

Question 1 *CSRF++*

0

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.

Q1.1 What kind of attack is the welcome page vulnerable to? Provide the name of the category of attack.

Q1.2 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?

Q1.3 `patsy-bank.com` sets `SameSite=strict` for all of its cookies. Does this stop the attack from part (b)? Assume the welcome page does not require a user to be logged in.

Q1.4 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.

Question 2 *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 `important!!!`

Steam: Thank you for your purchase

``

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.

- Q2.1 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?
- Q2.2 Keeping in mind the attack you constructed in the previous part, what is a defense that can prevent against it?

Question 3 *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.

Q3.1 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.

Q3.2 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.

The following questions are optional

Question 4 Phishing

()

A phishing attacker tries to gain sensitive user information by tricking users into going to a fake version of a website they trust. The attacker might convince the user to go to what *appears* to be their bank and to enter their username and password.

i. What are some ways that attackers try to fool users about the site they are going to? How do they convince people to click on links to sites?

ii. What are some defenses you should employ against phishing?

Question 5 *Clickjacking*

0

In this question we'll investigate some of the click-jacking methods that have been used to target smartphone users.

Q5.1 In many smartphone browsers, the address bar containing the page's URL can be hidden when the user scrolls. What types of problems can this cause?

Q5.2 Smartphone users are used to notifications popping up over their browsers as texts and calls arrive. How can attackers use this to their advantage?

Q5.3 QR codes haven't taken off and become ubiquitous like some thought they would. Can you think of any security reasons why this might be the case?