Weaver Fall 2017 — CS 161 Computer Security — Final Exam

Midterm 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. I am aware that Nick believes *in retribution.*

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 four 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 (248 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*** **(40 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) Stack canaries reliably block all stack overflow attacks.

○ **True** ● **False**

> **Solution:** False. Stack canaries will not protect local variables from being overwritten through an unprotected buffer. An attacker could also try to guess the canary value or leak the canary value and overwrite the canary with itself.

(b) Nonexecutable stacks reliably block all stack overflow attacks.

○ **True** ● **False**

> **Solution:** False. The executable code could be located somewhere else, such as the heap, and skilled attackers could use return oriented programming to exploit a system in a variety of ways.

(c) Nick's house is Ravenclaw ("Where those of wit and learning, will always find their kind").

○ **True** ● **False**

> **Solution:** False. Just for fun/attendance question.
>
> "Have you SEEN the rug in my office? Slytherin forever!" -Nick

(d) Forward secrecy means that, if your private key is compromised, an attacker can't recover the plaintext for previous messages when they have also captured the ciphertext.

● **True** ○ **False**

> **Solution:** True. This is the definition of forward secrecy.

(e) If you use a U2F security key as your second factor in a 2-factor supporting site, this generally prevents phishing attacks.

● **True** ○ **False**

> **Solution:** True. Even if you are tricked into visiting a phishing website and type in your password on the malicious website, U2F will not send a valid

authentication token to the malicious website, so the attacker won't have both factors needed to execute the phishing attack and log in as you.

(f) If a site uses DHE for TLS, an adversary who steals the site's private key can passively decrypt intercepted communications.

○ **True** ● **False**

Solution: False. The site private key in DHE-TLS is only used to sign the server's Diffie Hellman message. A passive attacker who learns the private key does not learn any new information about the Diffie-Hellman exchange–the DH messages are sent in the clear. The attacker, who does not have the site's DH secret a, is unable to compute the premaster secret g^{ab}.

However, an active attacker/MITM would be able to use the private key to their advantage by performing a MITM attack on the Diffie Hellman key exchange.

(g) The Chinese "Great Firewall" operates using the same basic mechanism as a corporate firewall.

○ **True** ● **False**

Solution: False. The Great Firewall is an on-path device that looks at requests and injects replies, while corporate firewalls are in-path (man-in-the-middle) devices.

(h) If you combine two independent detectors in a way which reduces the false-positive rate this combination will increase the false-negative rate.

● **True** ○ **False**

Solution: True. Intuitively, if the false positive (alerting when there's no attack) rate is lower, this means we are alerting less often, which leads to more false negatives (failing to alert when there's actually an attack).

(i) Using sprintf to format user input to an SQL query is safe if you just replace all ' characters with \' in the user input.

○ **True** ● **False**

Solution: False. We can escape the escaper. For example, the input `robert\';drop table students` is changed to `robert\\';drop table students`. The backslash is escaped and the apostro-

phe is still treated as SQL syntax.

(j) Most of the block cipher modes you learned about require keeping the IV secret

○ **True** ● **False**

Solution: False. In the block cipher modes we saw in this class (e.g. AES-CBC, AES-CTR), the IV is public and sent as part of the ciphertext.

(k) When configuring a firewall, it's safer to use an approach based on blacklisting hosts than whitelisting hosts

○ **True** ● **False**

Solution: False. Blacklisting (default-allow) is less safe than whitelisting (default-deny), because unspecified or unknown inputs will be allowed by default.

(l) HTTPS can prevent CSRF attacks

○ **True** ● **False**

Solution: False. HTTPS provides an end to end secure channel for communication, so it defends against network attackers. CSRF attacks are web attacks that trick a victim into performing an unintended action by making them visit a link. The link could be an HTTPS link, but the important thing in CSRF is the side-effect associated with visiting a link rather breaking confidentiality and integrity in the network connection. The two are relatively unrelated.

(m) A secure hash function will only produce collisions with an infinitesimally small probability.

● **True** ○ **False**

Solution: True. A secure hash function is collision-resistant, which means the probability of a collision (two inputs with the same output) is infinitesimally small.

(n) AES-CTR mode provides integrity when properly used

○ **True** ● **False**

Solution: False. AES-CTR only provides confidentiality.

(o) AES-GCM mode provides integrity when properly used

● **True** ○ **False**

Solution: True. AES-GCM provides both confidentiality and integrity when properly used.

(p) PBKDF2 turns a password into a large amount of key material by repeatedly encrypting the password with AES.

○ **True** ● **False**

Solution: False. PBKDF2 repeatedly applies HMAC on the password, not AES.

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

● **True** ○ **False**

Solution: True. Frames have the origin of the website in the frame, so by the same-origin policy, the inner frame cannot access the contents of the outer frame.

(r) Consider a scanning worm that picks addresses uniformly at random. IPv6 changes IP addresses from 32b to 128b. Selecting an IPv6 address uniformly at random is not an effective strategy for a worm.

● **True** ○ **False**

Solution: IPv6 has 2^{128} possible addresses, and IPv4 has 2^{32} addresses, so a much larger proportion of IPv4 addresses are valid compared to IPv6 addresses. In fact, the vast majority of IPv6 addresses are unused. A worm that randomly picks an IPv6 address will pick many invalid or unused IPv6 addresses, so random scanning is not an effective strategy.

Problem 2 ***Multiple Choice*** **(29 points)**

For multiple choice questions, select **all** which are correct.

(a) (1 point) (This question is *magic*, the amount of points may vary) The Magic Words are...

- ○ Livid Peregrine
- ● Squeamish Ossifrage
- ○ Senatorial Chickenhawk
- ○ Irate Vulture
- ○ None of the Above

Solution: Just for fun/attendance question. No relevant course content.

(b) (4 points) Valid analogies between the Influenza virus and computer viruses include:

- ● Both are often detected based on "known bad" features on the virus
- ● Both may have designs which can mutate to evade detection
- ● Both spread faster when the percentage of vulnerable hosts is greater
- ○ Neither have caused substantial loss of life
- ○ None of the Above

Solution: Both are often detected based on known features: recall signature-based detection.

Both can mutate to evade detection: recall computer virus mutations.

Both spread faster when the percentage of vulnerable hosts is greater: when there are more machines/people to infect, the virus can spread faster.

The influenza virus has caused substantial loss of life (1918 pandemic).

(c) (4 points) DNSSEC, when validated only by the client, provides which of the following security properties for DNS responses? **Mark ALL that apply.**

- ○ Confidentiality
- ● Integrity
- ● Authentication
- ○ Availability
- ○ None of the Above

Solution: DNSSEC does not provide confidentiality because records are not encrypted. DNSSEC does provide integrity and authentication because all records are signed. Availability is not protected by DNSSEC–a MITM attacker could drop DNSSEC traffic or DoS DNS servers or clients.

(d) (4 points) DNSSEC, when validated only by the recursive resolver, provides which of the following security properties for DNS responses? **Mark ALL that apply.**

- ○ Confidentiality
- ○ Integrity
- ○ Authentication
- ○ Availability
- ● None of the Above

Solution: Recall that in DNSSEC, the client makes a request to the recursive resolver. Then the recursive resolver queries all the relevant name servers. Finally, the recursive resolver passes the final answer to the client.

Since the client is not validating DNSSEC, the recursive resolver could lie to the client.

(e) (4 points) Which of the following are IND-CPA when properly used?

- ○ ROT-13
- ● CBC
- ○ ECB
- ● CTR
- ● One Time Pad
- ● GCM
- ○ None of the Above

Solution: ROT-13 (substitute A=N, B=O, C=P, etc.) is not IND-CPA secure. Anybody can perform the substitution to encrypt/decrypt messages.

CBC, CTR, one-time pads, and GCM are IND-CPA when properly used, as seen in lecture.

ECB is deterministic, so it is not IND-CPA secure.

(f) (4 points) Which of the following are IND-CPA when the IV is mistakenly resused?

- ○ CBC
- ○ ECB

- ○ CTR
- ● None of the Above
- ○ GCM

Solution: CBC, CTR, and GCM are not IND-CPA secure with IV reuse, because they become deterministic schemes.

ECB is not IND-CPA secure regardless of IV usage. (In fact, it doesn't use an IV at all.)

(g) (4 points) Which of the following can defend against many heap overflow attacks

- ○ Stack Canaries
- ● Memory safe languages
- ● ASLR
- ○ C++17
- ○ XSS
- ○ None of the Above
- ● Non executable heaps

Solution: Stack canaries only defend against attacks on the stack. Heap overflow attacks wouldn't overwrite the stack canary.

ASLR can defend against heap overflow attacks if the attack requires the attacker to input an address. ASLR would randomize the addresses on the heap and force the attacker to guess or leak memory addresses.

XSS is a web attack and unrelated to heap overflows.

Non-executable heaps can defend against heap overflows if the attack requires the attacker to write and execute some code in the heap. Non-executable heaps would prevent the attacker from executing any machine instructions they write to the heap.

Memory safe languages defend against all buffer overflow attacks, including heap overflows.

C++17 isn't a memory safe language, so it doesn't defend against heap overflows.

(h) (4 points) Which of the following attacks might allow an attacker to steal one of your browser's secure (HTTPS-only) cookies (**Mark ALL that apply**):

- ● Stored XSS
- ● Reflected XSS
- ○ Clickjacking
- ● Buffer overflow in your browser

○ Packet injection without exploitation

● Packet injection with a browser exploit

○ None of the Above

Solution: Stored XSS and reflected XSS allow the attacker to execute arbitrary JavaScript in your browser. The attacker can use JavaScript to steal your secure cookies (assuming the HTTPonly flag isn't set to prevent JavaScript from accessing the cookies).

Clickjacking allows an attacker to force you to visit a link of the attacker's choice (e.g. the attacker's website), but JavaScript on an attacker's website can't access secure cookies for different domains.

Secure cookies are only sent over HTTPS (TLS). TLS is end-to-end secure, so packet injection alone cannot be used to steal secure cookies.

A browser exploit might allow the attacker to force your browser to use an outdated or insecure version of TLS. An insecure version of TLS combined with packet injection would allow the attacker to steal secure cookies. The attacker could also use the browser exploit to directly steal your cookies.

A buffer overflow in your browser would allow the attacker to run arbitrary code in your browser. The attacker could use this to steal your cookies.

Problem 3 ***Moogle*** **(28 points)**

For the following questions, select **all** answers which apply.

(a) (4 points) Moogle is a search engine that claims to better than Google. Whenever a user searches for something, a line of text appears that says "You searched for: " followed by the user's query unescaped. Which of the following is this vulnerable to?

- ○ Command Injection
- ○ Buffer Overflow
- ○ Format String
- ○ CSRF
- ● XSS
- ○ SQL Injection
- ○ Clickjacking
- ○ None of the Above

Solution: This is a classic case of reflected XSS. If someone clicks on a link that searches for Javascript code in between script tags, since it is unescaped, it will be executed by a victim browser.

Some have argued that SQL injection is a possible answer, but it's not the best answer (and the intention was to select one). See this explanation from David Wagner:

I don't think I'd accept SQL injection as an answer to 3a. There's nothing in the question that suggests that the site is vulnerable to SQL injection, but there's information that directly implies it is vulnerable to XSS. If I were writing the grading rubric, I wouldn't give points for SQL injection. XSS is about untrusted inputs included in HTML; SQL injection is about untrusted inputs included in SQL queries; and the question only gives you information about what was included in the HTML.

The server could be vulnerable to SQL injection, but it could be vulnerable to anything, so by that logic, one would need to accept all answers. In this case, XSS is the best answer.

(b) (4 points) Moogle uses the Maboody Rank Algorithm to order web pages. A part of this algorithm logs queries into a file. The code is shown below:

```
char* search(char* query) {
    char logging[100];
    sprintf (logging, "echo %s > log.txt", query);
    system(logging);
    return logging;
}
```

Which of the following is this vulnerable to?

- ● Command Injection
- ● Buffer Overflow
- ○ Format String
- ○ CSRF
- ○ XSS
- ○ SQL Injection
- ○ Clickjacking
- ○ None of the Above

Solution: Command Injection comes from the system() call. Buffer overflow comes from the fact that the query string could be arbitrarily long since the bounds of the buffer are not checked.

There is no format string vulnerability because attacker input is not provided to the vulnerable `sprintf` argument (the second argument, where the percent formatters are located).

CSRF, XSS, SQL injection, and clickjacking are web attacks, which are not relevant to this C code snippet.

(c) (4 points) Moogle decides to enable ASLR, NX, and stack canaries. Which of the following does this completely defend against?

- ○ Command Injection
- ○ Buffer Overflow
- ○ Format String
- ○ CSRF
- ○ XSS
- ○ SQL Injection
- ○ Clickjacking
- ● None of the Above

Solution: While these defenses raise the bar for attack, they do not completely defend against memory safety exploits (command injection, buffer overflows, and format string exploits). For example, return oriented programming based attacks will bypass these defenses.

CSRF, XSS, SQL injection, and clickjacking are web attacks, which are not relevant to these memory safety defenses.

(d) (4 points) Moogle decides to use only Java as a programming language, they don't use the foreign function interface, and their Java compiler and runtime is bug free. Which of the following vulnerability does this completely defend against?

- ○ Command Injection
- ● Buffer Overflow
- ● Format String
- ○ CSRF

- ○ XSS
- ○ Clickjacking
- ○ SQL Injection
- ○ None of the Above

Solution: Java is a memory safe language, which protects against memory safety exploits (buffer overflows and format string vulnerabilities).

Command injection is still possible in memory safe languages. You can pass user input into a system call in Java, for example.

CSRF, XSS, SQL injection, and clickjacking are web attacks, which are not relevant to using a memory safe language.

Note: foreign function interface means that Java isn't calling functions in other languages (e.g. memory-unsafe languages like C).

(e) (4 points) Moogle decides to prevent other websites from including iframes to their page. Which of the following vulnerability does this completely defend against?

- ○ Command Injection
- ○ XSS
- ○ Buffer Overflow
- ○ SQL Injection
- ○ Format String
- ● Clickjacking
- ○ CSRF
- ○ None of the Above

Solution: Command injection, butter overflow, and format string vulnerabilities are memory safety attacks, which are not relevant to iframes (a web feature).

Among the other answer choices, clickjacking is the only attack that commonly takes advantage of iframes. For example, an attacker can put a malicious download button in an iframe on a legitimate website to trick the user into clicking the malicious button.

(f) (4 points) Moogle decides to define a content security policy. What vulnerability does this partially defend against?

- ○ Command Injection
- ● XSS
- ○ Buffer Overflow
- ○ SQL Injection
- ○ Format String
- ○ Clickjacking
- ○ CSRF
- ○ None of the Above

Solution: Content Security Policy is used to mitigate XSS attacks.

See https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP for more details.

(g) (4 points) Moogle decides to use hidden tokens inside their cookies to be sent along with their API requests. What vulnerability does this partially defend against?

- ○ Command Injection
- ○ Buffer Overflow
- ○ Format String
- ○ CSRF
- ○ XSS
- ○ SQL Injection
- ○ Clickjacking
- ● None of the Above

Solution: Command injection, butter overflow, and format string vulnerabilities are memory safety attacks, which are not relevant to hidden tokens in API requests (a web feature).

Hidden tokens inside don't defend against any web attacks. CSRF tokens defend against CSRF attacks, but they are not hidden in cookies (they're sent as a hidden form field).

Problem 4 ***Banana Messenger*** **(30 points)**

The computer-firm **Banana Inc** has developed a messaging system called *bMessage* running on the amazingly secure BananaPhone. bMessage is designed to provide end-to-end protection between users so that Banana Inc can't read the messages.

The initial version of bMessage has a central keyserver, similar to the one in the project but slightly different. Since a user can have multiple devices, each with their own 2048b random RSA public key, when **Bob** queries for **Alice**'s public key, Bob doesn't just get one key but gets a list of keys, all of which belong to Alice. Bob also performs a query for his own keys as well (so that his other devices can see what he sent).

Bob then encrypts the message using AES256-CFB-HMAC-SHA256 with a random key, encrypts the random key with each one of Alice's and Bob's public keys using RSA-OAEP, and forwards the message to Banana Inc to deliver. When queried, the keyserver not only knows what the query is for but also who is making the query.

The Federal Bulls**t Investigators (FBSI) are investigating Bob for Crimes against Humanity, namely playing Cards Against Humanity in the middle of Moffit Library while CS161 students were trying to study for their final. In their investigation, they want to wiretap Bob's communications.

(a) (8 points) The FBSI wants Banana Inc to provide the FBSI with a copy of all future communications sent by Bob by only modifying Banana Inc's server. Banana Inc complains that they can't, because to do so would require modifying Bob's bMessage client, and Bob does not believe in applying software updates. Who is correct? **Mark ONE of the following** and **BRIEFLY explain** (it should fit in a tweet) your answer.

○ Banana Inc ● FBSI

Tweet Length Explanation:

> **Solution:** You modify the keyserver so that when Bob looks up a key, it also returns a key for the FBSI.
>
> When Bob wants to send a message to Alice, the keyserver returns a list of public keys, one for each of Alice's devices. The modification is to add another public key for the FBSI, which Bob will think is one of Alice's devices. Then Bob will encrypt his message with a random key, and encrypt that key with all the public keys (including FBSI's public key).

(b) (6 points) The FBSI has the ability to ask Banana Inc for a copy of all of Bob's ENCRYPTED bMessages, but without the key they can't decrypt them. The FBSI, tired of Banana's intransigence, instead decides to get access to Bob's key another way, by hacking. (They are the Law, so it's "legal").

Banana recently released a web-browser based client which allows Bob to access his bMessages from any web browser, bMessage-Web, a feature Bob decides he likes to use. When it is first run, the bMessage web-client's JavaScript creates a random 2048b RSA key and a password (which is *different* from the password used to log onto the bMessage web site) from Bob. It then creates a key using PBKDF2-SHA256, using this key with AES256-CFB-HMAC-SHA256 to encrypt the private RSA key for storing on Banana Inc's web server. This random key becomes another key for Bob.

Now on future logins, the JavaScript gets the encrypted RSA private key from the server (where it is stored in a SQL database), decrypts it using Bob's password, and then uses it to decrypt Bob's bMessages. And Bob is, well, a typical user and his passwords in general have 50b of entropy or less.

Assume there exists an XSS vulnerability. Provide a tweet-length strategy where the FBSI can use a XSS vulnerability in bMessagesWeb to probably access bob's messages.

Solution: You inject JavaScript onto the page to send Bob's private key to the FBSI.

The web client will probably display Bob's messages to Bob in plaintext, so the injected JavaScript can read the plaintext messages on the site and send them to the FBSI.

(c) (6 points) Assume there exists an SQLI vulnerability. Provide a tweet-length strategy where the FBSI can use a SQLI vulnerability in bMessagesWeb to probably access Bob's messages.

Solution: Use SQL injection to get the encrypted private RSA key. Use a brute-force attack to guess Bob's password and decrypt the RSA key. Use the key to decrypt Bob's messages.

Bob's password has less than 50 bits of entropy, so it can be brute-forced. The private RSA key is encrypted with Bob's password (passed through PBKDF2-SHA256), so the FBSI just has to guess Bob's password to obtain the private RSA key and decrypt Bob's messages.

(d) (6 points) Assume that the FBSI has obtained Bob's website password but not his bMessage password. Provide a tweet-length strategy where the FBSI can use Bob's website password (NOT his bMessage password) to get his messages.

Solution: You get the encrypted blob and do a brute force attack on the password.

Use Bob's website password to login, causing the JavaScript to fetch the encrypted private RSA key from the server. Brute-force the password to decrypt the private RSA key (as in the previous part) and use the private RSA key to decrypt Bob's messages (as in the previous part).

(e) (4 points) Banana Inc is sick of the FBSI hacking, so they add a network-based Intrusion Prevention System that is capable of monitoring all requests to the bMessage web server (since the web server tells the NIDS about the keys for all TLS connections) and blocking all requests that don't satisfy the criteria. Which of the previous FBSI attacks could this host-based system potentially stop? Select all which apply:

- ● The XSS attack
- ● The SQLI attack
- ○ The FBSI has Bob's web password
- ○ None of the above

Solution: A NIDS capable of reading plaintext messages could look for characters that are usually associated with XSS and SQL injection attacks. For example, XSS attacks usually contain script tags, and SQLI attacks commonly contain SQL syntax such as quotes, semicolons, or dashes.

A NIDS would not be able to stop the FBSI attack where Bob's web password is compromised, because the NIDS cannot distinguish between the FBSI trying to log in as Bob and a legitimate login request from Bob.

Problem 5 ***A Tour of Tor*** **(24 points)**

As a reminder, when connecting to a normal website through Tor, your computer first queries the Tor "consensus" to get a list of all Tor nodes, and using this information it connects to the first Tor node and, from there, creates a circuit through the Tor network, eventually ending at an exit node.

(a) (4 points) Consider the scenario where you are in a censored country and the censor chooses not to block Tor, the censor is the adversary, and no Tor relays exist within this country. How many Tor relays must your traffic pass through, including the exit node, to prevent the censor from blocking your traffic.

- ● One
- ○ Two
- ○ Three
- ○ Four
- ○ Tor doesn't stop this adversary

Solution: The censor doesn't block Tor and the relay is outside of the country, so one hop will get you safely past the censor. The censor will see you sending packets to an encrypted Tor relay but will not be able to determine who you're actually communicating with.

This is equivalent to using a VPN where the VPN server is in a different country.

(b) (4 points) Consider the scenario where you are the only user of Tor on a network that keeps detailed logs of all IPs contacted. You use Tor to email a threat. The network operator is made aware of this threat and that it was sent through Tor and probably originated on the operator's network. How many Tor relays must your traffic pass through, including the exit node, to guarantee the network operator can't identify you as the one who sent the threat?

- ○ One
- ○ Two
- ○ Three
- ○ Four
- ● Tor doesn't stop this adversary

Solution: Since you are the only user of Tor, the network operator just needs to look at the IP of the only person trying to connect to a Tor relay. The network operator can look through the list of IPs and see that you contacted a Tor relay regardless of how many relays you use.

(c) (4 points) Consider the scenario where there is a single hostile Tor node but you don't know that node's identity, and that node can be an exit node. You want to

keep confidential from this node what HTTP sites you are visiting through Tor. How many Tor relays must your traffic pass through, including the exit node, to guarentee this adversary can't know what sites you visit?

- ○ One
- ○ Four
- ● Two
- ○ Tor doesn't stop this adversary
- ○ Three

Solution: If you only use a single relay, then if that relay is hostile they will be able to see your request and the site you're visiting. If you use two relays, the first relay cannot see your request, and the second can see your request but doesn't know who it's from. So in either case, you are protected.

In other words, if the second relay is positioned between you and the hostile node, the hostile node will not know the request originated from you since it only sees the incoming request coming from "that other node." If the second relay is positioned between the hostile node and your destination, then while the hostile node knows the request comes from you, it doesn't know the destination since it forwards the request to "that other node."

(d) (4 points) Consider the scenario where there are mulitple independent hostile Tor nodes but you don't know that nodes identitities, and these nodes can be a exit nodes. You want to keep confidential from all these nodes what HTTP sites you are visiting through Tor. How many Tor relays must your traffic pass through, including the exit node, to guarentee that every independent hostile node can't know what sites you visit?

- ○ One
- ○ Four
- ● Two
- ○ Tor doesn't stop this adversary
- ○ Three

Solution: The solution is the same as the previous question. Since the hostile nodes are independent (non-colluding), it doesn't matter that there are multiple. No individual node can ever both your identity and the request as long as you use at least two relays.

(e) (4 points) Consider the scenario where there are mulitple colluding hostile Tor nodes but you don't know those nodes identitities, and these nodes can be a exit nodes. You want to keep confidential from all these nodes what HTTP sites you

are visiting through Tor. How many Tor relays must your traffic pass through, including the exit node, to guarentee that the colluding system of hostile nodes can't know what sites you visit?

- ○ One
- ○ Two
- ○ Three
- ○ Four
- ● Tor doesn't stop this adversary

Solution: Now, since the hostile nodes are colluding, you cannot ever be sure you are anonymous since you could get "unlucky" and have every node in your path be colluding hostile nodes.

Note that in real life, using three relays makes the probability of this happening negligible (assuming a certain amount of randomness in relay selection).

(f) (4 points) Consider the scenario where there is a single hostile Tor node but you don't know that node's identitity, and that node can be an exit node. You want to have data integrity for the HTTP sites you are visiting through Tor. How many Tor relays must your traffic pass through, including the exit node, to guarentee this adversary can't manipulate the data you receive from the sites you visit?

- ○ One
- ○ Two
- ○ Three
- ○ Four
- ● Tor doesn't stop this adversary

Solution: The exit node could modify the HTTP response without detection before forwarding the HTTP response to you.

Problem 6 ***Distributed Web Engineering*** **(24 points)**

In the old days, web sites used to run on just a single computer. But now, a modern web "site" can be spread over multiple computers and even multiple domains. So a user's login "cookie" needs to work transparently. All the servers which make up the site share a common 256b secret S and a common secret key K_s. The login server has a public key K_l.

Consider some of the following schemes. In these, the adversary has multiple accounts to experiment with and their goal is to be able to create a fake cookie for a targeted account they don't control.

(a) (8 points) In scheme 1, when you login, your browser presents your password P to the login server. The login server sets your *login* cookie as $Name||expires$ and sets an additional *auth* cookie as $Sign(K_l, login)$, using 512b RSA.

Can this prevent an attacker from making fake cookies?

Mark ONE of the following and **BRIEFLY explain** (tweet-length) your answer.

○ Yes ● No

Tweet Length Explanation:

Solution: No. The RSA key length is too small.

An attacker could brute-force a valid signature and create a fake auth cookie.

(b) (8 points) In scheme 2, when you login, your browser presents your password P to the login server. The login server sets your *login* cookie as $Name||expires$ and then has JavaScript in the browser create the *auth* cookie as $HMAC(K_s, login)$.

Can this prevent an attacker from making fake cookies?

Mark ONE of the following and **BRIEFLY explain** (tweet-length) your answer.

○ Yes ● No

Tweet Length Explanation:

Solution: No. The attacker can get K_s from the JavaScript.

Recall that JavaScript runs in the user's browser, so if JavaScript is calculating the HMAC with K_s, the JavaScript in the user's browser must know the value of K_s. The attacker can create a dummy account and see the value of K_s in their browser and then use it to create a fake auth cookie.

(c) (8 points) In scheme 3, when you login, your browser presents your password P to the login server. The login server sets your *login* cookie as $Name||expires$ and then sets the *auth* cookie as $SHA256(S||login)$

Can this prevent an attacker from making fake cookies?

Mark ONE of the following and **BRIEFLY explain** (tweet length) your answer.

● Yes ○ No

Tweet Length Explanation:

Solution: Yes. Without knowing S, the attacker can't create a valid hash.

Note that unlike the last question, where the HMAC is calculated in the browser, the SHA256 calculation is being computed by the server. The attacker has access to internal memory and variables in the browser-side Javascript, but not the server-side code, so in this question, the attacker cannot access S.

Note that the attacker might be able to exploit a length-extension attack, but this would only let the attacker create $SHA(S||Name||expires||other)$, so this attack would only succeed if the targeted username happens to be $Name||expires||other$ (where *other* is an attacker-chosen value).

Problem 7 ***DNSSEC*** **(18 points)**

The UC Berkeley administrators recently announced a change in policy. When you create a new `berkeley.edu` subdomain, DNSSEC is now disabled by default, even thought `berkeley.edu` does support DNSSEC and uses NSEC for proving domains don't exist.

The CS161 staff just created a subdomain `insecurity.berkeley.edu` and populated domains within it.

(a) (4 points) If a recursive resolver with an empty cache which does NOT support DNSSEC wants to look up `tragic.insecurity.berkeley.edu`, which of the following queries will it make (select all that apply).

- ● `tragic.insecurity.berkeley.edu` from a root
- ● `tragic.insecurity.berkeley.edu` from a nameserver for `.edu`
- ● `tragic.insecurity.berkeley.edu` from a nameserver for `berkeley.edu`
- ● `tragic.insecurity.berkeley.edu` from a nameserver for `insecurity.berkeley.edu`
- ○ DNSKEY for `.` from a root
- ○ DS for `.` from a root
- ○ DNSKEY for `edu` from a root
- ○ DS for `edu` from a root
- ○ DNSKEY for `edu` from a nameserver for `.edu`
- ○ DS for `edu` from a nameserver for `.edu`
- ○ DNSKEY for `berkeley.edu` from a nameserver for `.edu`
- ○ DS for `berkeley.eduedu` from a nameserver for `.edu`
- ○ DNSKEY for `berkeley.edu` from a nameserver for `berkeley.edu`
- ○ DS for `berkeley.eduedu` from a nameserver for `berkeley.edu`
- ○ DNSKEY for `insecurity.berkeley.edu` from a nameserver for `berkeley.edu`
- ○ DS for `insecurity.berkeley.edu` from a nameserver for `berkeley.edu`

Solution: The recursive resolver first goes to the root name server with its request (tragic.insecurity.berkeley.edu).

The root name server will tell the recursive resolver information about the .edu nameserver.

The recursive resolver then goes to the .edu name server with the same request (tragic.insecurity.berkeley.edu).

The .edu name server will tell the recursive resolver information about the .berkeley.edu nameserver.

The recursive resolver then goes to the berkeley.edu name server with the same request (tragic.insecurity.berkeley.edu).

The berkeley.edu name server will tell the recursive resolver information about the insecurity.berkeley.edu nameserver.

The recursive resolver then goes to the insecurity.berkeley.edu name server with the same request (tragic.insecurity.berkeley.edu).

The insecurity.berkeley.edu name server will have the IP of the requested domain.

Note that no queries for DNSKEY or DS records are ever made because the recursive resolver does not support DNSSEC.

(b) (4 points) If a recursive resolver with an empty cache which does support DNSSEC wants to look up `tragic.insecurity.berkeley.edu`, which of the following queries will it make (select all that apply).

- ● `tragic.insecurity.berkeley.edu` from a root
- ● `tragic.insecurity.berkeley.edu` from a nameserver for `.edu`
- ● `tragic.insecurity.berkeley.edu` from a nameserver for `berkeley.edu`
- ● `tragic.insecurity.berkeley.edu` from a nameserver for `insecurity.berkeley.edu.edu`
- ● DNSKEY for `.` from a root
- ○ DS for `.` from a root
- ○ DNSKEY for `edu` from a root
- ● DS for `edu` from a root
- ● DNSKEY for `edu` from a nameserver for `.edu`
- ○ DS for `edu` from a nameserver for `.edu`
- ○ DNSKEY for `berkeley.edu` from a nameserver for `.edu`
- ● DS for `berkeley.edu` from a nameserver for `.edu`
- ● DNSKEY for `berkeley.edu` from a nameserver for `berkeley.edu`
- ○ DS for `berkeley.edu` from a nameserver for `berkeley.edu`
- ○ DNSKEY for `insecurity.berkeley.edu` from a nameserver for `berkeley.edu`
- ● DS for `insecurity.berkeley.eduedu` from a nameserver for `berkeley.edu`

Solution: The recursive resolver will make the same queries as in the previous part.

Additionally, the recursive resolver will need DNSKEY records from each name server it contacts (root, `.edu`, `berkeley.edu`, `insecurity.berkeley.edu`) containing the name server's own public key. (The DNSKEY for `insecurity.berkeley.edu` from a name server for `insecurity.berkeley.edu` is not listed as an option in the question, because the `insecurity.berkeley.edu` subdomain does not support DNSSEC.)

Additionally, the recursive resolver needs DS records for each name server it contacts, except for the root, containing an endorsement of the next (child) name server. Namely, root provides a DS record for `edu`. `edu` provides a DS record for `berkeley.edu`. `berkeley.edu` provides a DS record for `insecurity.berkeley.edu`.

(c) (4 points) If a validating recursive resolver looks up `horrible.insecurity.berkeley.edu` it gets an NXDOMAIN (this name doesn't exist) response. What crypotgraphic assertions can the resolver make about this NXDOMAIN error.

- ● `insecurity.berkeley.edu` does not support DNSSEC
- ○ `horrible.insecurity.berkeley.edu` does not exist
- ● Nobody has tampered with the repies from the `berkeley.edu` nameserver
- ○ Nobody has tampered with the repies from the `insecurity.berkeley.edu` nameserver

Solution: NXDOMAIN records are only used in regular DNS (not DNSSEC). DNSSEC uses NSEC records instead. Thus we can conclude that `insecurity.berkeley.edu` (the name server responsible for queries about `horrible.insecurity.berkeley.edu`) does not support DNSSEC.

The `berkeley.edu` name server uses DNSSEC, so we can be sure that nobody has tampered with replies from the `berkeley.edu` name server. All the records in the replies will be signed with a chain of trust to the root.

The `insecurity.berkeley.edu` name server does not use DNSSEC, so we cannot be sure that nobody has tampered with the replies from the `insecurity.berkeley.edu` name server. As a result, it's possible that `horrible.insecurity.berkeley.edu` does exist, but an attacker has replaced the A record (containing its IP address) with an NXDOMAIN record.

(d) (6 points) If all Berkeley services are mandated to use only end-to-end encrypted protocols, does the removal of DNSSEC have a practical impact on security? **Mark ONE of the following** and **BRIEFLY explain** ($\leq$ 2 sentences) your answer.

○ Yes ● No

Tweet Length Explanation:

Solution: If all Berkeley services use end-to-end encrypted protocols (e.g. TLS), then even if we remove DNSSEC and an attacker provides us with an incorrect IP address for a domain, we will find out that we're talking to the wrong server during the TLS handshake. (Recall that in the TLS handshake, the server authenticates itself by sending its certificate and then proving ownership of the private key corresponding to the public key in the certificate.)

Problem 8 ***Securing the Vault*** **(40 points)**

BearBank stores all of its sensitive company information on a set of "air-gapped" machines (i.e., no Internet connection). These machines are locked inside of a large vault with one door that contains a badge scanner (S). Employees at BearBank carry authentication badges that support all of the cryptographic primitives discussed in class.

After taking CS 161, Frodo is hired by BearBank to design a secure protocol for checking whether an employee is authorized to enter the vault. In particular, BearBank would like Frodo's system to protect against "skimming" attacks. In a skimming attack, an attacker (Mallory) knows all of the details about Frodo's authentication protocol, except secret key values; Mallory will interact with a victim on one day and then conduct an attack **the next day**. Specifically, on day #D, Mallory tricks an authorized user (Bob) into scanning his badge (P) on her malicious scanning device. This malicious device and P engage in Frodo's authentication protocol multiple times, and all of P's responses are recorded. The malicious device can spoof messages that look like what the real scanner (S) would send as long as the messages don't require a secret key to generate. Finally, on **the next day** (Day $\#D+1$), Mallory tries to gain access to the vault after analyzing the responses recorded from P.

Unfortunately, Frodo skipped CS 161 discussion sections, so he's not sure whether any of the following three designs are truly secure. For the three subparts below, select whether the protocol is secure or insecure. If the protocol prevents skimming attacks while still allowing authorized users to access the vault, then the protocol is secure. A protocol is insecure if either authorized users cannot access the vault or if Mallory can gain access to the vault. Justify your answer in 1-2 sentences: If the protocol is secure, explain what specific cryptographic primitive(s) guarantee its security and why. If the protocol is insecure, briefly describe an attack, or why legitimate users wouldn't be able to access the vault.

(a) (8 points) Authentication Protocol:

1. For each authorized user, P contains an RSA key pair pk, sk, where pk is the public key and sk is the private key. The vault scanner S stores a copy of each authorized user's pk and has access to an accurate clock.
2. When P is scanned by S, it sends U (the user's name) to S.
3. S then generates and stores $N =$ a new, random 128-bit string for the user and sends N to P. Since a legitimate badge will complete this entire protocol in under 1 minute, if the user doesn't attempt to authenticate within 2 minutes, S deletes N for that user and will randomly generate a new N during the user's next access attempt.
4. P signs N using its private key: $X = N_{sk}$, and sends (X, U) to S.
5. S checks that X is a legitimate signature for U on the N that it randomly generated for U. If the signature is valid, S allows the user to enter the vault and deletes N for the user. Otherwise, access is denied.

● Secure ○ Insecure

Solution: Secure. An attacker could try to get the victim to generate a signature on an arbitrary N' provided by the attacker, but this N' will not match any N generated by the legitimate scanner (with high probability). An attacker can't generate a valid signature on an N provided by the legitimate scanner in the future because RSA signatures are unforgeable without knowledge of the private key. N is randomly generated and deleted after each authorization attempt, and each N can only be used for one entry, so replay attacks are not possible.

(b) (8 points) Authentication Protocol:

1. For each authorized user, P contains a unique 128-bit symmetric key k, and S stores a copy of k for each user and has access to an accurate clock.
2. When P is scanned by S, it sends U (the user's name) to S.
3. S gets the current time from its clock: T, rounded to the neares 30 seconds, and sends T to P.
4. P computes an HMAC of T with k: $X = HMAC(T, k)$, and sends (X, T, U) to S.
5. S checks that X is a valid HMAC on T for U. Additionally, Since a legitimate badge will complete this entire protocol in under 1 minute, S also checks that T is within the past two minutes of the current time on its clock. If both these checks pass, then BearBank allows the user to access to vault. Otherwise, access is denied.

○ Secure ● Insecure

Solution: Insecure. The malicious scanner can just ask P (the victim) to compute HMACs on a bunch of *future* timestamps (by sending future timestamps to the victim). Then on the next day, the attacker makes an authorization request at the same timestamps so that they can present the HMACs obtained from the victim.

(c) (8 points) Authentication Protocol:

1. For each authorized user, P contains a unique random 128b secret key K_u and a highly accurate clock. S stores a copy of each user's k
2. When P is scanned by S, it sends U (the user's name) and $HMAC(S, time)$, with the current time rounded to the nearest 30 seconds to S.

3. S then checks if $HMAC(K_u, time)$ for both the current time (rounded to 30 seconds), and the rounded time +/- 30 seconds.

● Secure ○ Insecure

Solution: Secure. The attack from the previous part no longer works, because P (the victim) is using its own timestamp now instead of an attacker-provided timestamp. The attacker can receive HMACs of timestamps on the current day from the victim, but these will be useless for attacks on the next day. An attacker can't generate a valid HMAC on a timestamp of the next day because they don't have the user's secret key K_u, and HMAC is unforgeable without knowledge of the secret key.

Note that $HMAC(S, time)$ is probably a typo and should be $HMAC(K_u, time)$ to be consistent with step 3. Also note that K_u and k are used interchangeably in this subpart.

Just in case an attacker ever manages to get into the vault and install malware onto some of the machines, BearBank has tasked Frodo with installing a secure detection system on all of the machines.

(d) (4 points) For one approach, Frodo is thinking of configuring the machines to only allow read-operations. If any program tries to modify or delete data on a machine, it will explode and trigger an alarm in the vault. Which of the following detection approaches does this best represent? You do not need to explain your answer for this part.

○ Vulnerability based. ○ Anomaly based.

○ Signature based. ○ Behavioral based.

● Specification based. ○ Honeypot based.

Solution: Specification based detection is when you explicitly whitelist (or *specify*) what is allowed. In this case, the only thing that is allowed is read-operations. Violation against this specification triggers the alarm. Thus, specification best describes this approach.

(e) (4 points) Another approach Frodo has come up with is to install a bunch of dummy machines inside of the vault. If a new program is ever installed on any of these dummy machines, it will explode and trigger and alarm in the vault. Which of the following detection approaches does this best represent? You do not need to explain your answer for this part.

- ○ Vulnerability based.
- ○ Signature based.
- ○ Specification based.
- ○ Anomaly based.
- ○ Behavioral based.
- ● Honeypot based.

Solution: Honeypots are sacrificial "bait" machines that are intentionally meant to be attacked. Usually, we monitor for changes in the honeypot, which would signal that someone is trying to break into our system. This question is trying to nudge you towards the "honeypot" response by detailing that the machines are *dummy machines*.

(f) (8 points) Frodo has settled on a signature based HIDS that analyzes a program and achieves an 87% true positive rate and a 2% false positive rate for detecting whether a program is malware. However, Aragon, one of his co-workers has developed an anomaly based HIDS that analyzes a program and achieves a 95% true positive rate and a 5% false positive rate. Assume that one out of every ten-thousand programs that run on a machine in BearBank's vault is malware. The total cost of failing to detect a malware sample is \$100,000 dollars, the total cost of a false positive is \$1,000. Assuming these values hold over the long run, which detector should BearBank deploy? Select your answer from the multiple choice below and **explain your answer** in 1-2 sentences.

- ○ We cannot make this assessment since we do not know the base rate
- ○ Frodo's detector is better.
- ○ Aragon's detector is better.
- ○ They are equally good.
- ● Neither, they both cost too much.

Solution: In order to compare which detector is better, we need to know their FP and FN rates, as well as the relative cost of a FP and FN, AND the base rate of attacks to normal events. All of this information is provided in the question, so we can assess the cost of each detector in the long run.

Note that the true positive rate and false negative rate add to 1 (probability of alerting given there is an attack, plus probability of not alerting given there is an attack, equals the probability of either alerting or not alerting given there is an attack, which is 1, because you always either alert or don't alert).

- Frodo false negative rate: 100% - 87% = 13%
- Frodo false positive rate: 2%
- Aragon false negative rate: 100% - 95% = 5%

- Aragon false positive rate: 5%
- Base rate: 1/10,000
- Cost of each false negative: $100,000
- Cost of each false positive: $1,000

Cost of detector = Cost of false negatives + cost of false positives

= (# of attacks × FNR × cost of FN) + (# of non-attacks × FPR × cost of FP)

Cost for Frodo (per 10,000 requests):

(1 × 0.13 × 100,000) + (9,999 × 0.02 × 1,000) = $212,980

Cost for Aragon (per 10,000 requests):

(1 × 0.05 × 100,000) + (9,999 × 0.05 × 1,000) = $504,950

Note that it's actually cheaper to use neither detector in this case: in 10,000 requests, we expect 1 attack, which costs $100,000 if we don't find it. This is cheaper than the $212,980 or $504,950 it costs to use either detector.

To solve this without going through the entire derivation, you can use rough estimates to conclude the same thing:

In 9,999 ≈ 10,000 non-attack requests, Frodo will flag 2% as false positives, which is approximately 200 false positives. This costs 200 × $1,000 = $200,000 per 10,000 requests.

In 9,999 ≈ 10,000 non-attack requests, Frodo will flag 5% as false positives, which is approximately 500 false positives. This costs 500 × $1,000 = $500,000 per 10,000 requests.

Again, both detectors cost more than using no detector at all.

Problem 9 ***Name That Pwn*** **(15 points)**

For the following questions, some classic attacks are described. Identify which attack(s) they represent. Fill out all matching solutions.

(a) (3 points) My name is `Robert; drop table students;--`

- ○ Stored XSS
- ○ Reflected XSS
- ● SQLI
- ○ Clickjacking
- ○ Buffer Overflow
- ○ None of the Above

Solution: This is the overused example from the xkcd for SQL injection. Note the SQL syntax.

(b) (3 points) My name is `Robert <script src="https://www.evil.com/pwnme.js">`. Visit my Facebook page, OK?

- ● Stored XSS
- ○ Reflected XSS
- ○ SQLI
- ○ Clickjacking
- ○ Buffer Overflow
- ○ None of the Above

Solution: The script will be stored on the Facebook server, and people who visit the page will be hit with the XSS attack if it is unescaped by Facebook.

Note that JavaScript, which narrows down the choices to stored XSS or reflected XSS. Since the script is not in a URL, it's probably stored XSS.

(c) (3 points) My name is `Robert ... gah thats long ... shellcode goes here....`

- ○ Stored XSS
- ○ Reflected XSS
- ○ SQLI
- ○ Clickjacking
- ● Buffer Overflow
- ○ None of the Above

Solution: Shellcode injection is associated with buffer overflows.

(d) (3 points) Oh, Interesting web request... Lets inject a packet...

- ● NSA QUANTUM
- ○ PuppyKitty
- ● The Great Firewall
- ● Airpwn
- ○ Firesheep
- ○ None of the Above

Solution: Trivia question: the items listed in the solution all use packet injection as their main approach.

(e) (3 points) Hey, Ken, lets modify the compiler so that when it compiles login it inserts a backdoor...

- ○ Stored XSS
- ○ Reflected XSS
- ○ SQLI
- ○ Clickjacking
- ○ Buffer Overflow
- ● None of the Above

Solution: Compile time exploits/viruses do not correspond to any of these choices. You might think of buffer overflows, since compilers are related to memory safety. However, the compiler modification does not clearly take advantage of unprotected buffers and therefore "Buffer Overflow" is not the correct answer.

There is no need to penetrate a network when you can breach the people who run it. Networks are hard. People are soft. -Taylor Swift