aura
INFORMATION SECURITY

HTML 5

<!DOCTYPE html>

Who is this guy?

Mike Haworth

Aura Information Security

mike@aurainfosec.com

HTML 5

- Isn't it just markup?
  - Nope, Sites now have greater access to each other's content via new APIs
- HTML 5 features are deployed in major browsers
  - Therefore HTML 5 issues will affect you even if you aren't building a fancy pants HTML 5 site

Features

aura INFORMATION SECURITY

- Video/Audio tags
- ~~Drag and Drop~~
- ~~Webworkers~~
- LocalStorage
- History API
- Cross window messaging
- WebSockets
- Cross Origin Resource Sharing
- Content Security Policy
- ~~iframe sandboxing~~
- ~~WebGL~~

New Elements and Attrs

- New tags like <audio>, <video> & <canvas>
- Forms now have autofocus attribute
  - Gets fired on pageload
- List of new elements

  http://www.w3schools.com/html5/html5_new_elements.asp

New Elements – gotchas

- New tags means new XSS vectors
  - <input autofocus onfocus="alert(1)">
  - <video><source onerror="javascript:alert(1)">
- A whitelisting approach will still protect.
  - e.g. Drupal's filter_xss()
  - XSS vectors: http://html5sec.org/

Cross Window Messaging

aura
INFORMATION SECURITY

- What's it for? Facebook connect etc.
- window.postMessage()
- Requires you explicitly receive a message
- Need a reference to the window your sending to

```
// sending
win =
document.getElementById("iframe").contentWindow;
win.postMessage('my msg', "http://recv.com");
```

Cross Window Messaging


```
// receiving
window.addEventListener("message", function(e){
        if ( e.origin !== "http://sender.com" )
                return;
        document.getElementById("test").textContent
           = e.origin + " said: " + e.data;
```

Cross Window Messaging

- In the receiver, always check the origin
- Don't trust contents of postMessage() the data property
- Avoid XSS, e.g Use textContent not innerHTML
- Demo – to show why

DOM Storage

aura INFORMATION SECURITY

- localStorage and sessionStorage
- Key/Value pairs that cache application state
- sessionStorage available for the lifetime of tab or child tab
- localStorage available until manually cleared

- localStorage.setItem('mobile', '021111222');
- mobile = localStorage.getItem('mobile');

Storage – gotchas

aura INFORMATION SECURITY

- sessionStorage for session cookie?

- Obviously sessionStorage is accessable by JS
  - Therefore session token theft by XSS is possible.
  - Cookie w/ HTTPonly flag set is not accessible by JS
- **Use Cookies with HTTPOnly for session tokens**
- **Don't store sensitive info in browser storage**

CORS -- Cross Origin Resource Sharing

- Relaxing the same-origin policy for AJAX calls
- So now JS running in mysite can make AJAX GETs AND POSTs to yoursite.

CORS - simple & preflight

- There are 2 types of CORS
- Simple:
  - Simple request / response
  - GET or POST
  - No custom headers
- Preflighted:
  - Initial "preflight" request with OPTIONS method
  - If response was a 200 do actual request

CORS - preflight example

Browser visits evil.com which make request to.. **server on vuln.com**

OPTIONS /resources/post-here/ HTTP/1.1
Origin: http://evil.com
Access-Control-Request-Method: POST
Access-Control-Request-Headers: X-PINGOTHER

HTTP/1.1 200 OK
Access-Control-Allow-Origin: http://evil.com
Access-Control-Allow-Methods: POST, GET, OPTIONS
Access-Control-Allow-Headers: X-PINGOTHER
Access-Control-Max-Age: 1728000

POST /resources/post-here/ HTTP/1.1
...
Content-Type: application/xml; charset=UTF-8
X-PINGOTHER: pingpong

CORS can send Creds

- Script on site A sends a POST to site B with site B's session cookie.
- xhr.withCredentials = "true";
- What could possibly go wrong?

CORS abuse, file upload

Can use XMLHTTPRequest to do file uploads

```
function fileUpload(url, fileData, fileName) {
  var fileSize = fileData.length,
    boundary = "xxxxxxxxx",
    xhr = new XMLHttpRequest();

  // simulate a file MIME POST request.
  xhr.open("POST", url, true);
  xhr.setRequestHeader("Content-Type", "multipart/form-data, boundary="+boundary);
  xhr.setRequestHeader("Content-Length", fileSize);
  xhr.withCredentials = "true";

  var body = "--" + boundary + "\r\n";
  body += "Content-Disposition: form-data; name='contents'; filename='" + fileName + "'\r\n";
  body += "Content-Type: application/octet-stream\r\n\r\n";
  body += fileData + "\r\n";
  body += "--" + boundary + "--";

  xhr.send(body);
  return true;
}
```

http://blog.kotowicz.net/2011/05/cross-domain-arbitrary-file-upload.html

Cross Domain Upload

- We can do AJAX cross domain posts, now with credentials.
- We can upload file to victim site (where we are logged in) from a malicious site.
- Does NOT require victim server to have set header: "Access-Control-Allow-Origin: *"

DEMO

aura INFORMATION SECURITY

(Works in FF4/5 and Chrome)

NB: Upload could run *without* user interaction

Source:

http://j.mp/cors-file-upload

CORS – file upload

- Is there anything really new here?
  - Not from a defensive point of view, CSRF (e.g. form tokens) protections will prevent this
- If a custom header is set in the POST, then the request is 'pre-flighted', checking the Access-Control-Allow-Origin header on the receiving server.

XMLHttpRequest cannot load http://vuln.com/recv.php. Origin http://evil.com is not allowed by Access-Control-Allow-Origin.

CORS – defenses

- Do not use Origin header in access control decisions – it could be faked.

```
if ($_SERVER['HTTP_ORIGIN'] == 'nice.com' {
  echo $juicy_infos;
} ^-- bad idea
```

- Universal allow is bad – i.e. don't set:

```
header('Access-Control-Allow-Origin: *');
```

- Make sure access-control-max-age doesn't allow caching for too long eg. days/weeks

History API

- Change URL in the browser without a pageload
- Can't change domain
- history.pushState(state, title, url);
- Back button fires history.popState();

History – gotchas

- XSS can now change URL (client side)
- Unlike doc.location no request sent to server
- Make forged pages look more authentic :)
  - XSS in site.com/?p=<script>alert(1)</script>
  - Change URL to site.com/login
- Yet another way to spoof a referrer header
  - Don't trust URL or referrer (still)

Content Security Policy

- Prevents injected inline JavaScript
- Only load JS from whitelisted/trusted domains

- Only run scripts that are loaded from same domain as the page.
  - Use: X-Content-Security-Policy: allow 'self'

Content Security Policy

Set the header, now inline js NOT exec'd

```
<?php
header("X-Content-Security-Policy: allow, 'self'");
echo "
<html><head></head><body><h1>page</h1>
<script>alert(1);</script>
<script src="jquery.js" type="text/javascript>
</body>
</html>";
```

Content Security Policy

- Can also whitelist image and script source domains
- Can be run in 'report only' mode

- In use today e.g. lastpass.com
- Works in FF4/5 **but not Chrome 12**
- *Chromium 13* respects header: X-WebKit-CSP

- Policy examples:
  - https://developer.mozilla.org/en/Security/CSP/Using_Content_Security_Policy

WebSockets

aura INFORMATION SECURITY

- Server can push to client (no more polling)
- VNC, chat, collaborative editing etc.

- Works by in-channel 'upgrade'
- Client sends headers:
  - Connection: Upgrade
  - Upgrade: Websocket
  - Plus others

WS – Cache poisoning

aura INFORMATION SECURITY

Proxy misinterprets WS traffic as HTTP traffic

1. Attacker's client creates websocket traffic with attacker's server that looks like a legitimate HTTP request/response for popular JS file.
2. Traffic has a forged host header
3. Some transparent proxies cache based on "Host" header.
4. Result is proxy caches attackers copy of js file, which is served to everyone else.
5. Attacker wins

WebSockets – not in FF4

- WebSockets support removed from FF4, cache poisoning issue, will be back in FF6.
- Chrome uses a CONNECT request in the upgrade handshake to avoid this.
- Attack description:

  http://j.mp/ws-cache-poison

Summary

- More app logic is being pushed to the client
  - Increased browser attack surface
  - Therefore XSS & CSRF is of higher value
- The good news is Content Security Policy makes XSS harder (if you use Firefox).