

Web Security II

Question 1 *Session Fixation*

A *session cookie* is used by most websites in order to manage user logins. When the user logs in, the server sends a session cookie to the user's browser. The server also stores the cookie value in a database along with the corresponding username. The user's browser sends the session cookie to the server whenever the user loads any page on the site. The server then looks the session cookie up in the database and retrieves the corresponding username. Using this, the server can know which user is logged in.

Some web application frameworks allow cookies to be set by the URL. For example, visiting the URL

`http://foobar.edu/page.html?sessionId=42.`

will result in the server setting the `sessionId` cookie to the value "42".

- (a) Can you spot an attack on this scheme?
- (b) Suppose the problem you spotted has been fixed as follows: `foobar.edu` now establishes new sessions with session IDs based on a hash of the tuple (`username`, `time of connection`). Is this secure? If not, what would be a better approach?

Question 2 *Cross-Site Request Forgery (CSRF)*

In a CSRF attack, a malicious user is able to take action on behalf of the victim. Consider the following example. Mallory posts the following in a comment on a chat forum:

```

```

Of course, Patsy-Bank won't let just anyone request a transaction on behalf of any given account name. Users first need to authenticate with a password. However, once a user has authenticated, Patsy-Bank associates their session ID with an authenticated session state.

- (a) Sketch out the process that occurs if Alice wants to transfer money to Bob. Explain what happens in Alice's browser and patsy-bank.com's server, as well as what information is communicated and how.
- (b) Explain what could happen when Alice visits the chat forum and views Mallory's comment.
- (c) What are possible defenses against this attack?

Question 3 *CSRF++*

Patsy-Bank learned about the CSRF flaw on their site described above. They hired a security consultant who helped them fix it by adding a random CSRF token to the sensitive `/transfer` request. A valid request now looks like:

```
https://patsy-bank.com/transfer?to=bob&amount=10&token=<random>
```

The CSRF token is chosen randomly, separately for each user.

Not one to give up easily, Mallory starts looking at the welcome page. She loads the following URL in her browser:

```
https://patsy-bank.com/welcome?name=<script>alert("Jackpot!");</script>
```

When this page loaded, Mallory saw an alert pop up that says “Jackpot!”. She smiles, knowing she can now force other bank customers to send her money.

- (a) What kind of attack is the welcome page vulnerable to? Provide the name of the category of attack.

- (b) Mallory plans to use this vulnerability to bypass the CSRF token defense. She’ll replace the `alert("Jackpot!");` with some carefully chosen JavaScript. What should her JavaScript do?

- (c) Mallory wants to attack Bob, a customer of Patsy-Bank. Name one way that Mallory could try to get Bob to click on a link she constructed.