Alin Tomescu, CSE408
Tuesday, May 3rd, Lecture #24

Web security

Web security

Client has bank, Slashdot.org and Facebook with Farmville open on its browser, and they are all running JavaScript code.

HTTP is a stateless protocol. When you load a page or an image your browser sends a request (GET) and the server sends back a response. Server closes the TCP connection and forgets you ever existed.

Cookies

There is a database of cookies on the client and each cookie has a domain, a name and a value.

Example: $(bank.com, userid, rob)$

The browser will send the contents of the *bank.com* cookie when it makes the request to *bank.com*. Most importantly, it will only send the *bank.com* cookie and not other cookies from other websites.

The bank can then take the cookie and see that it's Rob who did the request, so it'll display his page.

This is screwed up because…

- easy to forge (fake) a cookie
- doesn't have any timeouts
- the cookie is sent in clear, so it's subject to man in the middle attacks.

Browser	**Facebook**	The Facebook server has a database with session ID's and their associated information: - user name - expiration date - maybe IP address (if you're all behind a NAT it doesn't help that much)
$SSL(GET\ login.html)$ →		
	← $SSL(HTTP\ response\ with\ HTML\ code)$	
User types in username and password →		
$SSL(POST\ user = \cdots, pass = \cdots)$→		
	←$SSL(SetCookie: (session\ id = 123456)$	
User now request a page on Facebook: $GET\ /\ Cookie: session_{id} = 123456$ → - this needs to be encrypted with SSL to avoid MITM attack stealing the cookie		
	←$SSL(HTTP\ response\ with\ HTML\ code)$	

A program like Firesheep will sniff wireless networks, looking for packets going to Facebook or Gmail and grabbing the cookies. In the end it will present you with a nice friendly list of websites and accounts for you to log in as another person.

Cookies have a feature (a field) that specifies they should only be sent over SSL. This would solve the Firesheep problem.

Session IDs need to be big random numbers so that an attacker cannot ever guess a valid session ID for anyone.

The cookie database has Session ID, username, lifetime, and a lot of other data. Instead of storing this into the server's database you can put it all in the cookie, signed by the server using a MAC. In fact you don't even need the session ID. So the server will only store the key k.

Alin Tomescu, CSE408
Tuesday, May 3rd, Lecture #24

JavaScript and the same origin policy

JavaScript code runs on your browser and it can access and set your cookies. You wouldn't want Slashdot to modify the bank website with JavaScript or get its cookies. That's why the **same origin policy** was created. Has been around for a decade.

Same origin policy:

- An origin is a domain name like *amazon.com*
  - What about *www.amazon.com*? Is that the same as *amazon.com*? Browser's have different rules for making educated guesses if these two are the same. Usually, JavaScript code running on *cs.sunysb.edu* can mess around with **.cs.sunysb.edu* stuff.
  - What about country codes?
  - Turns out a webpage contains a white-list of the same origin domains (equivalent domains).
- There are two types of resources in JavaScript
  - HTML
  - Cookies
  - Network
- The rule is that JavaScript code from domain d can access resources from domain d only.
  - For instance, from your bank website you can only open a network connection to your own bank

Ways to get around the same origin policy?

- cross site request forgeries attacks (CSRF)
- cross site scripting attacks (XSS)

Cross site request forgeries attacks (CSRF)

If I know what the link to add someone as your friend looks like on Facebook (Add as a friend) then I can send you an email that says to click on this link (under an excuse) If you're logged into Facebook, clicking on that link will add the friend without you realizing it.

Ambient security: there is no connection between the action and the authority.

Facebook can stick an extra token after the ID in the link Add as a friend and then validate the token. This token is associated with the user for which the link was generated. So the user cannot post this link to someone else. Because when they click on it and request the resource the token check will fail, since the line to the resource was not generated for them.

Another fix is to look at the referrer field. When you send a request, you tell the website you're going to where you're coming from. If you're going from Wikipedia to Amazon, the referrer is Wikipedia.