Paxson
Spring 2017

CS 161 Computer Security

Discussion 4

Week of February 13, 2017

Instructions. We will break into groups to discuss the following questions. Please think of as many solutions as you can. Be original! Maybe you will come up with something no one has thought of yet. Be prepared to talk about your solutions with the rest of the section.

Question 1 ***Clickjacking*** **(5 min)**

Watch the following video: https://www.youtube.com/watch?v=sw8CH-M3n8M

Question 2 ***Session management*** **(25 min)**

Let's design our own session management system, in order to visit the many considerations that go into making one. Suppose we're building a website where users can do online banking. You've probably seen what real banks do, but disregard that and see if we come up with something different.

We'll start with a few simple requirements:

1. Users have established a username and password with the bank.
2. Users can see their own past transactions.
3. Users can transfer an amount of money to a different user.

(a) Review: In web security, what are *cookies*? How do they come into existence, and how are they used?

(b) Suppose HTTP didn't support cookies (and there is no other client-side state for you web development buffs). How can we let a user see their own past transactions and transfer money to other users? What form(s) might we have on our website? Discuss the properties of your design.

(c) Now suppose we can use cookies in our design. Here's a straightforward usage: a user submits a form with their username and password, and responds by setting a cookie with the username and password.

Update your design accordingly. Has this improved anything? What might be some drawbacks?

(d) Let's evaluate some specific properties of this design:

- If a cookie is stolen, what happens?
- What would happen if a user changes their password?
- Can a user make sessions on other computers log out?

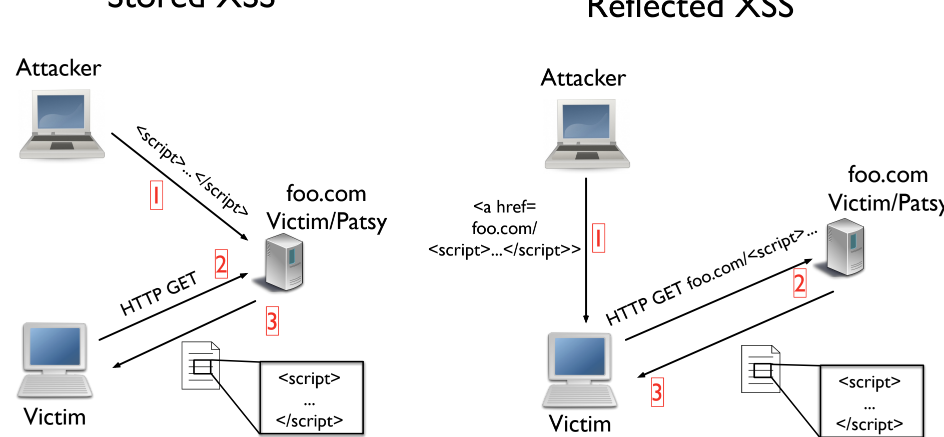
(e) To our requirements, we'll add several softer desiderata:

4. It should be convenient for users.
5. Minimize the impact of having a cookie stolen.
6. We should be able to decide how long a user can stay logged in for, even if they maliciously keep cookies past their max age.
7. Users should be able to log out from other computers remotely.
8. It shouldn't take a lot of memory in our database.
9. An attacker that gains access to a victim's computer shouldn't be able to transfer money.

Take a few minutes to come up with another design. See how many of these goals you can meet.

Question 3 ***Cross-Site Scripting (XSS)*** **(15 min)**

The figure below shows the two different types of XSS.



As part of your daily routine, you are browsing through the news and status updates of your friends on the social network FaceChat.

(a) While looking for a particular friend, you notice that the text you entered in the search string is displayed in the result page. Next to you sits a suspicious looking student with a black hat who asks you to try queries such as

```
<script>alert(42);</script>
```

in the search field. What is this student trying to test?

(b) The student also asks you to post the code snippet to the wall of one of your friends. How is this test different from part (a)?

(c) The student is delighted to see that your browser spawns a JavaScript pop-up in both cases. What are the security implications of this observation? Write down an example of a malicious URL that would exploit the vulnerability in part (a).

(d) Why does an attacker even need to bother with XSS? Wouldn't it be much easier to just create a malicious page with a script that steals *all* cookies of *all* pages from

the user's browser?

(e) FaceChat finds out about this vulnerability and releases a patch. You find out that they fixed the problem by removing all instances of `<script>` and `</script>`. Why is this approach not sufficient to stop XSS attacks? What's a better way to fix XSS vulnerabilities?

Question 4 *Cross-site not scripting* (5 min)

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:

```
<pre>
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">
</pre>
```

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

Users can send **arbitrary HTML code** that 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.

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?