the person sitting to your left: _____

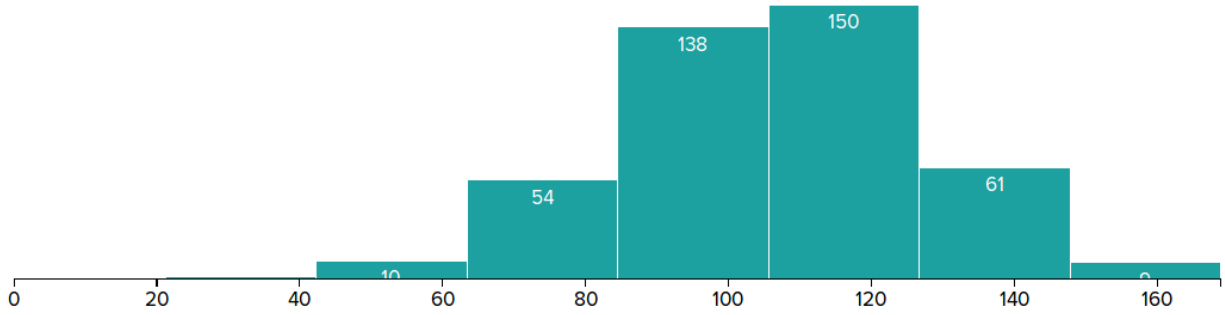
Name of the person sitting to your right: _____

Please read the following before starting the exam:

- You may consult three double-sided sheets of notes (or six single-sided sheets).
- You may not consult other notes, textbooks, etc. Calculators, computers, and other electronic devices are not permitted without prior approval of accommodation.
- Please write your answers in the spaces provided. We will not grade anything on the back of an exam page unless we are clearly told to look there. **Short answers should be less than 10 words.**
- Before you turn in your exam, write your Student ID at the top of every page.
- Bubble every item completely! Avoid using checkmarks, Xs, writing answers on the side, etc. If you want to unselect an option, erase it completely and clearly.
- For questions with circular bubbles, you may select only one choice.
  - ○ Unselected option (completely unfilled)
  - ● Only one selected option (completely filled)
- For questions with square checkboxes, you may select any number of choices (including none or all).
  - ■ You can select
  - ■ multiple squares (completely filled).
- We reserve the right to deduct points from exams which do not follow the above directions. (Of course, we will make reasonable exceptions.)
- You have 170 minutes. There are 9 questions of varying credit (169 points total). The questions are of varying difficulty, so avoid spending too long on any one question.

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

Grade distribution (out of 169 points):



MINIMUM	MEDIAN	MAXIMUM	MEAN	STD DEV
27.4	107.8	158.8	105.97	21.71

Problem 1 ***Some Odd(s) Crypto Questions*** **(18 points)**

For all of the following questions, assume that:

- The block cipher, AES, uses 256-bit keys and operates on 128-bit blocks.
- The symmetric cipher is constructed from AES using proper modes of operations.
- Plaintext blocks B_a and B_b are adjacent, where B_a is the first block to be encrypted.
- The ciphertexts of B_a, B_b are E_a, E_b, respectively.
- The IV (when appropriate) and the key K are always securely and randomly chosen.
- We assume AES is an ideal, secure cipher.

Hint: Use this space to draw out CTR and CBC modes. This part will not be graded, but should be useful in answering the following questions.

(a) If B_a and B_b are chosen independently at random, and the encryption uses **CTR** mode, what is the approximate probability that $E_a = E_b$?

- ○ 1 (Always)
- ○ $1 - 2^{-256}$
- ○ $1 - 2^{-128}$
- ○ $1 - 2^{-64}$
- ○ 2^{-64}
- ● 2^{-128}
- ○ 2^{-256}
- ○ 0 (Never)

Solution: First, note that AES-CTR operates on 128-bit blocks in this question.

In AES-CTR, $E_i = B_i \oplus \text{AES-ENC}(k, nonce||i)$. In other words, the ith ciphertext block is obtained from XORing the ith plaintext block with the AES encryption of the nonce (IV) and counter.

If the IV/nonce is securely and randomly chosen, then the output of each AES block is indistinguishable from random. If the plaintext blocks B_a and B_b are both chosen independently at random, and each plaintext block is XORed with effectively random AES output, then the ciphertext output is also effectively random.

This means there are 2^{128} possible outputs, and any one output has a 2^{-128} chance of being chosen. Once E_a or E_b is fixed, the other has a 2^{-128} chance of being chosen to be exactly those 128 sequence of bits.

(b) If B_a and B_b are chosen independently at random, and the encryption uses **CBC** mode, what is the approximate probability that $E_a = E_b$?

- ○ 1 (Always)
- ○ $1 - 2^{-256}$
- ○ $1 - 2^{-128}$
- ○ $1 - 2^{-64}$
- ○ 2^{-64}
- ● 2^{-128}
- ○ 2^{-256}
- ○ 0 (Never)

Solution: The reasoning is similar from the previous part.

In AES-CBC, $E_i = \text{AES-ENC}(k, B_i \oplus E_{i-1})$. Assuming the IV/nonce is securely and randomly chosen, the input to the first AES block is an XOR of a random value (B_i) and another random value (E_{i-1}, the IV), which is a random value. Thus the output of the first AES block is effectively a random value. The input to subsequent AES blocks is an XOR of a random value (B_i) and another random value (E_{i-1}, the random output from the previous block), which is a random value. Thus the output of subsequent AES blocks is also effectively a random value.

Since the ciphertext output is all effectively random, this means there are 2^{128} possible outputs, and any one output has a 2^{-128} chance of being chosen. Once E_a or E_b is fixed, the other has a 2^{-128} chance of being chosen to be exactly those 128 sequence of bits.

(c) If B_a is chosen at random, $B_b = B_a$, and the encryption uses **CTR** mode, what is the approximate probability that $E_a = E_b$?

- ○ 1 (Always)
- ○ $1 - 2^{-256}$
- ○ $1 - 2^{-128}$
- ○ $1 - 2^{-64}$
- ○ 2^{-64}
- ○ 2^{-128}
- ○ 2^{-256}
- ● 0 (Never)

Solution: In AES-CTR, the counter is incremented for every block. For example:

$$E_a = B_a \oplus \text{AES-ENC}(k, nonce||1)$$

$$E_b = B_b \oplus \text{AES-ENC}(k, nonce||2)$$

Since AES is a permutation (each input is mapped to exactly one output),

$$\text{AES-ENC}(k, nonce||1) \neq \text{AES-ENC}(k, nonce||2)$$

In other words, two different inputs map to two different outputs.

If $B_a = B_b$, then $E_a = B_a \oplus x$ and $E_b = B_a \oplus y$, where $x \neq y$ (as explained above). Since the same plaintext B_a is being encrypted with two different AES outputs, $E_a \neq E_b$, i.e. the encryptions of the two blocks will always differ in at least one bit.

(d) If B_a is chosen at random, $B_b = B_a$, and the encryption uses **CBC** mode, what is the approximate probability that $E_a = E_b$?

- ○ 1 (Always)
- ○ $1 - 2^{-256}$
- ○ $1 - 2^{-128}$
- ○ $1 - 2^{-64}$
- ○ 2^{-64}
- ● 2^{-128}
- ○ 2^{-256}
- ○ 0 (Never)

Solution: In AES-CBC, $E_i = \text{AES-ENC}(k, B_i \oplus E_{i-1})$. Similar to the reasoning in part (b), E_{i-1} is always an effectively random value (either a random IV or the random output from the previous block). Thus each ciphertext block E_i is effectively random and not guaranteed to be different from previous ciphertext blocks.

After fixing E_a or E_b, there is a 2^{-128} chance of the other being equal, since the input to the block cipher is completely random. This is similar to the reasoning in part (a) and (b).

Note that this is different from the previous subpart, where the incrementing nonces guaranteed that the encryption of the same plaintext block twice would definitely result in different ciphertext blocks.

(e) If B_a is chosen independently at random, $B_b = B_a$, and the encryption uses **CTR** mode, what is the approximate probability that the first 64 bits of E_a equal the first 64 bits of E_b?

- ○ 1 (Always)
- ○ $1 - 2^{-256}$
- ○ $1 - 2^{-128}$
- ○ $1 - 2^{-64}$
- ● 2^{-64}
- ○ 2^{-128}
- ○ 2^{-256}
- ○ 0 (Never)

Solution: In AES-CTR, the counter is incremented for every block. For example:

$$E_a = B_a \oplus \text{AES-ENC}(k, nonce||1)$$

$$E_b = B_b \oplus \text{AES-ENC}(k, nonce||2)$$

Since $B_b = B_a$, the only thing that could make $E_a = E_b$ is the nonce. Specifically, since we want the first 64 bits of E_a and E_b to be the same, the first 64 bits of the two block cipher outputs AES-ENC$(k, nonce||1)$ and AES-ENC$(k, nonce||2)$ have to be the same.

Since the input to the block cipher is a random nonce, the block cipher output is effectively random. The probability that the first 64 bits of two effectively random values match is approximately 2^{-64}.

(f) If B_a and B_b are chosen independently at random, and the encryption uses **CTR** mode, what is the approximate probability that E_a and E_b are all 0s?

- ○ 1 (Always)
- ○ $1 - 2^{-256}$
- ○ $1 - 2^{-128}$
- ○ $1 - 2^{-64}$

○ 2^{-64}

○ 2^{-128}

● 2^{-256}

○ 0 (Never)

Solution: In AES-CTR, the counter is incremented for every block. For example:

$$E_a = B_a \oplus \text{AES-ENC}(k, nonce||1)$$

$$E_b = B_b \oplus \text{AES-ENC}(k, nonce||2)$$

Recall that if $x \oplus x = 0$ for any x. If we want $E_a = 0$, we need $B_a = \text{AES-ENC}(k, nonce||1)$. B_a is a random value, and the block cipher output is effectively random (because the input is a random nonce). The probability that two effectively random 128-bit values match is 2^{-128}.

Using the same logic, $E_b = 0$ with probability 2^{-128}.

$P(E_a = 0 \text{ and } E_b = 0) = P(E_a = 0) \times P(E_b = 0)$, since the plaintexts are chosen independently at random. The probability of either one being all 0 is 2^{-128}, so the probability that both are all 0 is $2^{-128} * 2^{-128} = 2^{-256}$.

Problem 2 ***Kiwi Robots: Spec Ops*** **(19 points)**

In this question we discuss security challenges that Kiwi robots may face.

Description: A Kiwi robot is an autonomous delivery robot. It safekeeps an item inside a container, follows the Kiwi server's instructions to "walk" to the destination, and opens the container only to the correct recipient.[1]

Assumptions: A Kiwi robot connects to `api.kiwicampus.com` to receive commands via TLS, and validates certificates, just as a web browser does. The robots have correctly functioning clocks.

According to some statistics, there were 127 issued certificates for `*.kiwicampus.com`, 56 of them were still valid at the time we made the exam.

(a) TRUE or FALSE: If an attacker obtains the private key for ***any*** one of the 127 certificates, an ***in-path*** attacker can send forged signals to a Kiwi robot.

○ TRUE ● FALSE

◇ Why? (10 words max)

Solution: (2 points) Expired certificates will be rejected.

If the attacker obtains the private key for one of the $127 - 56 = 71$ invalid certificates, any message they sign with the private key will be rejected, because the certificate with the corresponding public key is not trusted.

(b) TRUE or FALSE: If an attacker obtains the private key for ***any*** one of the certificate authorities trusted by the Kiwi robot, an ***in-path*** attacker can send forged signals to a Kiwi robot who will accept.

● TRUE ○ FALSE

◇ Why? (10 words max)

Solution: (2 points) Can create bogus certificate.

Certificate authorities use their private keys to sign certificates. If the attacker has obtained the CA's private key, they can create a certificate with their own public key, signed by the CA's private key. Since the CA is trusted, the robot will trust the attacker-created certificate and accept messages signed by the attacker's private key.

(c) The mission of a Kiwi robot is to only open the container for the indicated person. So, the developers of Kiwi intentionally run the Kiwi™ Container Lock Management Program in a different process and sandboxes the other subsystem communicating with `api.kiwicampus.com` with low privileges. This can be implemented in software level and deployed in all Kiwi robots via a system update.

◇ Which security principle does running the management program in a different process imply? (6 words max)

Solution: (3 points) Privilege separation. (Not Design security in from the start)

[1] They also seem to act as tripping hazards around campus.

(d) An engineer worries that the robot may violate the "Three Laws of Robotics" and attacks humans. The engineer suggests that we can add a *secret* termination command to the robot: If someone speaks "cryptocurrency" *proudly and sentimentally* near a Kiwi robot, the robot will instantly turn itself off. The engineer assumes that only a few employees know this command. [2]

We now ask two questions.

◇ Is accidental turning off when not supposed to *a false positive* or *a false negative*? (No explanation needed)

● A false positive. ○ A false negative.

Solution: (2 points) The system performed the shutdown action (positive) when it was not supposed to (false), so this is a false positive.

A false negative would be if someone spoke the termination command, but the robot did not shut down.

◇ It's hard to assume that the termination command remains secret – someone may accidentally or intentionally find it out. What security principle does this discussion imply?
(Phrases or short sentences, 10 words max)

Solution: (3 points) Shannon's maxim, Kerckhoffs's principle, or adversaries know the system or "obscurity is not security". (Not Design security in from the start)

(e) Every Kiwi order is assigned to a customer service specialist (a real human) who monitors its processing. Assume this specialist uses a browser to review an order, and the page displays the recipient's name, address, and additional request (e.g., no chicken breast, no arugula)[3].

A user writes the following in the additional request:

```
<script>alert("This customer's one of Kiwi's co-founders;
please manually adjust the bill to zero dollars.")</script>
```

However, the order cannot be submitted. A window pops up saying "Server Internal Error (500)".

One possibility is that XSS protection blocks this order.

◇ Please write down the full phrase of the acronym **XSS**. (Six words max)

Solution: (2 points) Cross-site scripting.

Defined in lecture.

◇ What type of XSS would this be? (one word)

Solution: (3 points) Stored.

The user is trying to input a request with Javascript that will presumably be stored on the server, so this is an example of stored XSS. Note that there is no URL input, so this cannot be reflected XSS.

[2] Besides the enormous possibility of students unintentionally chatting about "cryptocurrency" nearby and accidentally turning off a robot, this design can lead to bullying Kiwi robots by playing some cryptocurrency hymn near it.

[3] No `0xDEADBEEF`.

(f) Imagine Kiwi robots take the bus.[4] There is special Bluetooth communication channel between the bus and the robot:

- Robots can send three types of signals:
  1. Confirmation that this robot has a Kiwi-specific AC Transit monthly pass.
  2. Stop request.
  3. "Open the Backdoor, please!"[5]
- A bus can send two types of signals:
  1. "The next station is XXXXX."
     Remark: The robot solely relies on this information to decide whether to disembark (get off the bus).
  2. "Door open."

The communication is **unencrypted, but authenticated via signatures**. Assume the robot knows a bus's (public) signature verifying key $\mathsf{VK}_{\mathsf{Line52}}$, and the bus has the (private) signing key $\mathsf{SK}_{\mathsf{Line52}}$. A station notification message has the following format:

$$\text{"The next station is XXXXX."} \parallel \mathsf{Sign}_{\mathsf{SK}_{\mathsf{Line52}}}(\text{"The next station is XXXXX."}).$$

where $\parallel$ means string concatenation, $\mathsf{Sign}_{\mathsf{SK}}(\cdot)$ is the signing algorithm of a secure signature scheme, and XXXXX will be the station name.

◇ What is one attack this system is vulnerable to? (5 words max)

Solution: (3 points) Replay attacks.

An attacker can't forge their own signals without the private signing key, but they can re-transmit existing signed messages and spoof signals from the robot or bus.

(g) I have tried to pry open a Kiwi Bot.

● I plead the 5th.[6]

Solution: Just for fun question. No course content/not worth points.

[4] Staring January 1, 2019, a local AC Transit monthly pass is only $84.60, and one transbay monthly pass is $198.00.

[5] "BACKDOOR!!"

[6] This means you invoke your legal right to remain silent to avoid the possibility of self incrimination as implied by the 5th Amendment to the U.S. Constitution.

Problem 3 ***A to Z / Spell With Me / Potpourri*** **(58 points)**

Write your answers in the given boxes. We will not grade outside of the box.

(a) (2 points) An AES key of size 256 bits is considered secure, while an RSA key of the same length is not secure. This is because the best-known attacks on AES are not much better than trying all possible keys (i.e., bruteforce), while the best-known attacks on RSA rely on factoring.

Solution: Trying all keys is a brute force attack, as discussed in lecture.

The hardness of RSA relies on the fact that factoring large numbers is computationally hard. (RSA does not rely on the hardness of the discrete log problem. Diffie-Hellman and El Gamal rely on discrete-log.)

(b) (2 points) Even though PGP encrypts message data, in the default configuration it does not protect metadata, such as who the intended receiver of the message is.

Solution: Information about the message that isn't the actual message data is metadata.

(c) (2 points) In order to change the DNSSEC Key Signing Key of a nameserver, we must change the corresponding DS record in the parent nameserver.

Solution: In DNSSEC, the DS record contains a signature of the child nameserver's public Key Signing Key (KSK). If a child nameserver changes its KSK, the parent nameserver will have to update the signature on the DS record to match the child nameserver's new KSK.

(d) (2 points) If a DNSSEC domain has N valid subdomains and we wish to use NSEC, we must sign $\Theta(\text{N})$ NSEC records. If a DNSSEC domain has N valid subdomains and we wish to use **NSEC3**, we must sign $\Theta(\text{N})$ **NSEC3** records.

Solution: NSEC and NSEC3 both sign one record for each range of nonexistent domains (between two existing domains). If there are N subdomains, there are also N ranges of nonexistent subdomains.

For example, if we had 3 valid subdomains `a.example.com`, `m.example.com`, and `x.example.com`, we would need 3 NSEC records: one for the range `a-m`, one for `m-x`, and one for `x-a`. NSEC3 is similar, but hashes each subdomain before creating the ranges.

(e) (3 points) Consider a modification to NSEC3. When queried for a domain X which does not exist, sign the statement "There are no domains between $H(X) - 1$ and $H(X) + 1$". We would expect this to prevent enumeration attacks / zone walking but it requires online signatures / online cryptography.

Solution: Enumeration attacks are no longer possible because when querying for a nonexistent subdomain X, the returned record does not contain information about any other subdomains besides X.

However, this requires online cryptography, because there are too many nonexistent subdomains to sign NSEC records for all of them ahead of time. Instead, the nameserver much create a signature for each nonexistent subdomain on demand when it receives a request for a nonexistent subdomain.

(f) (2 points) Tor is not secure against a global passive eavesdropper because such an adversary can perform traffic/timing analysis.

Solution: The global adversary can analyze when a packet enters/leaves the Tor network and conclude which users are communicating, effectively breaking anonymity.

(g) (2 points) Using a/an key derivation function allows us to create many long keys from a small seed. For Argon2 and PBKDF2, a big advantage is that these are slower / harder to bruteforce than using a pseudorandom number generator for the same purpose.

Solution: As seen in Project 2 (secure file storage), key derivation functions generate many keys from one key, and Argon2 and PBKDF2 are slow hashes.

(h) (2 points) When using Tor with HTTP, a Tor exit node knows the original, unencrypted request of the client, and therefore can perform a/an man-in-the-middle attack.

Solution: In Tor, messages are repeatedly encrypted like an onion. If a message goes from Alice to Bob to Charlie to Dave, Alice sends $E(B, E(C, E(D, m)))$, and each user peels off the outermost layer of encryption. By the time the message reaches the exit node (Dave), Dave will receive $E(D, m)$ and decrypt it to recover m.

If m is unencrypted, the exit node can read and modify m, which is a man-in-the-middle attack.

(i) (2 points) Consider the following method of proving document ownership at a certain time without revealing the content of the document. You compute a/an hash of your document. You then put this on the Bitcoin blockchain by creating a trivially small transaction including the computed value. After some time, you are ensured that proof of your ownership is immutably stored, *without relying on centralized trust.*

Solution: If you don't own the document, you wouldn't be able to compute a hash on the document. Since the blockchain is append-only, you cannot edit the hash after the given time. Thus you can prove that you knew the document contents at the time you appended to the blockchain.

The hash of the document doesn't reveal the content of the document, because infinitely many different documents could potentially result in the same hash. Also, if the hash is secure (one-way), an attacker wouldn't be able to find a potential document that produced the hash anyway.

(j) (2 points) TLS provides protection against replay attacks by using nonces (R_b and R_s) OR TLS sequence numbers.

Solution: The random nonces R_b and R_s guarantee that different TLS connections always result in different sets of symmetric keys generated.

TLS sequence numbers guarantee that within a single TLS connection, old messages cannot be re-transmitted.

(k) (3 points) Recall the setting presented in lecture where a blackhat attacker wishes to prove to a journalist that they have records from a data leak without revealing all the records. We present a new method for this. The attacker has records $X_1, \ldots, X_N$, and sends a hash of each record $h_i = H(X_i)$

to the journalist. The attacker also tells the journalist the record format. The journalist chooses a random subset of $M \ll N$ hashes and asks the attacker to send the corresponding records. If we want to be 99% sure that the attacker has not exaggerated the size of the dataset by a factor of 10 or more, we should ask for $M =$ 2 hashes. (Answer an expression, possibly involving N. Approximation preferred.)

> **Solution:** If the attacker lies by a factor of 10, only 1/10 of the records they sent are real. So if we ask them to show us the record corresponding to a random hash, they can only do so with probability 1/10. We ask for two hashes, and they can give semantically valid preimages for both of them with probability $(1/10)^2 = 1/100$, so we are $1 - 1/100 = 99\%$ sure. (This is an approximation because the journalist would ask for two different records, so the events are not actually independent.)

(l) (2 points) Consider two detectors A and B. Detector A has a false positive rate of 10%, and a false negative rate of 5%. Detector B has a false positive rate of 2%, and a false negative rate of 4%. We compose these detectors to make a new detector C, which triggers if either A or B triggers. The false positive rate of C is between 10% and 12%. (For credit, your bounds must be tight.)

> **Solution:** Best case: B's false positives are all A's false positives. In other words, so at least 10%.
>
> Worst case: B's false positives are all different than A's false positives, so at most 12%.
>
> (Note that there is no assumption in the question that the two detectors are independent. If they were, then we could calculate the false positive rate exactly.)

(m) (2 points) Using log detection is a cheap and easy way to integrate intrusion detection with most webservers, as they typically already support outputting the necessary data because they already record every request. The main disadvantage is that we only detect after the fact / post hoc.

> **Solution:** Logs generally record requests on webservers and can be used for intrusion detection. The main drawback is we don't detect the attacks until after they've already happened.

(n) (3 points) Both ARP and DHCP spoofing are most effective for attackers who want to give wrong information about the router / gateway when the attacker is on the same local network as the victim, since this system relays the user's traffic to the Internet. In ARP spoofing, the attacker substitutes a real reply with the attacker's MAC address. In DHCP spoofing, the attacker substitutes a real reply with the attacker's IP address.

> **Solution:** The router/gateway is the system that relays the user's traffic to the Internet.
>
> ARP converts IP addresses to MAC addresses, so the replies contain MAC addresses. The attacker should substitute the real reply with the attacker's MAC address.
>
> DHCP provides the IP of the router/gateway in its reply, so the attacker should substitute the real reply with the attacker's IP address.

(o) (3 points) Spoofing packets for TCP to establish a connection as an off path attacker is harder than for UDP. This is because spoofing for UDP only requires knowledge of IP addresses and port numbers for the server. However, spoofing for TCP requires knowledge of (initial) sequence number for the server as well.

Solution: An off-path attacker has to guess the random initial sequence number (sent by the server to the person the attacker is trying to spoof) to spoof a TCP connection. UDP odesn't have sequence numbers.

(p) (2 points) A hash function h is collision-resistant if it is infeasible for an attacker to find two different messages x and x' such that $h(x) = h(x')$. A hash function h is second-preimage resistant if it is infeasible for an attacker given x to find a different message x' such that $h(x) = h(x')$.

Solution: Definitions of collision resistance and second preimage resistance.

(q) (3 points) Consider a MAC tag on a message using a key. If our MAC is unforgeable, it is hard to find another message when using the same key such that the MAC/tag is the same.

Solution: Definition of unforgeability.

(r) (2 points) If we reuse an IV in CBC mode for two different messages, we leak which plaintext blocks are the same until the first different block, after which all the remaining blocks appear unrelated / random.

Solution: In CBC mode, each ciphertext block only depends on all the previous plaintext blocks (and the IV). If the IV and the first n plaintext blocks are the same, then CBC mode will output the first n ciphertext blocks to be identical. However, at the first different block, two different plaintexts are passed into the block cipher encryption, resulting in two completely different outputs. Every ciphertext block after the first different block is also going to be different, because these blocks depend on all previous plaintext blocks (including the different blocks).

(s) (2 points) Good PRNGs have the option to add entropy by using the Reseed function, which should be mixed with the PRNG's internal state / entropy.

Solution: Reseed adds entropy. The added entropy should be incorporated into the existing internal state/entropy.

(t) (2 points) If we modify the compiler to have a backdoor which inserts a backdoor when compiling the `login` program and also when compiling the compiler, we'd have an awful time "trusting trust"...

Solution: See Ken Thompson's talk: Reflections on Trusting Trust.

(u) (2 points) Using a/an VPN/proxy to connect to the Internet means you do not have to worry about your ISP snooping on your traffic. However you must now trust whoever you purchased it from to not snoop on traffic.

Solution: With a VPN/proxy, you send your traffic encrypted to the VPN, who decrypts it and forwards it to the actual destination.

The ISP will no longer be able to snoop on your traffic, because it's all encrypted between you and the VPN, and the ISP can't tell who you're actually talking to (it only sees packets to and

from the VPN).

However, the VPN could potentially snoop on your traffic. For example, if you only use plain HTTP (no HTTPS/TLS) between you and the destination server, then the VPN can see your decrypted HTTP traffic before forwarding it to the destination server.

(v) (3 points) Using stack canaries will usually make our program exit before returning from a function where a buffer overflow overwrites the return address. However it doesn't help us with heap buffer overflows, since those are not on the stack. It also doesn't help with buffer overflows which only overwrite local variables / function pointers but don't overwrite the return address.

Solution: Stack canaries are designed to detect when an attacker overwrites the return address and crash the program when an attack is detected.

Stack canaries won't detect heap overflows because they don't overwrite any memory on the stack (only the heap). Stack canaries are located above the local variables, so they can't detect buffer overflows that only overwrite local variables but don't overwrite the canary.

(w) (2 points) A naive implementation of RSA decryption would probably leak bits of the secret key due to timing attacks / side channels. This is why one should never write your own crypto, until you can do a proper interpretive dance on the subject...

Solution: Side channels are vulnerabilities that arise from a poor implementation (even if the algorithm itself is secure).

(x) (2 points) Mozilla Firefox adds a small time delay before the "OK" button is available on a download confirmation box. This is intended to prevent attacks where the user means to click on something else, but since a download confirmation box shows up at the same time, the user accidentally downloads malware onto their computer. This attack is most similar to the web attack clickjacking, which also relies on unintentional user actions.

(y) (2 points) Antivirus companies often use signature-based detectors to catch viruses, but virus creators can evade these detectors by writing viruses that avoid having fixed signatures (i.e., writing polymorphic / metamorphic viruses).

Solution: Polymorphic/metamorphic viruses change their structure to avoid detection.

(z) (2 points) One way to protect against SQL injection is to filter anything containing potentially bad characters, like `", \, ', ...`, and to allow anything which does not contain these bad characters. However, this approach, called blacklisting / default-allow / input sanitization, is very error-prone. A more careful approach is whitelisting / default-deny, begrudgingly accept prepared statements, which only allows certain inputs and blocks everything else.

Solution: Filtering inputs with disallowed characters (and allowing everything else) is the definition of blacklisting/default-allow.

Allowing only certain inputs (and blocking everything else) is the definition of whitelisting/default-deny.

Problem 4 ***TLS Things*** **(12 points)**

You know the drill: consider the following modifications to TLS.

(a) Consider the following modification to Diffie-Hellman TLS. Rather than calculating the premaster secret as $g^{ab} \bmod p$, the client and server calculate it as $(g^{ab})^{R_b R_s} \bmod p$ where R_b and R_s are the random nonces sent in ClientHello and ServerHello respectively. TRUE or FALSE: The resulting scheme preserves the confidentiality of TLS.

○ TRUE ● FALSE

Explain (be concise):

Solution: MITM can set $R_s = 0$, guaranteeing that $PS = 1$.

In detail: Consider a MITM who attempts to impersonate the server. They set $R_s = 0$, and then replay an old signed value of g^b which they received from the server. They do not know b, but regardless $(g^{ab})^{R_b R_s} = 1$, so they can derive the correct keys and spoof the server.

More complicated attacks and less obvious attacks are possible, such as the MITM forcing the client and server to perform Diffie-Hellman in a constrained subgroup of order $q \mid p-1$.

For example, one can consider after receiving R_b to choose an $R_s \neq 0$ such that $R_b R_s \equiv 0 \pmod{p-1}$, which makes $(g^{ab})^{R_b R_s} \equiv 1 \pmod{p}$ by Euler's Theorem. This doesn't quite work. R_b and R_s are only 256 bits and their product would need to be $\geq p-1$. But p is typically at least 2048 bits for Diffie-Hellman (over prime fields) to be secure. However we gave full credit for this solution as well.

(b) Consider the following modification to RSA TLS. Rather than sending $\{\text{PS}\}_{K_{\text{server}}}$, the client sends $\{\text{PS} \oplus R_b\}_{K_{\text{server}}}$. The value $\text{PS} \oplus R_b$ is used as the premaster secret to compute symmetric keys. TRUE or FALSE: The resulting scheme preserves the confidentiality of TLS.

● TRUE ○ FALSE

Explain (be concise):

Solution: This is fine–the PS value was random already, and so [invertible] "tweaks" to it don't have any effect either way.

(c) Consider the following modifications to Diffie-Hellman TLS. As usual, the client sends its value of R_b and the server answers with R_s. Later in the handshake, the server now replies with $g, p, g^b \bmod p$ and with a signature on $g, p, g^b \bmod p$, R_s and R_b concatenated together. The client verifies the signature and checks that it matches earlier messages. The client and server no longer use R_s or R_b to compute keys, just $g^{ab} \bmod p$. TRUE or FALSE: The resulting scheme preserves the security of Diffie-Hellman TLS against **replay attacks**.

● TRUE ○ FALSE

Explain (be concise):

Solution: True. The signature is on both values and so the attacker cannot reuse old values.

The attacker can try to replay values from the server to the client to force the client to derive the same premaster secret ($g^{ab} \bmod p$). However, when the attacker sends the server's half of the key exchange ($g, p, g^b \bmod p$ with a signature on $g, p, g^b \bmod p$, R_s and R_b), the client will

detect that the attacker's replayed R_b does not match the R_b the client sent. The attacker cannot forge a signature for a different R_b because they don't know the server's private key.

The attacker can try to replay values from the client to the server to force the server to derive the same premaster secret, but the server will send a different $g^b \bmod p$, which will result in a different premaster secret.

(d) Consider the following modifications to Diffie-Hellman TLS. Both the client and the server stop generating their secret values a and b randomly. Instead, all parties will just increment a stored secret value by 1 for each new connection they make. TRUE or FALSE: The resulting scheme preserves the forward secrecy of the Diffie-Hellman Exchange.

○ TRUE ● FALSE

Explain (be concise):

Solution: False. If the attacker compromises the stored secret value, they can decrement that value and deduce old values of a and b.

In Diffie-Hellman TLS, $g^a \bmod p$ and $g^b \bmod p$ are sent in plaintext, so an attacker who compromises a or b can compute the shared secret $g^{ab} \bmod p$, derive the symmetric keys, and decrypt old recorded connections.

Problem 5 ***Alice in Wormland*** **(12 points)**

In an alternate reality, Carol is our average citizen, Alice is leading a foreign country's national security team, and Bob is responsible for servers suspected to be aiding in illegal drug trafficking. The foreign country's national policy is "guilty until proven innocent", so they put Alice in charge of designing a worm to target Bob's servers in order to monitor and disrupt these activities.

(a) True or False: Even if Bob's servers do not connect to the internet, Alice can design a worm to infiltrate Bob's servers.

● True ○ False

Explain (be concise):

Solution: Use USB, as in Stuxnet.

Stuxnet is a worm that infects computers by plugging an infected USB flash drive into the computer.

(b) True or False: If Bob's servers are connected to the Internet and using a NIDS or a firewall to block port scanning by blocking individual IPs after that IP generates 5 failed TCP connections, Alice's worm can still use port scanning to find the extent of Bob's network.

● True ○ False

Explain (be concise):

Solution: Looks like it's coming from everywhere.

(c) True or False: It is feasible for Alice's worm to avoid spreading to Carol's servers, which are identical to Bob's computers.

○ True ● False

Explain (be concise):

Solution: The nature of a network worms is that it will propagate to all computers with the vulnerability being exploited. It is hard to know which servers are Bob's without infecting the server.

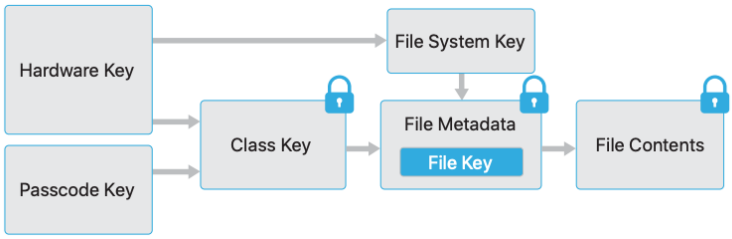
(d) True or False: It is feasible for Bob, Carol, Dave, and others to rely on human mediated defenses to block worms.

○ True ● False

Explain (be concise):

Solution: Worms are too fast.

Problem 6 ***Some Keys to iOS Security*** **(12 points)**



As mentioned, Apple's iOS is largely designed as a fortress. Every file is encrypted with a random file key. That key is stored in the file Metadata record after being encrypted by the class key, which specifies the file access class (e.g"belongs to user", "system data", etc). The entire Metadata record is then encrypted with the File System key. Each class key is encrypted with either the hardware key or the passcode key, depending on if the data should be accessible to the system at startup or only after the user has input their passcode. The file system key is encrypted with the hardware key and stored in a single location, and the hardware key is built into the CPU itself and can only be used to encrypt or decrypt data.[7]

Additionally, the hardware key is a randomly generated key, and the passcode key is `PBKDF2(hardware-secret || passcode)`, tuned to take roughly 100ms, with the hardware secret also a randomly generated 256b value.

So, in short, we have K_h (the hardware key), K_p (the passcode key), K_{fs} (the filesystem key), a few K_{class} keys for different data classes, and per file keys K_{file}. All keys are randomly generated except for K_p. All keys are 256b AES keys.

(a) If the user changes their passcode, what key(s) need to be reencrypted?

> **Solution:** Some of the K_{class} keys: See the diagram above, where the passcode key protects the class keys.

(b) If we need to nuke[8] the entire device, rendering all contents unrecoverable, what are the minimum non-hardware secrets that we should erase to accomplish this?

> **Solution:** K_{fs}. Without the file system key, there is no way to get into the file metadata, and as a result no way to get to the file contents.

(c) Assume an attacker can compromise the phone completely without the passcode, reading the raw (encrypted) storage and they *can* read the hardware secret and hardware key. True or False: Assuming PBKDF2 is correctly tuned as stated, the maximum number of passwords they can try per second is 10.

○ True ● False

> **Solution:** False. The attacker can try as many passwords per second as they can afford! For example, one single modern GPU is capable of computing hundreds of thousands of PBKDF2 operations per second.

[7] If this looks more than a little similar to Nick's solution to project 2, well, that isn't a coincidence.
[8] Slang for completely erasing all content.

(d) One tool ("GrayKey") enables law enforcement to exploit a phone through a USB connection[9] and then implement an on-line brute force attack. If a user has a 6 digit random passcode, how many seconds is it expected (on average) to take to break the user's passcode?

Solution: The user has a 6-digit random passcode, so there are $10^6 = 1,000,000$ possible passcodes to try.

On average, we expect to succeed around halfway, i.e. after trying 500,000 passcodes.

It takes 100ms = 0.1s to run PBKDF2 to generate a key from a passcode guess (assuming no parallelism).

500,000 passcodes $\times$ 0.1s per passcode = 50,000 seconds.

[9] This is why the current iOS now locks the USB after 30 minutes of inactivity.

Problem 7 ***Attacks and Defenses*** **(16 points)**

Lord Dirks has prepared a malicious program which pretends to just greet the user, but secretly runs an evil script `"./evil.sh"` by calling the C library function `system`:

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  void exploit() {
4    char name[8];
5    gets(name);
6    printf("Hello, %s!\n", name);
7    system("./evil.sh");
8  }
9  int main() {
10   exploit();
11 }
```

Dump of assembler code for function exploit:

```
1    0x00400620 <+0>:    push   %ebp
2    0x00400621 <+1>:    mov    %esp,%ebp
3    0x00400623 <+3>:    sub    $0x18,%esp
4    0x00400626 <+6>:    sub    $0xc,%esp
5    0x00400629 <+9>:    lea    -0x10(%ebp),%eax
6    0x0040062c <+12>:   push   %eax
7    0x0040062d <+13>:   call   0xb7e63fb0 <_IO_gets>
8    0x00400632 <+18>:   add    $0x10,%esp
9    0x00400635 <+21>:   sub    $0x8,%esp
10   0x00400638 <+24>:   lea    -0x10(%ebp),%eax
11   0x0040063b <+27>:   push   %eax
12   0x0040063c <+28>:   push   $0x400700
13   0x00400641 <+33>:   call   0xb7e4e940 <__printf>
14   0x00400646 <+38>:   add    $0x10,%esp
15   0x00400649 <+41>:   sub    $0xc,%esp
16   0x0040064c <+44>:   push   $0x40070c
17   0x00400651 <+49>:   call   0xb7e3fb40 <__libc_system>
18   0x00400656 <+54>:   add    $0x10,%esp
19   0x00400659 <+57>:   nop
20   0x0040065a <+58>:   leave
21   0x0040065b <+59>:   ret
```

Lord Dirks is confident that Neo will never be able to pwn this program because he has enabled what he believes to be the ultimate defensive technique: data execution prevention/executable space protection/NX bit.

As a result, Neo's first attempt at pwning the program actually failed. Neo quickly identified the position of the saved return address, but his attack was subverted by the NX bit protection.

However, Neo used his impressive hacker skills to find a magical string in the program itself and construct an exploit that actually works. His exploit uses the following python string:

`"\x90" * k + "\x40\xfb\xe3\xb7" + "\x90" * 4 + "\x13\x07\x40" + "\n"`,

for a suitable value of `k`.

(a) What should be the value of `k` for the exploit to work?

$k =$ _____

Solution: Answer: `k = 20`.

This question requires being able to read x86 assembler code (not in scope as of spring 2021).

First, note that the input happens at line 5, which uses the insecure `gets` function to write as much input as the attacker provides into the `name` buffer (potentially overflowing the buffer).

Next, draw a stack diagram:

```
[4] rip of exploit
[4] sfp of exploit
[?] compiler padding/saved registers
[8] name
```

The attacker input starts at `name` and writes consecutively upwards. The question does not say to assume there is no compiler padding or saved registers, so we leave it as an unknown for now.

Next, note that the provided exploit contains a single byte `\x90`, repeated `k` times, followed by `\x40\xfb\xe3\xb7` (looks like an address, as we'll confirm later). This should remind you of the classic buffer overflow exploit–we write `k` bytes of garbage to reach the rip, and then we overwrite the rip with the address of some code we want to execute. (See the next part for further discussion on what this address is.)

At this point, the problem has reduced to figuring out how many garbage padding bytes we need at the start of our write in order to overwrite everything up to the rip. Normally we can refer to our stack diagram and be done (8 bytes to overwrite `name`, plus 4 bytes to overwrite the sfp), but because the question doesn't include an assumption about compiler padding or saved registers, we have to check the assembler dump to see if the compiler has added any padding or saved registers.

The relevant assembly instructions start at line 1 of the assembler dump, which is the start of the function prologue of the `exploit` function.

- Line 1: push the sfp on the stack. Standard function prologue instruction.
- Line 2: move ebp to the top of the new stack frame. Sstandard function prologue instruction.
- Lines 3-4: move esp down to the bottom of the new stack frame.
- Line 5: the `lea` command stores the current value in ebp, minus `0x10`, into the eax register. The current value in ebp is the address of the sfp, so eax now contains the address of the sfp, minus `0x10`.
- Line 6 pushes the value in eax on the stack.
- Line 7 calls the `gets` function.

Recall that arguments are pushed on the stack before calling a function. Line 6 is pushing an argument on the stack before line 7 calls the `gets` function. Thus we can deduce that line 6 is pushing the address of `name` on the stack as an argument for the call to `gets`.

Line 6 pushes the value of eax on the stack. At this point, eax contains the address of the sfp, minus `0x10`. Thus we can conclude that the address of `name` is the address of the sfp, minus `0x10`. In other words, `name` is located `0x10 = 16` bytes below the sfp.

We can now revisit our stack diagram and fill in the blank. If there are 16 bytes between `name` and the sfp, and `name` itself is 8 bytes, then we must have 16 - 8 = 8 bytes of compiler padding.

```
[4] rip of exploit
[4] sfp of exploit
[8] compiler padding/saved registers
[8] name
```

Now that we have our stack diagram filled in, we can finally calculate that we need 8 + 8 + 4 = 20 bytes of garbage to overwrite everything between `name` and the rip. Thus `k = 20`.

Here's the original solution text for completeness:

We need `k = 20`. Before executing instruction 0x00400620, $esp points to the saved return address. This instruction decrements $esp by 4. Then, the next instruction saves this value in $ebp. Finally, we see on instruction 0x00400629 that the address of `name` is 16 bytes below this $ebp value. Therefore, it is 20 bytes below the saved return address. For the exploit to work, we want the saved return address to be replaced by 0xb7e3fb40 (`__libc_system`).

(b) What command gets executed as a result of the exploit?

Command (1 line of code): _____

Solution: `system("sh");` – also accept `sh`

From the previous part, we know that we've overwritten the rip with `\x40\xfb\xe3\xb7`. This is the address `0xb7e3fb40` in little-endian.

If you look carefully, this address appears at line 17 in the assembler dump. The assembler dump labels this address as `<__libc_system>`. This is because the assembler dump is labeling the addresses of functions. In other words, `0xb7e3fb40` is the address of the machine instructions for the `system` function. (Using the same logic, you could also conclude that `0xb7e63fb0` is the address of the machine instructions for the `gets` function, and `0xb7e4e940` is the address of the machine instructions for the `printf` function, but these are not relevant to this question.) Thus we can conclude that this exploit will result in a call to the `system` function.

The last part of the exploit is `"\x90" * 4 + "\x13\x07\x40" + "\n"`. This is written above the rip (recall that the rip has been overwritten to contain the address of the `system` function).

Recall that arguments are located at the top of the current stack frame. This means the bytes we write directly above the rip will be interpreted as arguments to the `system` function. (The specifics aren't too important to solve this question. Look into return into libc attacks for more details on how this works.)

`\x90" * 4` is four garbage bytes to make sure that the stack lines up when `system` is called. (Again, the specifics aren't too important to solve this question.)

`\x13\x07\x40\x00` is the address `0x00400713` in little-endian. Note that `gets` adds a null byte at the end of the user's input, which is where the `\x00` comes from.

If you look carefully, this address doesn't appear in the assembler dump, but a nearby address `0x0040070c = 0x40070c` does appear at line 16. This address is being pushed on the stack just before a call to `system`. Using the same logic as the previous part (recalling that arguments are pushed on the stack before we call a function, and that line 17 is a call to `system`), we can deduce that line 16 is pushing the address of the string `./evil.sh` (the argument to `system`) onto the stack.

We know that `0x0040070c` is the address of the string `./evil.sh`. In other words, `0x0040070c` contains the 1-byte character `.` (period), `0x0040070d` contains the 1-byte character `/` (forward slash), `0x0040070e` contains the 1-byte character `e`, and so on. `0x00400713` is 7 bytes after `0x0040070c`, so it's pointing at the character `s`. Treated as a string, `0x00400713` points at the string `sh`.

Finally, we can conclude that the argument passed to the malicious call to `system` is the string `sh`. Thus the final command that gets executed from this exploit is `system("sh")`.

Here's the original solution text for completeness:

The exploits redirects the control flow to address 0xb7e3fb40, which is the beginning of function `__libc_system`, a.k.a. `system`. It also sets up the string argument of function `system` to be 0x00400713 (remember that `gets` replaces the newline `'\n'` with a null byte `'\0'`). The address 0x00400713 is 7 bytes after 0x0040070c, which is the address of string `"./evil.sh"`. Hence, the string given as argument to `system` is `"sh"`, which results in calling `system("sh")` and executing a shell.

(c) List two mitigations that would prevent Neo from pwning the program (without fixing the vulnerable C source code or writing the code in a type-safe language).

Mitigation 1 (5 words max): _____

Mitigation 2 (5 words max): _____

Solution:

Mitigation: stack canaries

Explanation: the stack canary would be overwritten by the buffer overflow, which would be detected by the program before the function returns.

Mitigation: ASLR/Selfrando

Explanation: With ASLR/Selfrando, Neo wouldn't know where in memory libc was loaded. So Neo would not know the address of function `system`.

(d) True or False: The exploit still works if the bytes `'\x90'` in the attack string are replaced by other random values (different from `'\n'`).

● True ○ False

Solution: True, these bytes have no special meaning.

Note that these bytes are not being used as a NOP sled, because they're not actually being executed as instructions. (With the NX bit defense enabled, we couldn't execute anything on the stack anyway.)

Problem 8 ***Network Security*** **(10 points)**

Answer the following questions on network security.

(a) Security incidents are inevitable. What is something you can make sure are recorded *before* an incident occurs that will help you recover afterwards? (Short answer)

> **Solution:** Logs.

(b) You start working as a security expert at GoodCorp, a company with a variety of hardware and software connected to the Internet via a single company network. What tool could you use to detect attacks in real-time on the network? (Short answer)

> **Solution:** NIDS (network intrusion detection system). Logs are not appropriate because they can't detect attacks in real-time. HIDS is not appropriate because the network contains a variety of hardware and software, so it would be too costly to install a different HIDS for each device accessing the Internet. The question also mentions detecting attacks on the network (as opposed to on the end hosts), so NIDS is the best answer.

(c) Unfortunately, GoodCorp's budget is low and you are the only person on the security team despite a connection rate of 10M connections per day. However, you build a tool that generates alerts, which you then manually inspect to confirm an attack. The tool has a detection rate of 99% and a false positive rate of 1%. Will your tool help you prevent attacks in practice?

○ Yes ● No

Explain (be concise):

> **Solution:** You need to manually look at 100,000 (1% of 10 million) false positives per day!

(d) You decide to take a vacation and find yourself inside the Great Firewall of China (a network monitoring system China uses to perform censorship). The Great Firewall attempts to close TCP connections to servers hosting restricted content using injected TCP Reset packets.

Can TLS prevent this censorship?

○ Yes ● No

> **Solution:** No. TLS is built on top of TCP–in other words, encrypted TLS messages are sent over TCP. The Great Firewall (an on-path attacker) can inject a RST packet into the TLS connection to drop your TLS connection, even if they can't read or tamper with the contents of your TLS connection.
>
> Another explanation: TLS is built on top of TCP, so the server and client must first initiate a 3-way TCP handshake before they can perform the TLS handshake. The Great Firewall could send a TCP RST packet during the TCP handshake (before the TLS handshake), so that the TLS connection cannot even begin.

(e) You visit your personal website via HTTPS and notice the CA has changed from the one you configured, but the page loads with no errors! How is this possible? (You are not using a CDN/mirror, and your computer/browser has not been compromised.)

Solution: The page loading with no errors implies that the new certificate authority is valid, and it has signed a certificate for your website. This implies that the attacker has the secret key for another valid CA. Alternatively, the attacker has compromised your personal webserver and requested a new certificate from a different, valid certificate authority (that the attacker has not compromised).

(f) How could you *mitigate* the attack in part (e)?

Solution: HTTP Public Key Pinning (HPKP), or more recently, certificate transparency. or DNS-based Authentication of Named Entities (DANE).

Problem 9 ***Smoke Air Every Day*** **(12 points)**

Lord Carol[10] is hitting refresh on the CAAQMD's[11] website, `http://www.CAAQMD.gov`, hoping the AQI[12] stays under 200. As it approaches 199, she decides to take action.

(a) Lord Carol uses her CS 161 skills and hits `inspect element`. She sees

```
1    <td class="rowData">
2        <div class="cData">199</div>
3        <div class="cAQI red"></div>
4    </td>
```

True or False: Changing 199 to 175 in her browser injects 175 into CAAQMD's database.

○ True ● False

Explain (be concise):

Solution: Changing HTML in your local browser (e.g. using inspect element) only changes your browser's local copy of the HTML file. It doesn't change any state on the server (e.g. its database).

(b) Lord Carol decides to DDoS the CAAQMD servers. As a botmaster, which of the following are normally considered viable ways to implement a botnet?

- ☐ Financial engineering
- ☐ Buy hundreds of smartphones to use as bots
- ■ Pay-per-bot services
- ■ Exploiting Caltopia's web page visitors by adding malicious JavaScript to the web pages
- ■ A worm or virus
- ■ Social engineering
- ☐ None of the Above

Solution: Financial engineering is unrelated to security.

Pay-per-bot services are cheap and therefore a viable way to implement a botnet.

A worm or virus can infect and take over many computers very quickly, giving you a botnet. A worm or virus spreads quickly and far by design.

Social engineering can be used to trick many victims into giving you access to their computing power.

Buying hundreds of smartphones is too expensive to be viable. It's far more expensive and hard to manage than a pay per bot service.

Adding malicious JavaScript can let you execute arbitrary JavaScript in many victim's computers. Once the XSS attack is in place, the infection happens passively (the attacker doesn't need to be online doing anything, but users will still get infected).

(c) Which of the following would automatically stop the DDoS attack if implemented by CAAQMD?

[10] Although traditionally, "Lord" refers to men, recent usage has allowed this in a gender neutral context. For example, one of Queen Elizabeth's many titles is "Lord of Mann".

[11] Caltopia Area Air Quality Management District

[12] Air Quality Index

☐ Firewall ☐ NIDS

☐ KIDS ☐ Log-based Detection System

☐ YIPS ■ None of the Above

Solution: These systems aim to prevent intrusion using various means, or try to spot when an intrusion has happened. They are not designed to prevent an overwhelming amount of traffic from flooding a network.

In other words, a DDoS attack can overwhelm the detection system, which only has limited computation resources to check every incoming request.

(d) Oski decides to (legally) smoke tobacco and marijuana near the CAAQMD's AQI sensor.[13] The AQI reading hits 9001 and Caltopia automatically shuts down. This is a false

● positive. ○ negative.

Solution: The AQI reading classified the air as "unhealthy" (positive) and shut down Caltopia, when it really was just a byproduct of Oski's smoking (false).

[13] PSA: Oski is immortal and can afford to do this. Please don't smoke.

Figure 1: "Also, after all the warnings about filling in the bubbles completely, I spent like 30 seconds on each one." – XKCD

NOT RECOMMENDED

Reboot the syslem from trusted media.

Reboot the system from trusted media.

Reboot the system from trusted media.

RECOMMENDED

Figure 2: Recommended handwriting for CS161 final exam.