Raluca Ada Popa
Spring 2018

CS 161 Computer Security

Midterm 2

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

I am aware of the Berkeley Campus Code of Student Conduct and acknowledge that academic misconduct will be reported to the Center for Student Conduct.

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: ______________________

You may consult one sheet of paper of notes. You may not consult other notes, textbooks, etc. Calculators, computers, and other electronic devices are not permitted.

If you think a question is ambiguous, please come up to the front of the exam room to the staff. If we agree that the question is ambiguous we will add clarifying assumptions to the central document projected in the exam rooms. Write your student ID on the top of every page.

You have 80 minutes. There are 7 questions, of varying credit (133 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.

SID ______________________________

Problem 1 ***True or False*** **(20 points)**

Answer True or False for each of the questions below. You do not need to explain your answers.

(a) TRUE or FALSE: Javascript running on `example.com/index.html` cannot manipulate webpages on `me.example.com`.

[✓ TRUE] FALSE

Solution: True, cf. Same-origin policy.

(b) TRUE or FALSE: If `example.com` loads Javascript from `me.example.com/evil.js`, then `me.example.com` can manipulate webpages on `example.com`.

[✓ TRUE] FALSE

Solution: True, this is an exception to the same-origin policy if the page itself loads the Javascript.

(c) TRUE or FALSE: It is impossible for a cookie set on `example.com` to be accessed by Javascript running on `me.example.com`. (Assume `example.com` is not vulnerable to XSS.)

TRUE [✓ FALSE]

Solution: False, if the cookie is set with `Domain=example.com` then the subdomains can access the cookie.

(d) TRUE or FALSE: A stateless packet filter can drop packets belonging to a connection whose first packet payload (the first byte) starts with the character "x".

TRUE [✓ FALSE]

Solution: False, since it is stateless and does not remember which connections started with "x" or not.

(e) TRUE or FALSE: A DNS server that receives multiple UDP packets from the same client can be assured (with high probability) that these packets are in the order that they were sent. (Assume no attackers.)

TRUE [✓ FALSE]

Solution: False, since UDP does not provide any ordering guarantees. It is possible that the two packets travel different routes, and the one sent later gets there first.

(f) TRUE or FALSE: If an attacker guesses the IPs, ports, TCP sequence numbers and TLS sequence numbers in a TLS connection, then they can successfully inject traffic into the connection.

TRUE [✓ FALSE]

Solution: False, the attacker would need to know the symmetric TLS keys.

(g) TRUE or FALSE: If you join an unsecured network with an on-path eavesdropper, then the attacker could hijack all of your HTTP requests to `www.google.com`. (Assume all caches are empty.)

[✓ TRUE] FALSE

Solution: True, the attacker could hijack DHCP to set a malicious DNS server.

(h) TRUE or FALSE: Without dropping packets, an on-path attacker can still stop you from successfully creating a TLS connection with `https://google.com`. (Assume that `google.com`'s IP address is in your DNS cache.)

[✓ TRUE] FALSE

Solution: True, the attacker can see your packets, and attack at the TCP protocol level (by sending RST packets). Alternatively, they could send malformed TLS packets which would cause the connection to be aborted. Stated another way: TLS provides confidentiality, integrity and authenticity, but not availability.

(i) TRUE or FALSE: A man-in-the-middle attacker can successfully impersonate `https://google.com` if they forge a DNS response for `google.com`.

TRUE [✓ FALSE]

Solution: False, they will not have a valid certifcate / private key pair.

(j) Say that we have two detectors. We combine them in series, by only triggering an alert if both detectors detect an attack. TRUE or FALSE: The new false positive rate is always less than or equal to the false positive rate of both detectors.

[✓ TRUE] FALSE

Solution: True. Some of the false positives of the first detector will be eliminated by the second not catching it, and vice-versa.

Problem 2 ***A/B Testing*** **(27 points)**

For each of the questions below, select all of the technologies which would work for the scenario described. **You may select multiple choices, or none of the choices.**

(a) We know the IP address of a host, and we want to lookup the MAC address. (Assume we are already established on the network.)

■ ARP ☐ DHCP

☐ DNS ☐ HTTP

Solution: ARP performs IP to MAC lookups.

(b) Alyssa is about to open her laptop, connect to unsecured Starbucks WiFi, and load the URL `http://squigler.com/feed`. A man-in-the-middle on her local network wants to steal her Squigler cookies. (Assume that Squigler does not implement any defences. All caches are empty.)

■ ARP Spoofing ☐ A reflected XSS on `/feed`

☐ UDP Spoofing ■ A stored XSS on `/feed`

■ DHCP Spoofing ■ A buffer overflow on `squigler.com`

■ TCP Spoofing ☐ A root CA compromise

Solution: ARP Spoofing: MITM sends Alyssa an ARP packet, claiming to be the router gateway. Now the MITM receives all of Alyssa's packets.

UDP Spoofing: This one was a little ambiguous. It is useful to spoof UDP for the DNS resolution, but that doesn't give you the ability to spoof the DNS transaction ID. We decided to accept solutions which indicated either.

DHCP Spoofing: MITM sends Alyssa a DHCP offer, claiming to be the router gaterway. Now the MITM receives all of Alyssa's packets.

TCP Spoofing: This one is a tricky one! MITM waits until Alyssa connects to Squigler, then injects a TCP packet which appears to be the HTML page for `/feed`. This HTML page contains Javascript which sends the cookie to the MITM controlled site.

Reflected XSS: The MITM cannot use a reflected XSS. Reflected XSS relies on sending the target a malicious URL, which contains the script which will be echoed back into the page. This does not work, since Alyssa loads the feed page herself.

Stored XSS: The MITM makes a stored XSS which sends the cookie to them.

Buffer overflow: If there is a buffer overflow on Squigler, then the MITM can compromise the webserver. At that point, the webserver receives Squigler's cookies already.

A root CA compromise: This does not help, no HTTPS is being used so compromising a Certificate Authority has no effect.

(c) The user is visiting a website vulnerable to an XSS attack. Mark any technology that helps prevent the XSS attack or mitigate its effects.

■ Content-Security Policy ☐ Secure cookie flag

☐ VPN ■ Escape user input

☐ HTTPS ☐ Prepared statements

☐ DNSSEC ■ HttpOnly cookie flag

Solution: Content-Security Policy: A strong CSP can block inline scripts, which can help prevent XSS.

Escape user input: Escaping user input is the way to prevent XSS.

HttpOnly cookie flag: This prevents Javascript from accessing the cookie, which mitigates the impact of an XSS.

Some distractors:

VPN, **HTTPS**, **DNSSEC**: These all try to ensure that a man-in-the-middle does not mess with the server's answers. In the case where the server is sending the malicious data like in a stored XSS, this does not help.

Secure cookie flag: The secure cookie flag ensures a cookie is sent over HTTPS, but Javascript on an HTTPS site will still be able to read the cookie.

Prepared Statements: This prevents against SQL injection, not against XSS.

(d) We want to create an intrusion detection system which will work on a new class of never-before-seen attacks.

■ Anomaly-based ■ Specification-based

☐ Signature-based ■ Behavioral-based

Solution:

Anomaly-based detectors can detect new classes of attacks if they cause anomalies (for example, an anomaly of using too much CPU could indicate a DoS attack).

Specification-based detectors can detect attacks which violate the specifications.

Behavioral-based detectors can detect new attacks if the attack performs some bad behavior.

While signature-based detectors can detect new attacks (if they match the signature), the signature must be rewritten for each *class* of attack.

(e) We want to detect any attempted accesses to the file `/etc/passwd`.

■ HIDS ■ NIDS

Solution: A HIDS can detect accesses by, for example, hooking into syscalls.

A NIDS can detect accesses by looking for the string `/etc/passwd` in the request.

Note that both methods have false positives and false negatives, like all realistic detectors must.

(f) We want to detect any malware sent as email attachments.

■ HIDS ■ NIDS

Solution:

A HIDS can perform malware scanning when the email attachment is received on the user's computer. (This is how many antiviruses work.)

A NIDS can detect accesses when the email attachment is received by the mail server. (Google Mail does this.)

(g) Considering an on-path/man-in-the-middle attacker between a DNS resolver and a DNS server, we want to prevent a DNS spoofing attack in which the DNS resolver receives an incorrect IP address for a DNS lookup on a domain name to the DNS server. Select all technologies that can prevent the attacker from modifying the contents of the DNS reply from the DNS server.

■ TLS ☐ Firewall

☐ TCP ☐ HIDS

☐ UDP ■ DNSSEC

Solution:

TLS and DNSSEC both provide integrity and authenticity, which allows us to ensure that the attacker cannot modify the DNS response.

None of a TCP, UDP, Firewall and HIDS can provide integrity.

(h) Consider an attacker stole the password database from a server and wants to reverse Alice's password P in particular. Assume Alice chose P by choosing one word at random from the English dictionary, and the attacker knows this. Assume that hashes are stored alongside passwords in the database. Select all ways the server can store Alice's password in the password file that would prevent an attacker from determining Alice's password.

☐ Hashed password ($H(P)$) ☐ Hash of P using a slow hash (100 applications of SHA-256)

☐ Salted hash of P, with same salt per user

☐ Salted hash of P, different salt per user ■ Encrypted P using AES-CBC

Solution: Since Alice has a weak password, there is no way to prevent it from being cracked. (All of the words in the English language only gives about 18 bits of entropy.)

The exception is encryption with AES-CBC. We can store the key on the server itself. The attacker has only stolen the password database, which does not contain not need to contain the key. Even if the attacker steals the database, they cannot tell the difference between Alice's password and any other possible decryption.

(i) As in the question above, but now assume Alice chose P to be a random string from a large set (2^{200} possibilities). Select all ways the server can store Alice's password in the password file that would prevent an attacker from determining Alice's password.

■ Hashed password ($H(P)$) ■ Hash of P using a slow hash (100 applications of SHA-256)

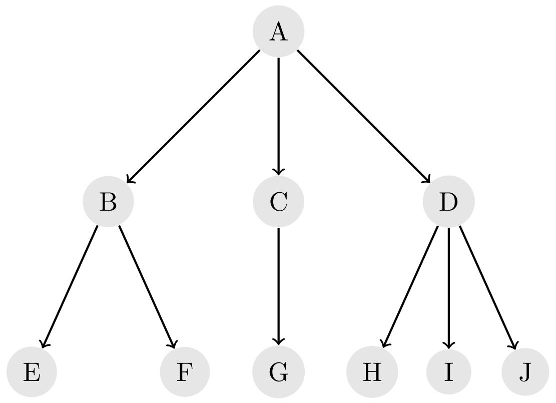
■ Salted hash of P, with same salt per user

■ Salted hash of P, different salt per user ■ Encrypted P using AES-CBC

Solution: All of these methods are one-way, and an attacker who wanted to find Alice's password would need to guess from 2^{200} possibilities, which is clearly impossible.

A side note: A lot of people seem to be confused with the presence of the word "prevent" in the question. We often ignore probabilities which are very small, such as how we say that a MAC "prevents" an attacker from forging messages when there really is a small chance that the MAC is randomly correct. 2^{200} is unfathomably large; there's just no way to break that sort of entropy.

Problem 3 ***DNS Tree*** **(20 points)**



This tree describes a name server hierarchy. The node `A` is a top level domain (like `.com`, `.edu`), and each child name server controls a subdomain of its parent. Each letter corresponds to both the nameserver and the name of the domain. For example, an uncached recursive query for `https://inst.E.B.A` would result in queries to NS A, then NS B, then NS E.

Assume this is the complete nameserver hierarchy for `A`. No other nameservers under `A` exist apart from the ones depicted.

(a) Which one of the following nameservers could theoretically provide the most number of additional records in its response?

○ B ○ D

○ C [✓ Not enough information]

Solution:

Not enough information. We have no idea how many domains (not name servers) exist per zone.

(b) If we use the most basic version of DNS, in which a resolver accepts any response over any domain from any nameserver, and NS D is compromised, which domains are safe from cache poisoning? (Assume D is contacted during resolution.)

Solution:

None of the domains are safe. NS D can cache poison any domain in the hierarchy.

(c) What domains should D be allowed to give additional records for to protect against cache poisoning?

Solution:

Subdomains H,I,J, or the rule *.D.A (technically also D.A itself but this was not explicitly covered in Spring 2018)

(d) Now suppose the nameserver hierarchy operates on DNSSEC. During resolution, the resolver sends a query for `inst.G.C.A` to nameserver C.

i. What public key should be included in C's response? Choose one.

inst C [✓ G] A

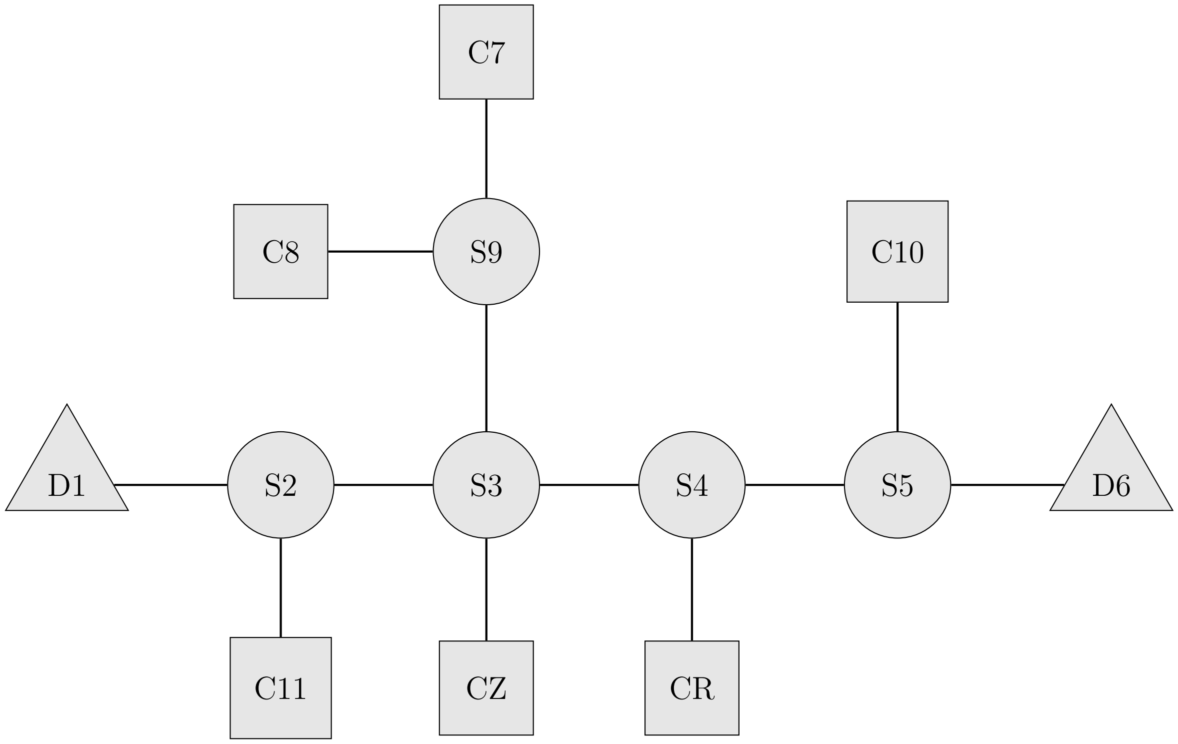
Solution: G's public key

ii. If the root key is compromised, can the resolver trust NS C's response?

Yes [✓ No]

Problem 4 ***Fun with L2*** **(20 points)**

Tux has gone undercover and wants to steal data from his enemy, **Rex**, who has plugged into the network with his computer labeled **CR**. Assume that the network is configured as such:



Note the tree-structure of the network.

Tux can choose **one** of the following options:

1. Taking control of some number of the DHCP servers (marked as triangles)
2. Taking control of some number of the switches (marked as circles)
3. Taking control of some number of the computers (marked as square)

Assume that if Tux has control of a switch, he has the power to drop, intercept, and modify packets. If Tux has control of a computer, he can send spoofed packets out to the network. However, the network is unpredictable and packets can arrive out of order, so you may not assume that the order of packets is tied to the number of links. For example, the packet from 6 may reach 11 before the packet from 1.

For each of the possible options, select the devices that Tux has to control to accomplish his mission. Tux wants to have to control the minimum number of devices. If there are multiple possible solutions with the same size, break ties by the solution **with the smallest numerical device**. For example, if both switch sets $\{2, 5, 9\}$ and $\{3, 4, 9\}$ are valid solutions, you should bubble $\{2, 5, 9\}$. Consider each option alone, namely, that the devices in that option are the only devices to be compromised.

SID ________________________________

(a) Suppose Tux wants to give a fake IP address to Rex, who is just joining the network. Tux wants this attack to work 100% of the time.

i. Option 1: DHCP Servers

■ D1　　■ D6

> **Solution:**
>
> Tux needs to ensure that he can send a fake DHCP offer before Rex receives a real one. To do so, he must control both DHCP server.

ii. Option 2: Switches

☐ S2　　☐ S3　　■ S4　　☐ S5　　☐ S9

> **Solution:**
>
> If Tux controls switch 4, he can simply drop all DHCP packets to Rex and spoof different ones.

iii. Option 3: Computers

☐ C7　　☐ C8　　☐ C10　　☐ C11

> **Solution:**
>
> None of the computers *ensure* that Tux can send a fake DHCP offer before Rex gets a real one.

(b) Tux wants to give the fake IP address to Rex, but this time, Tux is okay with the attack working only part of the time.

i. Option 1: DHCP Servers

■ D1　　☐ D6

> **Solution:**
>
> Here Tux only needs to sometimes win the race, which he can do by sending any DHCP offer. To do so, he just needs to be able to send packets on the network.

ii. Option 2: Switches

■ S2　　☐ S3　　☐ S4　　☐ S5　　☐ S9

> **Solution:**
>
> Here Tux only needs to sometimes win the race, which he can do by sending any DHCP offer. To do so, he just needs to be able to send packets on the network.

iii. Option 3: Computers: Purposely omitted.

(c) Tux wants his computer (located at CZ) to be sent **all** of Rex's outgoing Layer 3 (IP) traffic (but doesn't care about Rex's Layer 2 traffic). He intends to do this by spoofing DHCP packets. Which of the four steps of the DHCP protocol should Tux aim to spoof?

Solution: Offer step

(d) Tux wants the above attack to work 100% of the time. Which devices should he control? (Same rules as above)

i. Option 1: DHCP Servers

■ D1 ■ D6

Solution: Same as part (a).

ii. Option 2: Switches

☐ S2 ☐ S3 ■ S4 ☐ S5 ☐ S9

Solution: Same as part (a).

iii. Option 3: Computers

☐ C7 ☐ C8 ☐ C10 ☐ C11

Solution: Same as part (a).

Problem 5 ***Blind Signatures*** **(15 points)**

A *blind signature scheme* is a digital signature scheme where the contents of the original message are hidden from the person signing the message. Specifically, we define the following cryptographic primitives:

1. KeyGen(): returns a public key pk and a secret key sk.
2. Sign(sk, m): signs m, as in traditional signature schemes.
3. Verify(pk, s, m): verifies the signature s on the message m, as in traditional signature schemes.
4. Blind(m, r): takes in a message $m \in [0, N]$ and a random number r. It returns a "blinded" message m'.
5. Unblind(s', r): when applied to a valid signature s' on a blinded message m' with the original randomness r, Unblind produces a valid signature s on the original message m.

For the purpose of this question, we will assume that our messages, signatures and randomness are all encoded as numbers modulo some N. Assume that all math occurs modulo N.

If Alice wants Bob to blindly sign the message m, she can perform the following protocol:

1. Alice generates a random number r with $0 \leq r < N$.
2. Alice computes $m' \leftarrow$ Blind(m, r) and sends m' to Bob.
3. Bob computes $s' \leftarrow$ Sign(sk, m') and sends s' to Alice.
4. Alice computes $s \leftarrow$ Unblind(s', r).
5. Alice checks Verify(pk, s, m) to make sure that Bob is giving her a real signature.

A signature scheme is **valid**, if, for every correct run of Steps 1–4, it verifies correctly in Step 5 (except possibly with some negligible probability of failure).

Moreover, we say that it is **unforgeable** if Alice cannot create a signature on a message m for which she did not follow the protocol above, similarly to regular digital signatures.

We consider the following two possible definitions for blindness:

1. **Blindness I** (Random messages): Alice randomly chooses two messages m_0 and m_1. Alice flips a coin b to select either m_0 or m_1, and sends a blinded message m'_b to Bob. Alice then shows Bob both m_0 and m_1. Bob wins if he can identify which of m_0 and m_1 was blinded. If Bob cannot win with probability significantly greater than $\frac{1}{2}$, we say that the scheme satisfies **Blindness I**.
2. **Blindness II** (Bob chooses): Bob chooses two messages m_0 and m_1, and gives both to Alice. Alice flips a coin b to select either m_0 or m_1, and sends a blinded message m'_b to Bob. Bob wins if he can identify which of m_0 and m_1 was blinded. If Bob cannot win with probability significantly greater than $\frac{1}{2}$, we say that the scheme satisfies **Blindness II**.

Now we will consider some candidates for blind signature schemes. For each of the schemes below, identify which of the properties it supports. In all the schemes below, KeyGen, Sign and Verify are identical to the RSA signature scheme as presented in class, **except no hash is applied to the message before signing or verification**. The number N is set to be the public key. If a scheme is not a valid blind signature scheme, you do not need to mark its other properties.

[Quick RSA reminder: $sk = d$, Sign$(sk, m) = H(m)^d \mod N$, Verify$(pk, s, m) : s^3 \mod N \stackrel{?}{=} H(m) \mod N$.]

(a) $\textsc{Blind}(m, r) = mr^3$ and $\textsc{Unblind}(s', r) = s'r^{-1}$.

[✓ Valid scheme] ○ Invalid scheme

[✓ Blindness I (Random messages)] □ Unforgeable

□ Blindness II (Bob chooses)

Solution:

Valid scheme: if we simplify the steps in the protocol above, we see that a scheme is valid only if the following verification works:

$$\textsc{Verify}(pk, \textsc{Unblind}(\textsc{Sign}(sk, \textsc{Blind}(m, r)), r), m)$$

Now it is just a matter of plugging in definitions:

$$\textsc{Verify}(pk, \textsc{Unblind}(\textsc{Sign}(sk, mr^3), r), m)$$

$$\textsc{Verify}(pk, \textsc{Unblind}((mr^3)^d, r), m)$$

$$\textsc{Verify}(pk, \textsc{Unblind}(m^d r^{3d}, r), m)$$

Recall that in RSA we have that $3d \equiv 1 \pmod{\phi(N)}$, so that $r^{3d} = r$.

$$\textsc{Verify}(pk, \textsc{Unblind}(m^d r, r), m)$$

$$\textsc{Verify}(pk, m^d, m)$$

Now our verification step checks $m^d \equiv m^d$, which verifies.

Blindness I: Say that Alice generates two random messages m_0 and m_1. Then, we generate two random nonces r_0 and r_1. Note that $r \mapsto r^3$ is a permutation, and so multiplying by r_0^3 and r_1^3 cause m'_0 and m'_1 to look completely random.

Not Blindness II: Bob can take $m_0 = 0$ and $m_1 = 1$. If Alice sends 0, Bob knows she blinded m_0. Otherwise, she blinded m_1.

Not Unforgeable: Alice can take $s = 0$ and $m = 0$. (Or for that matter, any forging technique on unpadded RSA works.)

(b) $\textsc{Blind}(m, r) = H(mr^3)$ and $\textsc{Unblind}(s', r) = s'H(r^{-1})$, where H is a cryptographic hash function.

○ Valid scheme [✓ Invalid scheme]

□ Blindness I (Random messages) □ Unforgeable

□ Blindness II (Bob chooses)

Solution: This is not valid.

(c) $\textsc{Blind}(m, r) = H(m)r^3$ and $\textsc{Unblind}(s', r) = s'r^{-1}$, where H is a cryptographic hash function.

○ Valid scheme [✓ ● Invalid scheme]

☐ Blindness I (Random messages)

☐ Unforgeable

☐ Blindness II (Bob chooses)

Solution:

This is not valid.

Note it would be valid if we hashed in the verification step, but we do not according to the problem statement. We decided to give full credit for solutions which thought it was valid because they thought hashing should be applied before the verification step, as this is really a minor point.

If you chose "Valid", the proper solution is: Blindness I, Blindness II, and Unforgeable.

Blindness I: This is true for the same reasons as in part (a).

Blindness II: Bob cannot find a message with $H(m) = 0$ since a hash function is preimage resistant.

Unforgeable: This is true for the same reasons as in traditional signature schemes.

Problem 6 ***Blue and Red Teams*** **(16 points)**

Give only one answer to each question.

(a) Answer the following short-answer questions about network security defenses.

i. Explain the meaning and purpose of the rule `ext allow tcp *:80 -> *:* if ACK set`.

> **Solution:**
>
> This allows all packets from established port 80 (typically HTTP) connections which are coming in to the `ext` interface. (This might be used for a webserver which needs to accept connections from clients, but is not allowed to start its own port 80 connections.)
>
> The ACK suggests an existing connection, and although it could be spoofed that is the "purpose".
>
> In grading, we are looking for the following points:
>
> 1. A solution can assume that the `ext` interface refers to all external connections.
> 2. A solution can to use "port 80" instead of HTTP, but it must mention somehow what ports are allowed.
> 3. A solution must mention the directionality, that this rule refers to packets coming into the `ext` interface of the firewall.
> 4. A solution cannot just say "if the ACK flag is set", we are looking for an interpretation (something about the connection being "established" is sufficient).

ii. In class, we saw an attack where a packet containing something "bad" (the word "root") was split over multiple packets to evade detection. Name one technology that can detect the correct byte sequence received by the destination.

> **Solution:** Use a HIDS which runs on the local operating system after the TCP packets have been reconstructed.
>
> We also accepted solutions like "TCP reconstruction on the NIDS".
>
> Note a stateful packet filter is *not* sufffficient alone, since a stateful packet filter does not necessarily look at the contents of the actual TCP packets.

(b) Answer the following short-answer questions about web attacks.

i. Boogle uses the following PHP code to display what a user has searched for.

```
$SEARCHTERM = $_GET['QUERY'];
echo "You searched for: <b>$SEARCHTERM</b>";
```

What query should Alyssa enter to make the site display an alert box saying hacked?

> **Solution:** Enter `<script>alert("hacked");</script>`.
>
> In general, we did not expect the exact syntax of Javascript. Things like `<script> alert hacked </script>` received tiny deductions, as long as it is clear what the solution is attempting to do.
>
> It is **not** okay if the solution treats the input as if it will be injected into the PHP code. For example, something like `'' alert hacked; ``` is unacceptable. The $SEARCHTERM variable is not being interpreted as PHP, it is being interpreted as HTML.

ii. Boogle's website performs the following database query for every search query:

```
INSERT INTO query_log VALUES ('$SEARCHTERM');
```

where $SEARCHTERM is the search term that a user searches for. Moreover, Boogle checks the syntax of SQL queries before they are executed, and will not execute them if their syntax is invalid (i.e., quotes/parenthesis are closed incorrectly, missing semicolon separating queries). Boogle also does not allow the command "`--`" (SQL comment) in queries because it saw in CS 161 that it can be exploited by hackers.

Boogle also has a table `prohibited_users` where it stores (in a column called `user`) the name of each user who is not allowed to access certain parts of its site. Eve knows that her name 'Eve' is in that database and wants to remove it. She has access to the form that triggers the insert query above. What should Eve enter in the search box to do so? (Keep in mind Boogle's filter described above.)

Solution: An example solution:

```
'); DELETE FROM prohibited_users WHERE user = 'Eve';
INSERT INTO query_log VALUES ('
```

A correct solution will:

1. End the single quote (`'`), close the parenthesis (`)`) and end the statement `;`.
2. Attempt to delete or remove from the `query_log` table using SQL syntax like `DROP`, `TRUNCATE` or `DELETE`. You do not need to know the exact syntax of these queries, so it is ok if the query is not properly formatted as long as the syntax with respect to semicolon, quotes and parenthesis is ok. In particular, a solution needs to have correct syntax when they are closing the $SEARCHTERM quotes and handling the end of the `INSERT` query.
3. Start a new statement such that appending `');` will create a SQL statement with valid syntax.

Problem 7 ***New TLS*** **(15 points)**

You are in charge of creating the new TLS specification! You have received the following suggestions by email. **Evaluate** each suggestion by determining if it is secure or insecure, and then **explain** your answer.

(a) Consider Diffie-Hellman TLS. It is possible that the connection dies due to network issues. SUGGESTION: Let the client restart the connection by sending the plaintext PS to the server. The server will keep track of all recently used PS values and their corresponding keys, allowing for easy resumption. Evaluate and explain.

Solution:

Awful. Sending the PS in plaintext allows an attacker to see it. Using the PS, R_b and R_s, the attacker can reconstruct the TLS symmetric keys.

Vague justifications like "the attacker should not learn the PS" got partial credit.

(b) Consider Diffie-Hellman TLS. SUGGESTION: Rather than having the server choose g and p and send them, have the client decide on the values as part of the ClientHello message. The server no longer sends g and p, and only sends $g^a \mod p$ along with a signature. The server assumes that the client chooses appropriate values for g and p. Evaluate and explain.

Solution:

Bad. Say that a client is currently connecting to the server, and it chooses some good values for g and p.

An attacker connects to the server, purposefully using weak g' and p values. For example, they can use $g' = 1$. (Other values of g' work, although some care is needed to select them.)

Now the server sends back a signed message with $(g')^a = 1$. Then the attacker can continue the session with the client, presenting a signed $(g')^a = 1$. The client sends g^b, which the MITM can ignore since they already know the client will calculate $(g')^{ab} = 1$. Then both sides agree on the premaster secret $(g')^{ab} = 1$. This allows the attacker to pretend to be the server, since it has successfully agreed on a premaster secret.

(c) Consider RSA TLS. SUGGESTION: Rather than sending $\{PS\}_{K_{\text{server}}}$, we have the client choose the server keys C_s, I_s. It then sends $\{C_s, I_s\}_{K_{\text{server}}}$ to the server. Then the server chooses the client keys C_b, I_b and sends $\{C_b || I_b, \text{MAC}_{I_s}(C_b || I_b)\}_{C_s}$ to the client. Evaluate and explain.

Solution:

Bad-ish. This leaves us more vulnerable to replay attacks if the server's randomness R_s is not secure, since nothing here contains the nonces R_b and R_s.

In particular, say that we have a client connect to a server, and an attacker logs all of this information. Assume that the server's randomness R_s is always 0. Then an attacker can replay the entire TLS connection against the server. The old TLS is not vulnerable to this, since the client's randomness is incorporated into the connection.