# Final exam review

EECS 388

## Logistics

- Time: Wed 4/29, 7-8:30PM
- Location:
  - Stamps Aud ('DE'->'Z')
  - Beyst 1670 ('A'->'DD')
  - Beyst 1690 (Additional time)

- One cheat sheet (two pages of 8.5x11 in) allowed

## Main topics

- Crypto
- Web security
- Network security
- Control hijacking
- Forensics
- Other topics, e.g., side channels, anonymity…

TRUE or FALSE?

- The SHA256 hash of a user-generated passphrase makes a good key for an encryption scheme because hashes are random.
  - FALSE. Human-selected passphrases don't have enough entropy and are susceptible to offline guessing attacks

- The reason that AES-CBC mode is preferred over ECB mode is because CBC provides integrity.
  - FALSE. CBC mode doesn't provide integrity. The reason it is preferred is because ECB mode reveals partial information about the plaintext.

## TRUE or FALSE?

- A denial-of-service attack requires injecting spoofed packets.
  - FALSE. You can do denial-of-service without sending spoofed packets, but you may expose your IP addresses.

- TLS provides both confidentiality and integrity for messages sent using it.
  - TRUE.

## TRUE or FALSE?

- If there are only a few possibilities for M, an eavesdropper who sees SHA256(M) can figure out what M was.
  - TRUE. Given a guess g at the message, the attacker can compute SHA256(g) and check whether it matches the known message; the attacker can repeat this once for each possibility for the message.
- Given x^y mod p, x, and p (where p is a 2048-bit prime), there is no known, efficient way to compute y.
  - TRUE. Discrete logarithm problem.

## TRUE or FALSE?

- In onion routing (e.g., Tor), if one of the mixes is dishonest, then there is no guarantee of anonymity for the user.
  - FALSE. Tor is designed to provide anonymity as long as at least one mix is honest.

- A surefire way to avoid censorship systems is to just encrypt all of your communications.
  - FALSE. The censors can block encrypted communications (or encryption software).

## Fill in the blank

- An attack that reconstructs what you are typing on your keyboard by recording and then analyzing the specific sounds made as you type would be an example of a _______ attack.
  - Side channel
- When you click on a link, the ______ header tells the server which URL you were at that took you to the link.
  - Referer

## Very Short Answer

- All else being equal, would a smart attacker prefer a denial-of-service attack with amplification factor 3 or amplification factor 17?
  - 17
- Name two techniques to avoid or defend against replay attacks.
  - Timestamps. Counters. Nonces.

## Short Answer

- Does TLS defend against SQL injection attacks? Why or why not?
  - No. TLS only protects the data while in transit, but doesn’t stop a malicious client from sending a request containing malicious data.

## Very Short Answer

- All else being equal, would a smart attacker prefer a denial-of-service attack with amplification factor 3 or amplification factor 17?
  – 17
- Name two techniques to avoid or defend against replay attacks.
  – Timestamps. Counters. Nonces.

## Web security

- FaceSpace has implemented an all-new friend finder feature! Now, users of FaceSpace can enter a search string such as "rohin" in order to find people to friend. FaceSpace also keeps track of the most popular search strings. Since the feature is new, the most popular search strings at the moment have only been searched for a few hundred times.

- When the user visits a URL such as

**https**://www.facespace.com/friendfinder.php?name=rohin, Facespace runs the following (in pseudocode):

```
• name = getParameterFromUrl(url, 'name');
  increment_number_of_searches(name);
• command = "SELECT username FROM users WHERE name =
  '" + name + "'";
• results = execSQLForTable(command, "users");
• html = "<p>You searched for: " + name + ".</p><p>"
  + results + "</p>"; send_to_client(html);
```

## Web security

- When the user visits a URL such as

**https**://www.facespace.com/friendfinder.php?name=rohin

- `name = getParameterFromUrl(url, ’name’); increment_number_of_searches(name);`
- `command = "SELECT username FROM users WHERE name = ’" + name + "’";`
- `results = execSQLForTable(command, "users");`
- `html = "<p>You searched for: " + name + ".</p><p>" + results + "</p>"; send_to_client(html);`
- The code above first extracts “rohin” from the URL (simply by scanning for “name=” and returning whatever is after that). It issues an SQL query to the users table, which contains the username, password (in cleartext), and name of all of the FaceSpace users. The developers have made sure to ensure that the SQL command can access only the “users” table. The code sends back the results to the user.

## Web security

- When a user asks for the most popular search strings, FaceSpace goes through the database containing the number of searches for each search string, and returns the top 100 search strings sorted by number of searches.
- Alice is a FaceSpace user. She does not care about the most searched names, so she will never view the top 100 search queries. However, since she knows Mallory, she will visit any site that Mallory sends to her.
- Bob is another FaceSpace user. He does not know Mallory and is cautious by nature, so he will not visit any sites recommended by Mallory. He likes to follow celebrities and so views the top 100 search queries page frequently.
- Charlie is another FaceSpace user. He will not visit any sites recommended by Mallory and does not care about the most searched names, so he will never view the top 100 search queries. He is very careful and if anything seems suspicious, he will close his browser and go do something else.
- Mallory is an off-path attacker who has an account on FaceSpace.

- Mallory wants to steal Alice’s FaceSpace cookie. Can she?
  - Yes. The attack is to send Alice a link to a search results page where the name contains Javascript,

  e.g., https://www.facespace.com/friendfinder.php?name=<SCRIPT>...</SCRIPT>.
- Mallory wants to steal Bob’s FaceSpace cookie. Can she?
  - Yes. Mallory could perform a bunch of repeated searches for the same name to get it listed on the top-100 page—sneakily choosing a name that contains Javascript—and then wait for Bob to visit the top-100 page.
- Mallory wants to steal Charlie’s FaceSpace cookie.
  - No.
- Mallory wants to steal Alice’s FaceSpace password.
  - Yes Attack name: SQL injection

  she could search for something like

  ’; SELECT * FROM users; -- and the results page might reveal everyone’s password.

- Mallory wants to steal Bob's FaceSpace password. Can she?
  - Yes. Attack name: SQL injection
- Mallory wants to steal Charlie's FaceSpace password. Can she?
  - Yes. Attack name: SQL injection

- FaceSpace hires a security auditor, who recommends that they add an unpredictable CSRF token to the search form and search URL. Thus, the search URL now looks like
- https://www.facespace.com/friendfinder.php?token=1A0B743FC08DA37A&name=rohin
- The code for that page is modified to first check that the token is correct; if it is incorrect or missing, none of the rest of the code is executed and the page doesn't load. Nothing is changed about the code for the page with the top 100 search strings.
- Which of above goals become impossible, once this change is made?

- Mallory wants to steal Alice's FaceSpace cookie.
  - The attack is to send Alice a link to a search results page where the name contains Javascript,

  e.g., https://www.facespace.com/friendfinder.php?name=<SCRIPT>...</SCRIPT>.

  - Yes. The attack is to send Alice a link to a search results page where the name contains Javascript,

  (no longer possible under the new scheme)

  - Mallory could perform a bunch of repeated searches for the same name to get it listed on the top-100 page—sneakily choosing a name that con- tains Javascript—and then wait for Bob to visit the top-100 page.
  - Then send Alice a link to the top-100 page.

**Software security**

The following C code has a vulnerability:

```
/* requires: len == size(in) */
void vuln(uint64_t in[], size_t len {
  char a[20];
  size_t i;
  for (i=0; i<len && i<20; i++) {
   memcpy(&(a[i]), &(in[i]), sizeof(uint64_t));
  }
}
```

- Assume that the elements and length of in are controlled by the attacker, but it is guaranteed that vuln() will be called with arguments where len is equal to the number of elements in in (i.e., assume that the precondition of vuln() always holds).
- vuln() is vulnerable to a buffer overflow attack. Explain why, and given an example value of len that would enable such an attack.

The following C code has a vulnerability:

```
/* requires: len == size(in) */
void vuln(uint64_t in[], size_t len {
  char a[20];
  size_t i;
  for (i=0; i<len && i<20; i++) {
   memcpy(&(a[i]), &(in[i]),
  sizeof(uint64_t));
  }
}
```

- Each memcpy() writes 8 bytes, which will write past the end of a if i is 13 or larger. (The call to memcpy() overwrites a[i..i+7], and if i+7 is 20 or larger, this writes outside the bounds of a.)