HTML5 Cross-Domain Messaging

"Code with Passion!"

Topics

- Single Origin Policy (SOP)
- HTML5 Cross-domain messaging
- Cross-Origin Resource Sharing (CORS) & XHR2

Single Origin Policy

What is Same Origin Policy?

- The same origin policy does not allow JavaScript code loaded from one origin accessing or communicating with documents from another origin
  - In other words, documents retrieved from distinct origins are isolated from each other
- JavaScript codes are considered from the same origin only if they are loaded from the sites that have the same
  - protocol (http:, https:, ws:, wss:)
  - host
  - port
- The same origin policy is imposed by browsers

Same Origin Policy Examples

- The JavaScript code from http://store.company.com/dir/page.html can access the following documents since they are considered from same origin
  - http://store.company.com/dir2/other.html
  - http://store.company.com/dir/inner/another.html
- But it cannot access the following documents since they are considered from different origins
  - https://store.company.com/secure.html (different protocol)
  - http://store.company.com:81/dir/etc.html (different port)
  - http://news.company.com/dir/other.html (different host)

Why Same Origin Policy in the first place?

- In order to prevent Cross-site-scripting (XSS) security risk
- In XSS, attackers inject client-side script into Web pages viewed by other users

What is the downside of SOP?

- Same Origin Policy (SOP) makes it hard to do “mash-up”
  - Because it disallows XMLHttpRequest object from accessing documents from different origins
  - What if I want the JavaScript code loaded from server A to access data from Flickr, Google, Yahoo, etc through Ajax call
- SOP based security model of HTML
  - Introduced in Netscape Navigator 2.0
  - Is becoming a hindrance of writing Rich client application
- Work-arounds (not a desirable solutions, however)
  - The client asks the server to access the document on behalf of it – not efficient
  - JSONP – security vulnerability

Issues of SOP dealt with in HTML5

- Issue #1
  - A document running in a window A (or iframe A) cannot access a document running in window B (or iframe B)
  - HTML5 solves this through “Cross-Domain Messaging”
- Issue #2
  - A document loaded from origin A cannot access service running in Origin B through traditional XHR
  - HTML5-enabled browsers solves this by supporting Cross-Origin Resource Sharing (CORS) and CORS-enabled XMLHttpRequest2 (XHR2)

HTML5 Cross-Domain Messaging

What is & Why Cross-Domain Messaging?

- Before Cross-Domain Messaging, communications between iframes, windows, tabs are disallowed by browser
  - In order to prevent XSS

- Cross-Domain Messaging enables secure cross-domain messaging across iframes, windows, tabs regardless of origins of the documents they loaded
  - These iframes, windows, tabs can send/receive data to/from other iframes, windows, and tabs of different origins

Posting a Message to another iframe (from Host window)

```
<body>
 <p> The source origin of this page is http://abc.domain1.com</p>
 <script>
      aframe = document.getElementById('iframe');

      // The source origin (of this page) gets constructed by the browser
      // and sent along to the target origin.

      // The target origin needs to be specified as a second argument
      aFrame.postMessage(
            “Hello world!”,                    // Message to post
            "http://def.domain2.com"           // Target Origin
       );
 </script>

 <p>Target iframe:</p>
 <iframe id="iframe" src="http://def.domain2.com/my_iframe.html"></iframe>

 </body>
```

Receiving a Message from the iframe (in Host window)

- Add “onmessage” event handler
- The event handler receives “event”, which contains “data”, “origin”, and “source” properties
- You need to check the “origin” to make sure the message is from the trusted origin or origins

```
<script>
    // Handle message received from the iframe
    window.addEventListener('message', function(e) {
      if (e.origin !== "http://def.domain2.com" ) { // filter origin for security reasons
         // Disregard the message since it is not from a valid origin
      } else {
         // e.data contains message from the sender
      }
    }, false);
</script>
```

Lab:

Exercise 1: Cross-Domain Messaging
1236_html5_messaging.zip

Cross-Origin Resource Sharing (CORS) & XHR2

What is CORS?

- W3C Working Draft that defines how the browser and server must communicate when accessing sources across origins
  - Currently supported by most browsers - Internet Explorer 8+, Firefox 3.5+, Safari 4+, and Chrome
- The basic idea behind CORS is to use custom HTTP headers to allow both the browser and the server to know enough about each other to determine if the request or response should succeed or fail
  - This is to address the limitations of the JSONP
- Used when the server doesn't require cookie or session-based authentication to expose data for universal access

CORS Scheme

- The HTTP request is sent with an extra header called *Origin*
  - The *Origin* header contains the origin (protocol, domain name, and port) of the requesting page so that the server can easily determine whether or not it should serve the response
  - *Origin: http://abc.domain1.com:8787*
- If the server decides that the request should be allowed, it sends a *Access-Control-Allow-Origin* header echoing back the same origin that was sent or "*" if it's a public resource
  - If this header is missing, or the origins don't match, then the browser disallows the request
  - *Access-Control-Allow-Origin: http://abc.domain1.com:8787*
  - *Access-Control-Allow-Origin: **

XMLHttpRequest2 (XHR2)

- Modern browsers create CORS-aware XHR object (sometimes called XHR2) with the *Origin* header automatically
  - If the CORS-enabled server sends back with *Access-Control-Allow-Origin* header with proper value, then the browser will allows cross-domain access

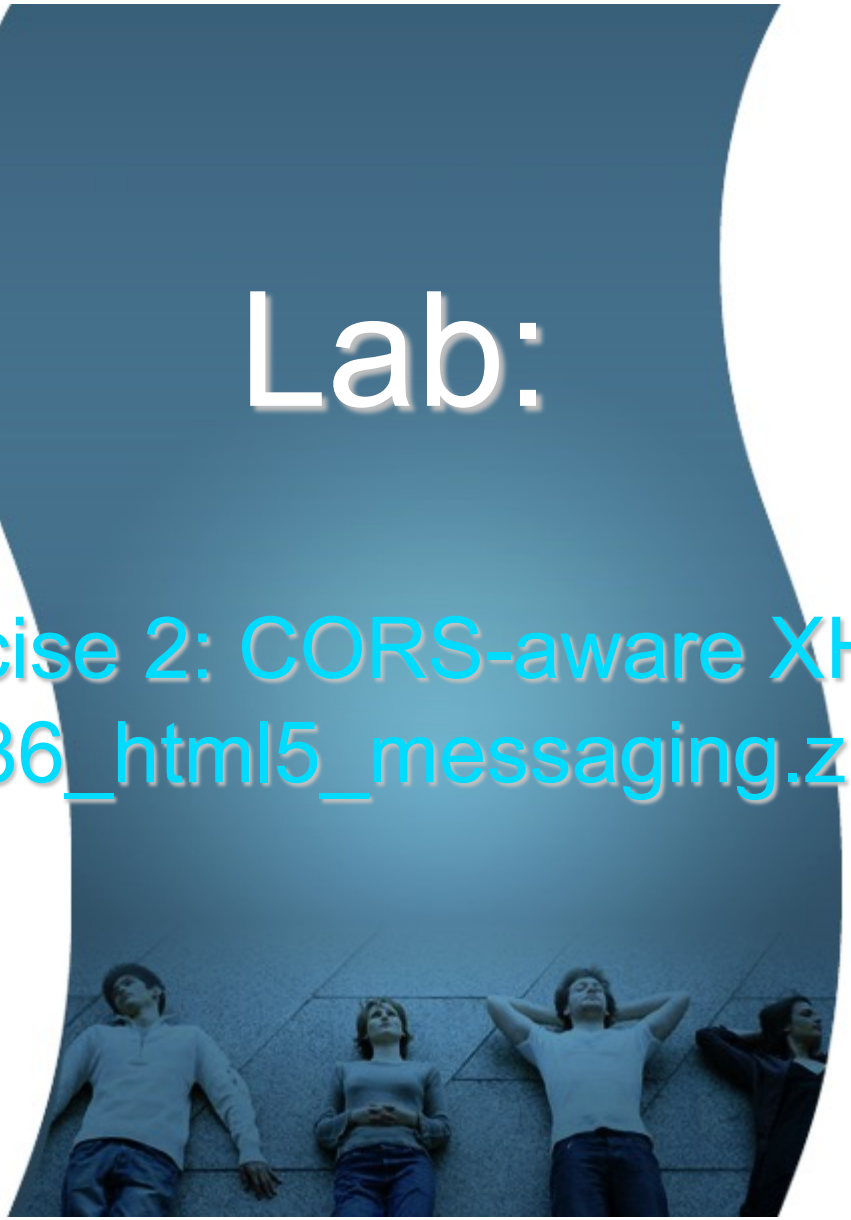
```
var xhr = new XMLHttpRequest();
xhr.open("get", "http://abc.domain1.com:8787/some_resource/", true);
xhr.onload = function(){  //instead of onreadystatechange
   //do something
};
xhr.send(null);
```

Issues of CORS

- CORS is not yet available in old browsers (of course!)
- There are many existing services that don't yet take advantage of this specification

Lab:

Exercise 2: CORS-aware XHR2
1236_html5_messaging.zip



Code with Passion!