Song
Spring 2015

CS 161
Computer Security

Discussion 8

March 10 & 11, 2015

Question 1 ***The DOM*** **(10 min)**

You will need to know a little bit about the Document Object Model (DOM) for this class. Pages on the web are structured trees with nodes that display the information we see when we visit a webpage. Javascript allows us to interact with it in browsers. Here's a very simple webpage:

```
<html>
  <body>
    <p>Hello World</p>
    <img src="doge.com/doge.jpg">
    <script>
        alert("Hello!");
    </script>
    <script src="ads.com/doubleclick.js"></script>
  </body>
</html>
```

Question 2 ***The SOP*** **(10 min)**

Let's get back to security. Notice that cross-site requests like getting the image from doge.com are all over the web. How does my browser prevent the owner of doge.com from sending some malicious file in place of the image I request and ruining my entire website? The Same Origin Policy (SOP) helps browsers maintain a sandboxed model by preventing certain webpages from accessing others. Two resources (can be images, scripts, HTML, etc.) have the same origin if they have the same protocol, port, and host. As an example, the URL `http://inst.berkeley.edu/eecs` has the protocol HTTP, its port is implicitly 80, the default for HTTP, and the host is inst.berkeley.edu.

Fill in the table below indicating whether the webpages shown can be accessed by `http://amazon.com/store/item/83`. Note: by "access," we generally mean that browsers disallow reading, but often still allow writing and embedding!

Origin	Can Access?	Reason if not
http://store.amazon.com/item/83		
http://amazon.com/user/56		
https://amazon.com/store/item/345		
http://amazon.com:2000/store		
http://amazin.com/store		

Solution:

Origin	Can Access?	Reason if not
http://store.amazon.com/item/83	No	different host
http://amazon.com/user/56	Yes	
https://amazon.com/store/item/345	No	different protocol
http://amazon.com:2000/store	No	different port
http://amazin.com/store	No	different host

Question 3 ***Cookies and Other Food for Thought*** **(10 min)**

In this question we'll consider some loopholes that attackers can manipulate.

1. An iframe can be loaded transparently on top of other elements in a page. Assuming an attacker can get users to visit a malicious site, how can they get them to like their Facebook page? How would you prevent this?
2. Cookies are often random secret strings used to store the fact that a user authenticated recently in their browser. Then, when the user requests the page again, the browser sends the cookie and the server knows the user is authenticated. How could this be a problem? How would you fix it?
3. What is the origin of a script like the example in the first section of the worksheet loaded from ads.com? How can an attacker use this to their advantage if they can somehow get their scripts on that page?

Solution:

1. You can position the transparent iframe on top of some flashy button saying something like "Click here to claim your $10000!" When users click the button, they will actually be clicking the hidden "like" button in the iframe. There are various ways to fix this, but generally you want to make sure your webpage isn't loaded in the background like that with "framebusting"; for example, site makers used to use "top.location = location". It's a hack, like much of web security, so this is a good talking point.
2. A malicious site could cause the user to go to a URL like "wellsfargo.com/send?amt=80&to=atta If they're authenticated, the request will go through. Use CSRF to protect against.
3. The script is actually loaded with the origin of the page that requests it. Otherwise, scripts loaded from other sites would be pretty useless! Can use this to introduce them to XSS if you have time.

A final note: do not hesitate to ask for help! Our office hours exist to help you. Please visit us if you have any questions or doubts about the material.