



IBM T. J. Watson Research Center

OpenAjax Security Work Session Topic Proposal

Naohiko Uramoto
Sachiko Yoshihama
Frederik De Keukelaere
IBM Research, Tokyo Research Laboratory

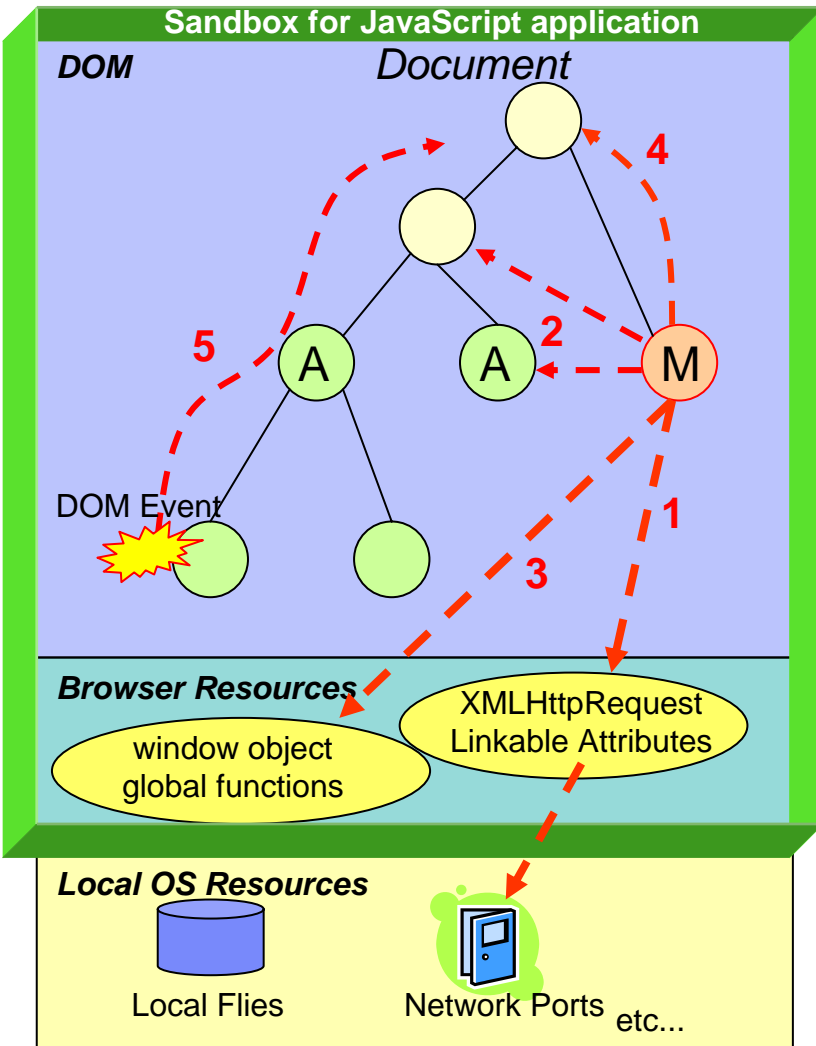


Proposed topics

- **Ajax security threat analysis**
- **DOM-based access control**
- **Alternative browser security model**

AJAX SECURITY THREAT ANALYSIS

Browser APIs that can be misused by Attackers



1. **Network Access via XMLHttpRequest or linkable attributes**
2. **Access to subdocument**
 - Reading/Writing Data
 - Code Overriding
 - Event handler overriding
3. **Access to sensitive window object properties, including global JavaScript functions and variables**
4. **Access to sensitive document properties**
5. **Information flow by events**

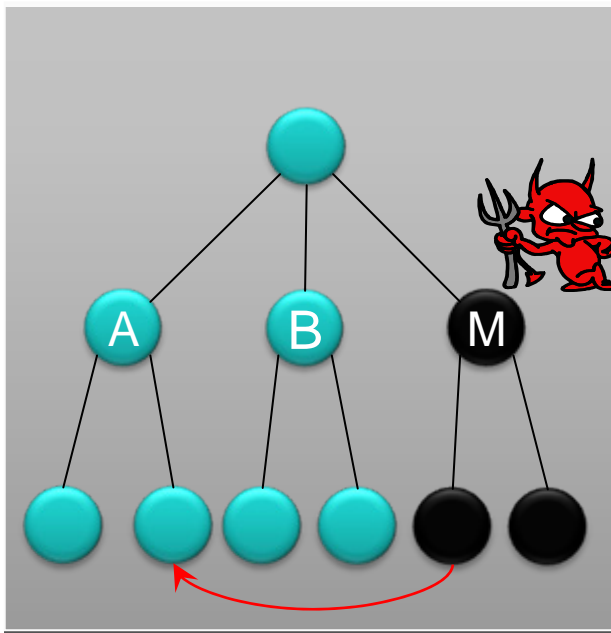
DOM-BASED ACCESS CONTROL

DOM-based Access Control – Problem background

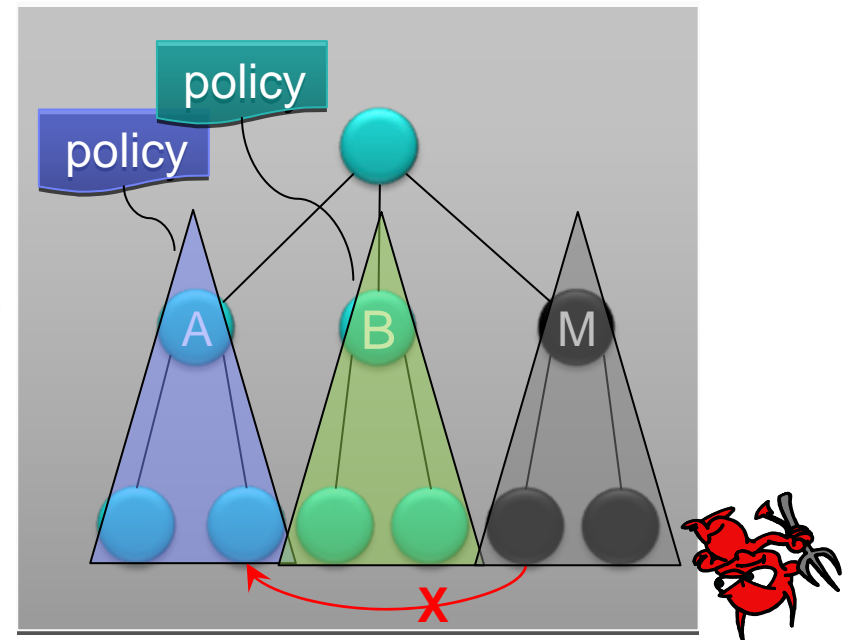
- **Mashups compose code/data from multiple sources**
 - Represented as a single DOM tree on browser
 - No concept of “domain” on the tree
 - Malicious JavaScript code can access any part of the DOM tree (e.g. password field)
- **Ajax Programming model is too flexible from the security perspective**
 - Easy to generate a new DOM node
 - Easy to override functions even DOM API
- **Questions**
 - How can we represent “domain”?
 - How can we attach policy and enforce it?
 - What is the programming model? Tool support?

DOM Level Access Control – A Component Model

- Sub-tree is associated with “domain” with policy
- Access to DOM is controlled by runtime



**Malicious data access
is allowed**



**Malicious data access
is prevented by policies**

ALTERNATIVE BROWSER SECURITY MODEL

Alternative Browser Security Model

- **Question:**

If you could change anything you want, what would the browser security model look like?

- What are high priority?
- Use cases?
- Standardization?

Backups

Threat 1. Network Access via XMLHttpRequest or linkable attributes

- **XMLHttpRequest is protected by the same-domain policy, but limited**
 - Only server name comparison as string literal
 - Path in the URL is ignored (`http://abc.com/~alice` vs. `http://abc.com/~malroy` are not distinguished)
 - Firefox allows to relax domain to superdomain (`www.ibm.com` -> `ibm.com` -> `com`)
- **Linkable attribute can be used for send/receive information**
 - Send arbitrary information
 - ``
 - Receiving information via a callback function
 - `<script src='http://foo.com/some?callback=mycallback'>`
 - Port Scanning
 - Try `<script src="http://192.168.0.x/icons/c.gif" />` where $0 < x < 255$, if the picture exists, an apache server is most likely running
 - Covert Channels
 - To send a small amount of information at a time by using some encoding patterns (e.g., `http://abc.com/img0.jpg` for sending '0', `img1.jpg` for '1', and so on)

2. Access to subdocument

- **Steal sensitive information**
 - `var pw = document.getElementById("password").value;`
- **Alternation of sensitive static contents or stylesheets**
 - `document.getElementById("price").value = "my_evil_price";`
 - `document.getElementById("warning").style = "color:white";`
- **Script injection**
 - `<script src="...malicious.js" />`
- **Event handler overriding**
 - `node.addHandler(maliciousEH);`
- **Overriding by innerHTML**
 - `document.body.innerHTML = ... <script>.....</script> ...`

3. and 4. Sensitive document/window properties

- **document.cookie – steal cookie info**
- **document.location, window.location**
 - Jump to an arbitrary URL (i.e., phishing)
 - Send an arbitrary information to URL (same as linkable attributes)
- **window.open()**
 - Send an arbitrary information to URL (same as linkable attributes)
- **document.write() – overwrite the document content**
- **Overriding JavaScript global functions and variables**
 - `document.getElementById = function(){ ...do something malicious }`
 - `Function UserFunc(){...}` overrides existing `UserFunc()`

5. Information Flow By Events

- **Keyboard Sniffer**

- Steal keystrokes by key events and report to a remote server

- **Mouse Sniffer**

- Steal mouse events and guess which buttons are pressed in a software keypad

- **Ambushing attacks**

- Event handlers can be used to hide an attack to be fired at a later time.

DOM Level Access Control - Discussions

- **How can we represent “domain”?**
 - Origin of data (e.g. URI, digital signature)
 - Leveraging existing technologies such as iframe
- **How can we define policy and attach it?**
- **How can we enforce access control?**
 - Modification of browser code
 - Plugin / extentions (e.g. Greasemonkey)
 - Realize it as JavaScript library
- **What is the programming model? Tool support?**