Peyrin Kao
Fall 2022

CS 161
Computer Security

Discussion 8

Question 1 *Cross-site not scripting* ()

Consider a simple web messaging service. You receive messages from other users. The page shows all messages sent to you. Its HTML looks like this:

```
Mallory: Do you have time for a conference call?
Steam: Your account verification code is 86423
Mallory: Where are you? This is <b>important!!!</b>
Steam: Thank you for your purchase
       <img src="https://store.steampowered.com/assets/thankyou.png">
```

The user is off buying video games from Steam, while Mallory is trying to get ahold of them.

Users can include **arbitrary HTML code** messages and it will be concatenated into the page, **unsanitized**. Sounds crazy, doesn't it? However, they have a magical technique that prevents *any* JavaScript code from running. Period.

Q1.1 Discuss what an attacker could do to snoop on another user's messages. What specially crafted messages could Mallory have sent to steal this user's account verification code?

Q1.2 Keeping in mind the attack you constructed in the previous part, what is a defense that can prevent against it?

Question 2 ***Second-order linear... err I mean SQL injection*** ()

Alice likes to use a startup, NotAmazon, to do her online shopping. Whenever she adds an item to her cart, a POST request containing the field item is made. On receiving such a request, NotAmazon executes the following statement:

```
cart_add := fmt.Sprintf("INSERT INTO cart (session, item) " +
                            "VALUES ('%s', '%s')", sessionToken, item)
db.Exec(cart_add)
```

Each item in the cart is stored as a separate row in the cart table.

Q2.1 Alice is in desperate need of some pancake mix, but the website blocks her from adding more than 72 bags to her cart . Describe a POST request she can make to cause the cart_add statement to add 100 bags of pancake mix to her cart.

When a user visits their cart, NotAmazon populates the webpage with links to the items. If a user only has one item in their cart, NotAmazon optimizes the query (avoiding joins) by doing the following:

```
cart_query := fmt.Sprintf("SELECT item FROM cart " +
                             "WHERE session='%s' LIMIT 1", sessionToken)
item := db.Query(cart_query)
link_query = fmt.Sprintf("SELECT link FROM items WHERE item='%s'", item)
db.Query(link_query)
```

After part(a), Alice recognizes a great business opportunity and begins reselling all of NotAmazon's pancake mix at inflated prices. In a panic, NotAmazon fixes the vulnerability by parameterizing the cart_add statement.

Q2.2 Alice claims that parameterizing the cart_add statement won't stop her pancake mix trafficking empire. Describe how she can still add 100 bags of pancake mix to her cart. Assume that NotAmazon checks that sessionToken is valid before executing any queries involving it.

Question 3 ***CalCentral Security*** **()**

Given your performance as a skilled attacker, university administrators have asked you to assess the security of the CalCentral platform.

The CalCentral website is set up as follows:

- CalCentral is located at `https://calcentral.berkeley.edu/`.
- The Central Authentication Service (CAS) is located at `https://auth.berkeley.edu/`.
- CalCentral uses session tokens stored in cookies for authentication, similar to Project 3. The session token cookie has domain `berkeley.edu`, and the `Secure` and `HttpOnly` flags are set.
- CalCentral does **not** use CSRF tokens or any form of CSRF protection.

Each subpart is independent.

Q3.1 (3 points) You find a reflected XSS vulnerability on CAS. `https://berkeley.edu` has a footnote that says "UC Berkeley."

TRUE OR FALSE: Using this vulnerability, you can cause the victim to see "CS 161 Enterprises" in the footnote when they visit `https://berkeley.edu`.

○ True, because the script runs with the same origin as `https://berkeley.edu`.

○ True, because XSS subverts the same-origin policy.

○ False, because the script runs with a different origin from `https://berkeley.edu`.

○ False, because the script only affects the browser's local copy of the site.

Q3.2 (3 points) You find a stored XSS vulnerability on CalCentral.

TRUE OR FALSE: Using this vulnerability, you can cause the victim to load CalCentral with the "My Academics" button changed to link to `https://evil.com/`.

○ True, because Javascript on a page can change that page's HTML

○ True, because CalCentral does not implement CSRF tokens.

○ False, because Javascript on a page cannot change that page's HTML

○ False, because `https://evil.com` has a different origin from CalCentral

Q3.3 (5 points) You try searching for `<script>alert(1);</script>` on `https://calcentral.berkeley.edu/search/`, and you see a pop-up.

Select all domains where you'd be able to leak at least **some** cookies set by that domain, assuming the appropriate cookies exist.

□ `https://evil.edu/`

□ `https://berkeley.edu/`

□ `https://auth.berkeley.edu/`

□ `https://evil.calcentral.berkeley.edu/`

□ `http://calcentral.berkeley.edu/`

□ None of the Above