CSC 482/582: Computer Security

Web Browser Security
Topics

1. HTML
2. JavaScript, JSON, and the DOM
3. Same Origin Policy (SOP)
4. XHR and Ajax
5. Cross-Origin Resource Sharing (CORS)
6. Extensions and Plug-ins
7. Browser Fingerprinting
Web Browser Context Model
HTML

- Hierarchical tree structure of tags.
- Tags have optional name=value parameters.
- Text nodes may exist between tags.
- Special characters: `< > “ ‘ &`
# Entity Encoding

<table>
<thead>
<tr>
<th>Entity</th>
<th>Character</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;</td>
<td>&lt;</td>
</tr>
<tr>
<td>&gt;</td>
<td>&gt;</td>
</tr>
<tr>
<td>&amp;</td>
<td>&amp;</td>
</tr>
<tr>
<td>&quot;</td>
<td>“</td>
</tr>
<tr>
<td>'</td>
<td>‘</td>
</tr>
<tr>
<td>©</td>
<td>©</td>
</tr>
<tr>
<td>¶</td>
<td>¶</td>
</tr>
<tr>
<td>€</td>
<td>€</td>
</tr>
<tr>
<td>≈</td>
<td>≈</td>
</tr>
<tr>
<td>½</td>
<td>½</td>
</tr>
<tr>
<td>&amp;#nnnn;</td>
<td>Unicode code point nnnn (decimal)</td>
</tr>
<tr>
<td>&amp;#xhhhh;</td>
<td>Unicode code point hhhh (hexadecimal)</td>
</tr>
</tbody>
</table>
HTML vs. XHTML

HTML

- Generously interprets tags, with many variants between browsers.
- Interprets string between certain tags as non-HTML text: `<style>`, `<script>`, `<textarea>`, `<xmp>`.

XHTML

- Strict: tags are case sensitive; all tags must be closed and properly nested; attributes must be quoted; etc.
- Supports raw text inside any tag via `<![CDATA[ ... ]]>`
- Can incorporate sections using other XML-based markup languages like MathML.
HTTP/HTML Integration

Why express HTTP headers in HTML?
- HTML document loaded from local file.
- HTML document received via non-HTTP protocol.

How to express HTTP headers in HTML?
- http-equiv meta tags
  
  <meta http-equiv="Content-Type" content="text/html;charset=utf-8">

Dangers of HTTP/HTML integration
- Undefined behavior when meta tags conflict with each other or with HTTP headers.
- Browser has already made some decisions about how to process document. Can’t change content type from HTML or change location to load content from.
1: IE will allow a NUL to be inserted.
2, 4: Can replace space with vertical tab (oxoB) or form feed (oxoC) or UTF-8 nb space (oxAo) in Opera.
2: Whitespace can be replaced by /.
3: NULs or whitespace may be inserted.
5: IE accepts backtick (``) as well as quotes.
6: Whitespace after quotes can be skipped.
HTML Obfuscation Techniques

1;�新年快乐!x\r
/onbegin=\u007b\u5411\u544a\u6709\u4e0d\u5355\u5343\u7684\u4e0d\u6d77\u52a8\u65b0\u5e76\u4e0a\u6307\u5b9a\u70b9\u6709\u5355\u5343\u7684\u4e0d\u6d77\u52a8\u65b0\u5e76\u4e0a\u6307\u5b9a\u70b9\}

- Fake invalid namespaces
- Invalid but working attribute separators
- Decimal and hex entities inside HTML attributes
- CSS entities inside the style attribute
- Double encoded entities inside the style attribute
- Backticks as attribute value delimiters
- Invalid but working escapings
- JavaScript Unicode entities in onbegin event handler
- Crippled decimal entities inside onbegin event handler
- Invalid garbage before the ending tag

Snippet above runs JavaScript using following obfuscation techniques.
HTML Input Validation

1. Don’t accept HTML input.
   • The best approach if you can choose it.
2. Whitelist validation with a parser.
   1. Use HTML (or XML) parser to create in-memory representation of input string.
   2. Remove all unknown or undesired tags, attributes, and values.
   3. Serialize in-memory data structure to a well-formed correctly escaped HTML document.
HTML Forms

<form>  tag
- action=URL destination for form input.
- method=get sends input as query string parameters
- method=post sends input as data in POST method

<input>  tag
- name=name of input.
- type attribute specifies checkbox, radio, text, etc.
<input type=“hidden” name=“user” value=“james”>

- Used to propagate data between HTTP requests since protocol is stateless.
- Clearly visible in HTML source.
- User can modify hidden values since form can be copied, modified to change hidden fields, then used to invoke script.
## HTTP POST Request

<table>
<thead>
<tr>
<th>Method</th>
<th>URL</th>
<th>Protocol Version</th>
<th>Headers</th>
</tr>
</thead>
<tbody>
<tr>
<td>POST</td>
<td><a href="http://www.example.com/">http://www.example.com/</a></td>
<td>HTTP/1.1</td>
<td>Host: <a href="http://www.example.com">www.example.com</a></td>
</tr>
<tr>
<td></td>
<td>HTTP/1.1</td>
<td></td>
<td>User-Agent: Mozilla/5.0 (Windows NT 5.1) Gecko/20060909 Firefox/1.5.0.7</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Accept: text/html, image/png, <em>/</em>/</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Accept-Language: en-us,en;q=0.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Blank Line</td>
</tr>
<tr>
<td></td>
<td>name=Jane+Doe&amp;sex=female&amp;color=green&amp;over6feet=true&amp;over200pounds=false&amp;athletic ability=NA</td>
<td></td>
<td>POST data</td>
</tr>
</tbody>
</table>
JavaScript

Common web scripting language.

- Standardized as ECMAScript (currently version 5).
- Runs in browser via a Just-In-Time (JIT) compiler.

Can be included in a web page via

- Inline <script> blocks.
- Remote scripts via <script src="...">
- javascript: URLs in HTML params and CSS.
- CSS expression(...) syntax
- Event handlers (onload, onclick, onerror, ...)
- Timers (setTimeout, setInterval)
- eval(...) calls from within JavaScript.
JavaScript Security Issues

Each <script> block is processed individually in the order encountered on page.

- Syntax error won’t stop later <script>s from running.
- All scripts can set variables in global namespace.
- Scripts can replace built-in classes and functions.

Nested script inclusion requires nested encoding

```html
<div onclick="setTimeout('do_stuff(\'user_string\'),1)">
1. HTML parser extracts onclick and puts in DOM.
2. When button clicked, timeout is set.
3. When timeout triggered, inside script executed.
```
To be secure, double-encode `user_string` with JS backslashes, then encode with HTML entities.
JSON

JSON = JavaScript Object Notation

- Lightweight data interchange format.
- Based on a subset of JavaScript, but is language independent; libraries for any language.
- Standards: RFC 4627 and ECMA-404.

JSON parsing

- Use JSON.parse(...)  
- Do not use eval(...) as it will execute any JavaScript code, not just parse JSON.
JSON Arrays

Arrays are lists of items

- Delimited by square brackets: []
- Items in array separated by commas
- Items can be of different data types

Array Examples

- [1, 2, 3]
- [“one”, “two”, “three”]
- [1, “two”, 3]
- [1, “two”, [1,2,3]]
JSON Objects

Objects are associative arrays

- Delimited by curly braces: {}
- Key/value pair syntax is “key” : value
- Pairs separated by commas
- Values can be objects, arrays, or scalar types.

Object Examples

- { “spam” : “eggs” }
- { “x” : 1, “y” : 2, “z” : 3 }
- { “hostname” : “kosh”, “ips” : [ “10.0.0.1”, “172.31.0.1”], “age” : 3 }
```json
{
    "firstName": "John",
    "lastName": "Smith",
    "age": 25,
    "address": {
        "streetAddress": "21 2nd Street",
        "city": "New York",
        "state": "NY",
        "postalCode": 10021
    },
    "phoneNumbers": [
        { "type": "home", "number": "212 555-1234" },
        { "type": "fax", "number": "646 555-4567" }
    ]
}
```
Document Object Model (DOM)

- DOM connects JavaScript and CSS to HTML documents.
- JavaScript can read and modify every element of HTML.
- Dynamic HTML (DHTML) = DOM + JavaScript + CSS.
- Capability used by threats in cross-site scripting attacks.
XMLHttpRequest (XHR) API

JavaScript API to request data from server.
- Without loading a new web page in browser.
- Can be done asynchronously so web application UI responsive during loads.
- Resources typically XML or JSON data.

 Allows highly interactive web applications
- AJAX = Asynchronous JavaScript and XML
- Examples: Google Maps, Gmail, etc.
- Can only request resources from server that JavaScript came from (Same Origin Policy.)
DHTML vs. Ajax
Browser Storage

- Why aren’t cookies enough?
  - Performance hit: included with every HTTP request.
  - Limited to about 4KB in size.
- Flash storage
  - Local Stored Objects (LSOs) 100KB per domain.
  - Client can request more storage with user approval.
- Web Storage (aka DOM Storage)
  - Standard supported by all browsers.
  - Key/value storage in string format.
  - 5MB of storage per origin.
- WebSQL exists but is not supported by IE or FF.
Same Origin Policy for DOM

**Policy:** Given any two JavaScript execution contexts, one should be able to access the DOM of the other only if protocols, DNS names, and port numbers of their documents match exactly.

- Cannot isolate home pages of different users.
- Disallows communication between login.example.com and payments.example.com.

<table>
<thead>
<tr>
<th>Originating document</th>
<th>Accessed document</th>
<th>Non-IE browser</th>
<th>Internet Explorer</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://example.com/a/">http://example.com/a/</a></td>
<td><a href="http://example.com/b/">http://example.com/b/</a></td>
<td>Access okay</td>
<td>Access okay</td>
</tr>
<tr>
<td><a href="http://example.com/">http://example.com/</a></td>
<td><a href="http://www.example.com/">http://www.example.com/</a></td>
<td>Host mismatch</td>
<td>Host mismatch</td>
</tr>
<tr>
<td><a href="http://example.com/">http://example.com/</a></td>
<td><a href="https://example.com/">https://example.com/</a></td>
<td>Protocol mismatch</td>
<td>Protocol mismatch</td>
</tr>
<tr>
<td><a href="http://example.com:81/">http://example.com:81/</a></td>
<td><a href="http://example.com/">http://example.com/</a></td>
<td>Port mismatch</td>
<td>Access okay</td>
</tr>
</tbody>
</table>
**document.domain**

- JavaScript property that permits two cooperating web sites that share a common TLD to agree to be considered same domain for SOP checks.

**Example:**
- On login.example.com and payments.example.com
- `document.domain = “example.com”`

<table>
<thead>
<tr>
<th>Originating document</th>
<th>Accessed document</th>
<th>Outcome</th>
</tr>
</thead>
<tbody>
<tr>
<td>URL</td>
<td><code>document.domain</code></td>
<td>URL</td>
</tr>
<tr>
<td><a href="http://www.example.com/">http://www.example.com/</a></td>
<td>example.com</td>
<td><a href="http://payments.example.com/">http://payments.example.com/</a></td>
</tr>
<tr>
<td><a href="http://www.example.com/">http://www.example.com/</a></td>
<td>example.com</td>
<td><a href="https://payments.example.com/">https://payments.example.com/</a></td>
</tr>
<tr>
<td><a href="http://payments.example.com/">http://payments.example.com/</a></td>
<td>example.com</td>
<td><a href="http://example.com/">http://example.com/</a></td>
</tr>
<tr>
<td><a href="http://www.example.com/">http://www.example.com/</a></td>
<td>(not set)</td>
<td><a href="http://www.example.com/">http://www.example.com/</a></td>
</tr>
</tbody>
</table>
**postMessage() API**

**postMessage() API**

- HTML 5 extension that permits secure communication between client scripts from different domains.

**Example**

- **On login.example.com (sender)**
  
  ```javascript
  parent.postMessage(key=val, 'http://payments.example.com)
  ```

- **On payments.example.com (receiver)**
  
  ```javascript
  addEventListener("message", info, false)
  If (msg.origin == "https://login.example.com") {
      // use msg.data that was sent by login.example.com
  }
  ```
Same Origin Policy for XHR

- XHR requests work like HTTP except
  - XHR URL must match origin of document
  - document.domain setting is ignored
- XHR limited on a per-browser basis on
  - HTTP methods (none allow TRACE)
  - HTTP headers (none allow Host, Referer, Content-Length)
CORS allows secure cross-domain requests.
- Simple: GET or POST text/plain, no custom headers
- Preflighted: Different request, body types + headers

Cookies are not sent by browser with either type.

Simple Request Mechanism
- HTTP request specifies its origin with a header:
  - Origin: URL
- If request is allowed, HTTP response is
  - Access-Control-Allow-Origin: URL or
  - Access-Control-Allow-Origin: *
- * is for public resources and ignores SOP entirely.
XHR+CORS Interaction

JavaScript Code → Browser → Server

xhr.send(); → preflight request (if necessary) → preflight response (if necessary) → actual request → actual response → Fire onload() or onerror()
CORS Preflight Mechanism

- Preflight OPTIONS HTTP request
  - Origin: URL
  - Access-Control-Request-Method: method
  - Access-Control-Request-Headers: optional header with a , separated list of custom headers being used.

- Preflight HTTP response
  - Access-Control-Allow-Origin: URL
  - Access-Control-Allow-Methods: , separated list of methods.
  - Access-Control-Allow-Headers: optional header with a , separated list of headers permitted by server.
  - Access-Control-Max-Age: time to cache preflight response
  - Access-Control-Allow-Credentials: true if want to permit authentication credentials to be sent.
Cookie and DOM SOP Interaction

Path scope used by cookie SOP, but not by DOM.

- JavaScript in same domain can overwrite cookies regardless of path scope.

An attack

- User browses to secure.example.com, which uses SESSIONID cookie for authentication.
- Attacker on test.example.com installs script.
- User browses to test.example.com, runs script.
- Script sends many cookies, overflowing cookie jar.
- Script sets SESSIONID token for *.example.com.
- User uses attacker SESSIONID on next access of secure.example.com.
Pseudo-URLS

- Allow inclusion of data directly in HTML pages.
  - about:, data:, and javascript: are pseudo-URLs.
  - Each browser treats pseudo-URL origins differently.

Uses of Pseudo-URLs

- about:blank is typically used to create blank DOM in iframes for scripts to control.
- `<img src="data:image/jpeg;base64,/9j/4AAQSk...">`
- `<iframe src="data:text/html;<h1>Hello world</h1>"`>
- `<iframe src="javascript:alert('Hello world')">`
Browser Security Architecture

- HTML5
- Silverlight
- Flash
- Plug-In
- Presentation
- JavaScript
- DOM/Events
- Parser/Threads
- Process & Logic
- WebSQL
- Storage
- XHR
- WebSocket
- Plug-in Sockets
- Network Services
- Same Origin Policy (SOP)
- Sandbox
- Core Policies
Browser Vulnerability History

Firefox

Chrome
Extensions and Plug-ins

- **Extensions**
  - Exist inside browser process.
  - Can create browser menus and tabs.
  - Can affect any and all web pages.

- **Plug-ins**
  - Can exist outside browser process.
  - Only affects page plugin is loaded into.
  - Loaded by MIME type or `<object>` tag.
  - Have their own security policies.

- **Add-on**
  - Umbrella term for plug-ins, extensions, themes, etc.
Extensions

Extensions run inside browser process.

- Typically written in JavaScript, HTML, XML.
- Has more privilege than a web page.
- Not restricted by SOP.
- Can read/write files, access login manager, ...
Extensions

Extensions are dangerous.

- Running an extension gives creator access to almost everything the browser can do.
- Can bypass efforts by applications to attempt to secure interactions with SSL, etc.

Extensions can have vulnerabilities

- Extensions that use user input, such as web page contents, need to validate that input carefully to avoid being controlled by attacker.
- XCS (cross-context scripting) is attack similar to XSS but since it occurs in extension, gives system access.
- Example: http://www.gnucitizen.org/blog/firebug-goes-evil/
Invoking a Plug-in

HTML invocation of a plug-in

```html
<object data="app.swf" type="application/x-shockwave-flash">
  <param name="param1" value="value1">
  <param name="param2" value="value2">
  ...
</object>
```

Browser processing of plug-in invocation

- If type specified, compare type with MIME types registered by all active plug-ins.
- If a match is found, then start the matching plug-in.
- If match not found, browser may check file suffix in data URL or Content-Type returned by server when that URL is fetched.
- If still no match, check body or ask user to install plug-in.
GIFAR Vulnerability

Graphics Interchange Format java Archive

- Has GIF header, so it’s a valid image.
- Has ZIP footer, so it’s a valid JAR.

Attacker uploads on image hosting site

- Victim downloads image.
- Content-type handling confusion transfers control to Java plug-in which runs file as a Java program, which
- Has access to victim cookie’s for image host.
Adobe Flash

Multimedia plugin with ActionScript
- Found on 95+% of all browsers.
- ActionScript is derived from JavaScript.

ActionScript has more permissions than JavaScript
- Full screen rendering for UI spoofing.
- Access to inputs like microphone, webcam.

Cross Domain Policy (as of Flash Player 7)
- Performs domain matching before allowing access.
- Can expand limits with crossdomain.xml policy file.
ActiveX

Executable code downloaded from server
- Native code binary format
- Can perform any action on client system.
- IE 9 disables Active X by default.

Security model
- Digital signature authentication
- Zone-based access control
- No control once execution starts
Java

Applet plugin

- Found on 80+% of all browsers.
- Deprecated `<applet>` tag superseded by `<object>`.

Java has more permissions than JavaScript

- Can open URL connections to any host at same IP, undoing isolation between virtual hosts.
- Can make TCP connections to any port.

Java security architecture

- Runs applets in a sandbox, optional code signing.
- Supports Flash crossdomain.xml files since 6u10.
- Digital signatures on applets required after 7u21.
Java Security Architecture

- **Bytecode Verifier**
  - Type checking.
  - Bounds checking.

- **Class Loader**
  - Enforces namespaces.

- **Security Manager**
  - Checks code signatures.
  - Enforces security policies.

Java 2 Security architecture and basic elements
Browser Fingerprinting

- Clients can identify specific browsers
  - HTTP headers, including User-Agent, and
  - Network addresses, and
  - System characteristics.
- System characteristics include
  - Browser plug-ins
  - Fonts
  - Screen resolution
  - Window sizes
  - Clock drift
  - RNG behavior
Panopticlick

How Unique — and Trackable — Is Your Browser?

Your browser fingerprint appears to be unique among the 2,524,318 tested so far.

Currently, we estimate that your browser has a fingerprint that conveys at least 21.27 bits of identifying information.

The measurements we used to obtain this result are listed below. You can read more about our methodology, statistical results, and some defenses against fingerprinting in this article.

Help us increase our sample size.

<table>
<thead>
<tr>
<th>Browser Characteristic</th>
<th>bits of identifying information</th>
<th>one in x browsers have this value</th>
</tr>
</thead>
<tbody>
<tr>
<td>User Agent</td>
<td>8.9</td>
<td>479.27</td>
</tr>
<tr>
<td>HTTP_ACCEPT Headers</td>
<td>8.61</td>
<td>391.56</td>
</tr>
</tbody>
</table>

value

Mozilla/5.0 (Windows NT 6.1; WOW64; rv:16.0) Gecko/20100101 Firefox/16.0
text/html, */* gzip, deflate en-US, en;q=0.5

Plugin 0: Adobe Acrobat; Adobe PDF Plug-In For Firefox and Netscape v9.5.2
application/pdf; Adobe PDF in XML Format; application/vnd.adobe.pdfxml; pdfxml)
(Adobe PDF in XML Format; application/vnd.adobe.x-mars; mars) (Acrobat Forms Data Format; application/vnd.fdf; fdf) (XML Version of Acrobat Forms Data Format; application/vnd.adobe.xpdf; xpdf) (Acrobat XML Data Package; application/vnd.adobe.xdp+xml; xdp) (Adobe FormFlow Data File; application/vnd.adobe.xdp+xml; xdp) Adobe Acrobat; Adobe PDF Plug-In For Firefox and Netscape 10.1.4
Private Browsing Modes

- Provide limited privacy
  - Disable browser history and web cache.
  - Disable storing cookies and LSOs to disk.
  - Designed to protect against other users of PC.

- Privacy exceptions
  - Browsers will store site configuration (pop-up blocking, SSL certificates, etc.)
  - Windows OS will have sites visited in DNS cache.
  - Servers can determine if private browsing is enabled by measuring time to write cookies.

- Browsers can still be fingerprinted
  - Most fingerprint features still enabled.
MPack Browser Malware

1. User visits site.
2. Response contains iframe.
3. Iframe code causes browser to make request.
4. Request redirected to MPack server.
5. Server identifies OS and browser, sends exploit that will work for client configuration.
6. Exploit causes browser to send request for code.
7. Mpack downloader sent to user, begins d/ling other malware.
MPack

Commercial underground PHP software

- Sold for $700-1000.
- Comes with one year technical support.
- Can purchase updated exploits for $50-150.

Infection Techniques

- Hacking into websites and adding iframes.
- Sending HTML mail with iframes.
- Typo-squatting domains.
- Use GoogleAds to draw traffic.
Key Points

1. HTML
   - HTML vs. XHTML parsing differences.
   - Input validation requires a whitelist approach with a parser.
2. JavaScript and the DOM
   - DHTML vs. XHR approaches to application design.
   - How and when JavaScript executes in a page.
3. Plug-ins and Extensions
4. Same Origin Policy (SOP)
   - Prevents web sites from accessing cookies, etc. from other sites.
   - CORS provides for safe cross-domain XHR.
   - Clickjacking, XSS, and CSRF attacks bypass SOP.
5. CSP and Sandboxing
References


7. Mark Pilgrim, Dive into HTML5, http://diveintohtml5.info/


