Alin Tomescu, CSE408
Tuesday, May 10th, Lecture #26

Mashup security

A mashup is a web application that combines data and possible code for multiple websites. **Example:** *housingmaps.com*

Ways to integrate the services

You've got the user's browser, which talks to the mashup website server, which talks to individual web servers like craigslist.com and maps.google.com.

- as far as craigslist.com and Google Maps are concerned, their client is the mashup.com web server

Imagine a *mashup.com* website that draws your Facebook friends on a Google Map. It needs some way to log into Facebook. Facebook can give mashup.com the ability to look at your contacts and only that.

Another way to organize a mashup, is you have a browser and it loads *mashup.com* and then inside that page it has an *iframe* (inline frame where you can load a different page). JavaScript within the *mashup.com* webpage cannot modify the webpage in the iframe (because of the same origin policy). Same thing holds for the website in the iframe with respect to Mashup.com

iGoogle and gadgets are one example of mashups with iframes that load little snippets from other providers. For instance, the Google webpage loads the current weather in an iframe.

Another way of having mashups is for the browser to load a page from *mashup.com* and in this page JavaScript code can be embedded as follows:

```
<script src="facebook.com/fb.js">
       JavaScript code from facebook.com
</script>
```

So you've got a script with code delivered from Facebook.com, rendered on the *mashup.com* website. The origin of the script is *mashup.com*. The downside is that you cannot access Facebook user credentials stored in cookies. Also, the mashup must trust the Facebook code not to do anything bad on the *mashup.com* website. This is analogous to a shared library on the OS, except in an OS you get your shared library from trusted sources and you only get them once (the Facebook JS has to be loaded every time the webpage is loaded).

JSON (JavaScript Object Notation), allows you to store data in *fb.js*. So *fb.js* might just be data and not code (for instance, *friendslist.js* generated on the fly by Facebook).

Message passing in HTML5

OS uses message passing to achieve cooperation between processes. HTML5 designers integrated similar message-passing functionality through a new API called postMessage.

The postMessage API allows one domain to send messages to another domain. For example, if you have a webpage loaded from *mashup.com* and it has an iframe in it, loaded from Facebook.com, then the outer *mashup.com* frame can do a postMessage to the iframe and send a request:

```
element = doc.getElementById(iframe_id);
element.postMessage("listfriends", "facebook.com");
```

The code in the Facebook iframe on mashup.com has a listener to respond to that message, such as handleMessage(Event e). Facebook should know who sends the message, so the origin would be stored in the event e.

handleMessage is running in the iframe so its origin is Facebook, which means it can access your Facebook cookies so it can access your friends through a HTTP request.

Conclusion

There's no really good solution for mashups.

OpenID and Facebook applications

The goal of OpenID is to have a single ID on the web that you can use across many services. It was called **single sign on for the web**.

Weird decision: your ID is a URL, ex: http://cse508.blogspot.com/

When you visit a webpage and they have a box for OpenID where you type in your URL and you click login. The protocol is complicated because they wanted everyone to be able to manage their IDs, to host their own OpenID provider and to easily move their IDs from one provider to another.

Brief description of how it works:

- load *amazon.com*
- type in your OpenID and hit login
- *amazon.com* sends you back a redirect (contains your OpenID and also where to return on *amazon.com*) to your OpenID provider
- you send the redirect info to your OpenID provider
- the OpenID provider sends you a password page, you log in
- if it likes your password it sends you a redirect to where to go, with an **authentication token**
- you go on *amazon.com* with the authentication token, and *amazon.com* checks it
    - Maybe it's signed by the OpenID provider
    - Maybe *amazon.com* can just confirm with the OpenID provider somehow

This is good because if you want to log in to another website in addition to *amazon.com*, you can now log in automatically (the OpenID provider stores a cookie).