The New Threat: Browser Security

Marco Pistoia, Ph.D.
IBM T. J. Watson Research Center
Agenda

• Why browser security
• Browser security architecture
• HTML5 security
• Information-flow security
• JavaScript security
Agenda

• Why browser security
• Browser security architecture
• HTML5 security
• Information-flow security
• JavaScript security
Threat Evolution

- 1960: Program installation only by experts
- 1970: Program sharing
- 1980: Program downloading
- 1990: Mobile code
- 2000: Web applications
- 2011:
Access Control

• Mechanism to define and enforce which principals can access which resources

• Two components:
  – *Authentication* ascertains the identity of the principal who is making the requests
  – *Authorization* verifies that the principal is allowed to access the resource that has been requested
Authorization Decisions and Authorization Matrix

- An authorization decision can be seen as a function
- An authorization policy can be seen as a matrix [Lampson, 1992]
  - The columns of the matrix are Access Control Lists (ACLs)
  - The matrix grants access to system resources to users and groups

\[(\text{principal}, \text{request}, \text{object}) \rightarrow \text{true/false}\]

<table>
<thead>
<tr>
<th>Principal</th>
<th>File C: \log.txt</th>
<th>Socket ibm.com:80</th>
<th>System configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Administrator principal</td>
<td>read, write, execute</td>
<td>listen, connect</td>
<td>read, write</td>
</tr>
<tr>
<td>Text editor program</td>
<td>read, write</td>
<td>-</td>
<td>read</td>
</tr>
<tr>
<td>Internet browser</td>
<td>-</td>
<td>connect</td>
<td>read</td>
</tr>
</tbody>
</table>
Role-Based Access Control (RBAC)

• RBAC is a form of access control that can better represent the protection of information in enterprise systems [Ferraiolo and Kuhn, 1992]

• A role is a set of permissions

• Each permission represents a responsibility in an enterprise

• Roles are then assigned to users and groups
The Principle of Least Privilege

• In a computing environment, every module (such as a process, a user, or a program) must be able to access only such information and resources that are necessary to its legitimate purpose [Saltzer and Shroeder, 1975]

• Example:
  – Grant a text editor the permission to access the file system
  – Do not grant a text editor the permission to open a socket connection
Problems in Enforcing the Principle of Least Privilege

• An authorization policy must be neither too permissive nor too restrictive
  – Too permissive:
    • Violation of the Principle of Least Privilege
    • Program exposed to security attacks
  – Too restrictive
    • The policy-enforcement mechanism will generate run-time authorization failures
    • Security problems may arise
The Principle of Complete Mediation

- Every access to any resource must be mediated by an appropriate authorization check [Saltzer and Shroeder, 1975]
The Browser Is the New Operating System

• Cloud computing allows data and applications to be remote and accessible through the browser
  – Data: emails, messages, photos, movies, documents, spreadsheets, etc.
  – Applications: mail, messengers, photo and movie viewers and editors, document editors, etc.

• The model in which everything is local is no longer applicable in today’s computing world
Social Networks

• Most of our interactions with other people happen through social networks and email programs
  – Facebook
  – Myspace
  – Twitter
  – Google+
  – Various email programs (Gmail, Yahoo! Mail, etc.)
• Social networks are used to store and share data, communicate with others
• Nothing is maintained locally
• Every piece of information is remote
• The only application end users access is the browser
Why Browser Security

• Browser security has become as important as operating-system security
• If the browser is compromised, the consequences can be devastating:
  – Web-site defacement
  – Impersonation
  – Loss of confidential information
  – URL redirection
Web-site Defacement

• The contents of a Web site are replaced with malicious, unauthorized contents
• The end user may not realize that
Impersonation

• The authentication credential of a user are stolen
• A malicious user acts as the legitimate user
• Possible actions:
  – Send email
  – Post contents on social networks
  – Perform bank and/or credit card transactions
• Impersonation is a federal crime, but tracking it down can be very difficult
Theft of Confidential Information

• Credit card numbers
• User IDs and passwords
• Social security numbers
• Medical records
• Purchase history
URL Redirection

- A user visits a given URL
- The browser is maliciously redirected to another page, perhaps equally looking
- The user interacts with a malicious Web site without knowing it
Agenda

- Why browser security
- Browser security architecture
- HTML5 security
- Information-flow security
- JavaScript security
Browser Usage

I.E. 38.2%
Firefox 25.4%
Safari 9.9%
Chrome 16.5%
Opera 4.4%
Android 1.4%
Other 4.2%

Browser Usage on Wikimedia June 2011
# Known Vulnerabilities

<table>
<thead>
<tr>
<th>Browser</th>
<th>Extremely critical (number / oldest)</th>
<th>Highly critical (number / oldest)</th>
<th>Moderately critical (number / oldest)</th>
<th>Less critical (number / oldest)</th>
<th>Not critical (number / oldest)</th>
<th>Total (number / oldest)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Google Chrome 12</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Internet Explorer 6</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>17 November 2004; 6 years ago</td>
<td>8</td>
<td>20 November 2000; 10 years ago</td>
</tr>
<tr>
<td>Internet Explorer 7</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>30 October 2007; 5 years ago</td>
<td>12</td>
<td>26</td>
</tr>
<tr>
<td>Internet Explorer 8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>26 February 2007; 4 years ago</td>
<td>7</td>
<td>62</td>
</tr>
<tr>
<td>Internet Explorer 9</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>Firefox 3.6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Firefox 5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SeaMonkey 2.2</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>24 June 2011; 34 days ago</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Opera 11</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Safari 5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>8 June 2010; 13 months ago</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Microsoft Internet Explorer – General Architecture

• Internet Explorer uses a zone-based security framework that groups sites based on certain conditions, including whether a site is an Internet- or intranet-based site as well as a user-editable whitelist

• Security restrictions are applied per zone; all the sites in a zone are subject to the restrictions
Microsoft Internet Explorer – Attachment Execution Service

• Internet Explorer 6 SP2 onwards uses the Attachment Execution Service of Microsoft Windows to mark executable files downloaded from the Internet as being potentially unsafe

• Accessing files marked as such will prompt the user to make an explicit trust decision to execute the file, as executables originating from the Internet can be potentially unsafe

• This helps in preventing accidental installation of malware
Microsoft Internet Explorer – Phishing

• Internet Explorer 7 introduced the phishing filter, which restricts access to phishing sites unless the user overrides the decision

• With version 8, it also blocks access to sites known to host malware

• Downloads are also checked to see if they are known to be malware-infected
Microsoft Internet Explorer – Protected Mode

• In Windows Vista, Internet Explorer by default runs in what is called *Protected Mode*, where the privileges of the browser itself are severely restricted; it cannot make any system-wide changes
• One can optionally turn this mode off but this is not recommended
• This also effectively restricts the privileges of any add-ons
• As a result, even if the browser or any add-on is compromised, the damage the security breach can cause is limited
Microsoft Internet Explorer – Patches

• Patches and updates to the browser are released periodically and made available through the Windows Update service, as well as through Automatic Updates.

• Although security patches continue to be released for a range of platforms, most feature additions and security infrastructure improvements are only made available on operating systems which are in Microsoft’s mainstream support phase.
Mozilla Firefox – Standard Features

• Firefox uses a sandbox security model and limits scripts from accessing data from other Web sites based on the Same Origin Policy.
• It uses SSL/TLS to protect communications with Web servers using strong cryptography when using the HTTPS protocol.
• It also provides support for Web applications to use smartcards for authentication purposes.
The Mozilla Foundation offers a bug bounty to researchers who discover severe security holes in Firefox.

Official guidelines for handling security vulnerabilities discourage early disclosure of vulnerabilities so as not to give potential attackers an advantage in creating exploits.

Because Firefox generally has fewer publicly known unpatched security vulnerabilities than Internet Explorer, improved security is often cited as a reason to switch from Internet Explorer to Firefox.

- The Washington Post reports that exploit code for critical unpatched security vulnerabilities in Internet Explorer was available for 284 days in 2006.
- In comparison, exploit code for critical security vulnerabilities in Firefox was available for 9 days before Mozilla issued a patch to remedy the problem.
Mozilla Firefox – Patched Vulnerabilities

• A 2006 Symantec study showed that, although Firefox had surpassed other browsers in the number of vendor-confirmed vulnerabilities that year through September, these vulnerabilities were patched far more quickly than those found in other browsers.

• Symantec later clarified their statement, saying that Firefox still had fewer security vulnerabilities than Internet Explorer, as counted by security researchers.

• All patched vulnerabilities of Mozilla products are publicly listed: http://www.mozilla.org/security/known-vulnerabilities/
Google Chrome – Blacklists

• Chrome periodically retrieves updates of two blacklists (one for phishing and one for malware), and warns users when they attempt to visit a harmful site

• This service is also made available for use by others via a free public API called "Google Safe Browsing API"

• Google notifies the owners of listed sites who may not be aware of the presence of the harmful software
Google Chrome – Separation

• Chrome will typically allocate each tab to fit into its own process to prevent malware from installing itself and prevent what happens in one tab from affecting what happens in another
• Following the Principle of Least Privilege, each process is stripped of its rights and can compute, but cannot write files or read from sensitive areas (such as documents and desktop)
• Malicious software running in one tab is unable to sniff credit card numbers entered in another tab, interact with mouse inputs, or tell the operating system to run an executable on start-up and it will be terminated when the tab is closed
• This enforces a simple computer security model whereby there are two levels of multilevel security (user and sandbox) and the sandbox can only respond to communication requests initiated by the user
Google Chrome – Plugins

- Plugins, such as Adobe Flash Player, are not standardized and as such, cannot be sandboxed as tabs can be
- These often must run at, or above, the security level of the browser itself
- To reduce exposure to attack, plugins are run in separate processes that communicate with the renderer, itself operating at very low privileges in dedicated per-tab processes
- Plugins will need to be modified to operate within this software architecture while following the Principle of Least Privilege
- Chrome supports the Netscape Plugin Application Programming Interface (NPAPI), but does not support the embedding of ActiveX controls
- On March 30, 2010 Google announced that the latest development version of Chrome would include Adobe Flash as part of the browser
  - That eliminates the need to download and install it separately
  - Flash would be kept up to date as part of Chrome's own updates
- Java applet support is available in Chrome with Java 6 update 12 and above
Google Chrome – Private Browsing

• A private browsing feature called *incognito mode* prevents the browser from storing any history information or cookies from the Web sites visited

• Incognito mode is similar to the private browsing feature in Internet Explorer 8 (and up), Mozilla Firefox 3.5 (and up), Opera 10.5 (and up) and Safari
Agenda

• Why browser security
• Browser security architecture
• HTML5 security
• Information-flow security
• JavaScript security
HTML5

- HTML5 is a language for structuring and presenting content for the World Wide Web, a core technology of the Internet
- It is the fifth revision of the HTML standard
Message Passing

• HTML5 allows various components on a Web page to send messages to each other
• Messages can later be interpreted as commands to be executed or can be printed to the Web page
• Validation is essential because if a message that is supposed to be printed contains JavaScript code, that code will be executed and not just printed
Local and Global Storage

- HTML5 allows for local and global storage
- It is the same as “large cookies”
- The security policy around these storages is the same as cookies
Agenda

• Why browser security
• Browser security architecture
• HTML5 security
• Information-flow security
• JavaScript security
Information Security

• No illicit flow of information should be allowed in a program
• Two dimensions of information security:
  – Integrity: Valuable information should not be damaged by any computation
  – Confidentiality: Valuable information should not be revealed by any computation
• Confidentiality different from:
  – Secrecy: Secret information is not leaked to public listeners
  – Anonymity: A public observer cannot learn the identities of the participating principals even though actions might be known
Downgrading

• An information-security policy can establish that:
  – Certain parts of secret information can be *declassified* and revealed to certain public listeners. For example:
    • Last 4 digits of SSN can be revealed to bank teller
    • Result of a password check can be revealed to anyone
  – Certain parts of untrusted input can be *endorsed* and used in certain trusted computations. For example:
    • Untrusted user input can be used in a Web application if it is properly formatted

<table>
<thead>
<tr>
<th></th>
<th>Integrity</th>
<th>Confidentiality</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>High</strong></td>
<td>Untrusted</td>
<td>Secret</td>
</tr>
<tr>
<td><strong>Low</strong></td>
<td>Trusted</td>
<td>Public</td>
</tr>
<tr>
<td><strong>Downgrading</strong></td>
<td>Endorsement</td>
<td>Declassification</td>
</tr>
</tbody>
</table>
Example: Injection Flaws in Web Applications

```java
public void submitQuery(String userName) {
    String query = "SELECT id FROM users WHERE name = '' + userName + '''";
    execute(query);
}
```
XSS

- Consider a site that accepts a user name in form input and then displays it in the page.
- Entering the name John and clicking Submit might result in loading a URL like http://www.example.com/mycgi?username=John, and the following snippet of HTML to appear in the resulting page: Hello, <b>John</b>!
- If someone can get you to click on a link to http://www.example.com/mycgi?username=John<script>alert('Uh oh');</script>, the CGI might write the following HTML into the resulting page: Hello, <b>John</b><script>alert('Uh oh');</script></b>
- The script passed in through the username URL parameter was written directly into the page, and its JavaScript is executed as normal.
XSS Scenario

<SCRIPT>...</SCRIPT>

Web Application

Attacker's evil script executed using victim's credentials

Attacker's evil script

Attacker

Victim

<SCRIPT>...</SCRIPT>
XSS Prevention

• Input validation
• HTML-escape data
Malicious File Executions

- Web application manages files in the file system
- The name or contents of such files are often obtained from user input
- Maliciously crafted user inputs could cause the execution or deletion of security-sensitive files
Information Leakage and Improper Error Handling
Problems in Enforcing Information Security

• Policies can become very complex

• It may be difficult and expensive to track the actual flows of information
  – Complex flows through the program
  – Covert channels

• Implicit flows
  – Confidentiality: value of $x$ may reveal values of $a$ and $b$
  – Integrity: value of $b$ influences value of $x$ even if $b$ is false

```java
int x = 0;
if (b) {
    x = a;
}
```
Agenda

• Why browser security
• Browser security architecture
• HTML5 security
• Information-flow security
• JavaScript security
JavaScript

• JavaScript is the programming language for the Web
• 99% of all the Web sites in the world use JavaScript, according to alexa.com
• JavaScript allows for full manipulation of the Document Object Model (DOM)
• JavaScript has replaced Java as the client-side programming language
Risks with JavaScript

• Downloading and running programs written by unknown parties is dangerous

• Most people do not realize that nearly every time they load a Web page, they are allowing code written by an unknown party to execute on their computers

• Since it would be annoying to have to confirm your wish to run JavaScript each time you load a new Web page, browsers implement a security policy designed to reduce the risk such code poses to the end user

• Example: JavaScript code cannot access your file system
JavaScript Security Model

- Scripts are confined inside a sandbox where they cannot have access to the operating system or file system.
- Scripts are permitted access only to data in the current document or closely related documents (those from the same site as the current document).
- No access is granted to the local file system, the memory space of other running programs, or the operating system’s networking layer.
The Reality

- The reality of the situation, however, is that often scripts are not properly sandboxed.
- There are numerous ways that a script can exercise power beyond what you might expect, both by design and by accident.
- The fundamental premise of browsers’ security models is that there randomly encountered code is by default hostile.
- However
  - Code coming from trusted sources can escape the sandbox, often without requiring the explicit consent of the user.
  - Scripts can gain access to otherwise privileged information in other browser windows when the pages come from related domains.
Same-Origin Policy

• It is the primary JavaScript security policy
• It prevents scripts loaded from one Web site from getting or setting properties of a document loaded from a different site or using a different protocol and port number
• It applies to scripts attempting to access the content of frames
  – If two frames have not been loaded from the same site using the same protocol, scripts cannot cross the framed boundary
Same-Origin Check

• When a script attempts to access properties or methods in a different window, for example, using the handle returned by window.open(), the browser performs a same-origin check on the URLs of the documents in question
  – If the URLs of the documents pass this check, the property can be accessed
  – If they do not, an error is thrown

• The same-origin check consists of verifying that the URL of the document in the target window “has the same origin” as the document containing the calling script

• Two documents have the same origin if they were loaded from the same server using the same protocol and port
Problems

• Older browsers did not enforce the same-origin policy correctly
• The same-origin policy does not protect against cross-site interactions when two Web sites are hosted by the same server
• You cannot turn off the same-origin policy, for example in an intranet, so you have to use ActiveX controls or use signed scripts
• Denial of service attacks are possible
JavaScript is present on many popular Web sites.
Consequences of Taint Violations

• Read and write access to saved data in cookies and local data stores
• Read and write access to data in the web page
• Key loggers
• Impersonation
• Phishing via page modifications or redirects
```javascript
var el1 = document.getElementById("d1");
function foo() {
  var el2 = document.getElementById("d2");
  function bar() {
    var el3 = new Element();
    var s = encodeURIComponent(el2.innerText);
    document.write(s);
    el1.innerHTML = el2.innerText;
    document.location = el3.innerText;
  }
  bar();
}
foo();
function baz(a, b) {
  a.f = document.URL;
  document.write(b.f);
}
var x = new Object();
baz(x, x);
```
var el1 = document.getElementById("d1");
function foo() {
    var el2 = document.getElementById("d2");
    function bar() {
        var el3 = new Element();
        var s = encodeURIComponent(el2.innerText);
        document.write(s);
        el1.innerHTML = el2.innerText;
        document.location = el3.innerText;
    }
    bar();
}
foo();
function baz(a, b) {
    a.f = document.URL;
    document.write(b.f);
}
var x = new Object();
baz(x, x);
```javascript
var el1 = document.getElementById("d1");
function foo() {
    var el2 = document.getElementById("d2");
    function bar() {
        var el3 = new Element();
        var s = encodeURIComponent(el2.innerText);
        document.write(s);
        el1.innerHTML = el2.innerText;
        document.location = el3.innerHTML;
    }
    bar();
}
foo();
function baz(a, b) {
    a.f = document.URL;
    document.write(b.f);
}
var x = new Object();
baz(x, x);
```
Rules

• Application developers must be aware of security rules
• A rule is a triple <Sources, Sinks, Sanitizers>
• Not all sources are valid for all sinks, and not all sanitizers are valid for all sinks
Rules

• A rule is a triple <Sources, Sinks, Sanitizers>
• Not all sources are valid for all sinks, and not all sanitizers are valid for all sinks

• Sources
  – Seeds of untrusted data
  – Field gets or returns of function calls
  – Ex: document.url
Rules

• A rule is a triple <Sources, Sinks, Sanitizers>
• Not all sources are valid for all sinks, and not all sanitizers are valid for all sinks

• Sources
  – Seeds of untrusted data
  – Field gets or returns of function calls
  – Ex: document.url

• Sinks
  – Security critical operations
  – Field puts or parameters to function calls
  – Ex: element.innerHTML
Rules

• A rule is a triple <Sources, Sinks, Sanitizers>
• Not all sources are valid for all sinks, and not all sanitizers are valid for all sinks

• Sources
  – Seeds of untrusted data
  – Field gets or returns of function calls
  – Ex: document.url

• Sinks
  – Security critical operations
  – Field puts or parameters to function calls
  – Ex: element.innerHTML

• Sanitizers
  – Marks flow as non-dangerous
  – Function calls
  – Ex: encodeURIComponent(str)
Example

```javascript
function foo(p1, p2) {
  p1.f = p2.f;
}

var a = new Object();
var b = new Object();
b.f = window.location.toString();

var c = new Object();
var d = new Object();
d.f = "safe";

foo(a, b);
foo(c, d);

document.write(a.f); // This is a taint violation

document.write(c.f); // This is NOT a taint violation
```

Since `d.f` is not tainted, `c.f` will not be tainted
Conclusion

• As a user:
  – Update your browsers and plugins
  – Check the URL for HTTPS and parameters

• As a developer:
  – Validate your inputs
  – Always use the latest technologies
  – Minify your JavaScript
Thank You!

pistoia@us.ibm.com
research.ibm.com/people/p/pistoia