Goals of Web security

- Safely browse the web
  - Users should be able to visit a variety of web sites, without incurring harm:
    - No stolen information (without user’s permission)
    - Site A cannot compromise session at Site B

- Support secure web applications
  - Applications delivered over the web should have the same security properties we require for stand-alone applications
Two Sides of Web Security

- Web browser
  - Responsible for securely confining Web content presented by visited websites
- Web applications
  - Online merchants, banks, blogs, Google Apps ...
  - Mix of server-side and client-side code
    - Server-side code written in PHP, Ruby, ASP, JSP... runs on the Web server
    - Client-side code written in JavaScript... runs in the Web browser
  - Many potential bugs: XSS, XSRF, SQL injection

Where Does the Attacker Live?

Browser

Network attacker

Malware attacker

Website

Web attacker
Web Threat Models

- Web attacker
- Network attacker
  - Passive: wireless eavesdropper
  - Active: evil Wi-Fi router, DNS poisoning
- Malware attacker
  - Malicious code executes directly on victim’s computer
  - To infect victim’s computer, can exploit software bugs (e.g., buffer overflow) or convince user to install malicious content (how?)
    - Masquerade as an antivirus program, video codec, etc.

Web Attacker

- Controls a malicious website (attacker.com)
  - Can even obtain an SSL/TLS certificate for his site ($0)
- User visits attacker.com – why?
  - Phishing email, enticing content, search results, placed by an ad network, blind luck ...
  - Attacker’s Facebook app
- Attacker has no other access to user machine!
- Variation: “iframe attacker”
  - An iframe with malicious content included in an otherwise honest webpage
    - Syndicated advertising, mashups, etc.
Browser Security Model

- Http
- Rendering content
- Isolation
- Communication
- Navigation
- Security User Interface
- Cookies

HTTP
HTTP: 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
- Evolution
  - HTTP 1.0: simple
  - HTTP 1.1: more complex

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

<table>
<thead>
<tr>
<th>Method</th>
<th>File</th>
<th>HTTP version</th>
<th>Headers</th>
</tr>
</thead>
</table>
| GET    | /index.html     | HTTP/1.1     | GET /index.html HTTP/1.1
Accept: image/gif, image/x-bitmap, image/jpeg, */*
Accept-Language: en
Connection: Keep-Alive
User-Agent: Mozilla/1.22 (compatible; MSIE 2.0; Windows 95)
Host: www.example.com
Referer: http://www.google.com?q=dingbats |

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>
<th>Headers</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP/1.0</td>
<td>200 OK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Date: Sun, 21 Apr 1996 02:20:42 GMT</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Server: Microsoft-Internet-Information-Server/5.0
Connection: keep-alive
Content-Type: text/html
Last-Modified: Thu, 18 Apr 1996 17:39:05 GMT
Set-Cookie: ...
Content-Length: 2543 |

Data

Cookies

<HTML> Some data... blah, blah, blah </HTML>
A **cookie** is a file created by a website to store information in the browser.

HTTP is a stateless protocol; cookies add state.

### What Are Cookies Used For?

- **Authentication**
  - The cookie proves to the website that the client previously authenticated correctly
- **Personalization**
  - Helps the website recognize the user from a previous visit
- **Tracking**
  - Follow the user from site to site; learn his/her browsing behavior, preferences, and so on
Rendering and events

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

Events can be
- User actions: OnClick, OnMouseover
- Rendering: OnLoad, OnBeforeUnload
- Timing: setTimeout(), clearTimeout()
Document Object Model (DOM)

- Object-oriented interface used to read and write docs
  - 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)`
- Includes Browser Object Model (BOM)
  - `window`, `document`, `frames[]`, `history`, `location`, `navigator` (type and version of browser)

JavaScript

- “The world’s most misunderstood programming language”
- Language executed by the Web browser
  - Scripts are embedded in webpages
  - Can run before HTML is loaded, before page is viewed, while it is being viewed, or when leaving the page
- Used to implement “active” webpages and Web applications
  - A potentially malicious webpage gets to execute some code on user’s machine
JavaScript History

- Developed by Brendan Eich at Netscape
  - Scripting language for Navigator 2
- Later standardized for browser compatibility
  - ECMAScript Edition 3 (aka JavaScript 1.5)
- Related to Java in name only
  - Name was part of a marketing deal
  - “Java is to JavaScript as car is to carpet”
- Various implementations available
  - SpiderMonkey, RhinoJava, others

Common Uses of JavaScript

- Page embellishments and special effects
- Dynamic content manipulation
- Form validation
- Navigation systems
- Hundreds of applications
  - Google Docs, Google Maps, dashboard widgets in Mac OS X, Philips universal remotes ...
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?

Security consequences

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

Important Point: A web page can send information to any site
JavaScript onError

- **Basic function**
  - Triggered when error occurs loading a document or an image

  ```html
  <img src="image.gif" onerror="alert('The image could not be loaded.')"
  />
  ```

  - Runs onError handler if image does not exist and cannot load

  [http://www.w3schools.com/jsref/jsref.onError.asp](http://www.w3schools.com/jsref/jsref.onError.asp)

---

JavaScript timing

- **Sample code**

  ```html
  <html><body><img id="test" style="display: none">
  <script>
    var test = document.getElementById('test');
    var start = new Date();
    test.onerror = function() {
      var end = new Date();
      alert("Total time: " + (end - start));
    }
    test.src = "http://www.example.com/page.html";
  </script>
  </body></html>
  ```

  - When response header indicates that page is not an image, the browser stops and notifies JavaScript via the onerror handler.
Security consequence

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

1) "show me dancing pigs!"
2) "check this out"
3) port scan results

Remote scripting

- Goal
  - Exchange data between a client-side app running in a browser and server-side app, without 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

Important Point: A web can maintain bi-directional communication with browser (until user closes/quits)
ISOLATION

Browser Sandbox

- Goal: safely execute JavaScript code provided by a website
  - No direct file access, limited access to OS, network, browser data, content that came from other websites
- Same origin policy
  - Can only access properties of documents and windows from the same domain, protocol, and port
- User can grant privileges to signed scripts
  - UniversalBrowserRead/Write, UniversalFileRead, UniversalSendMail
Same Origin Policy

Same Origin Policy (SOP) for DOM:
Origin A can access origin B’s DOM if A and B have same (protocol, domain, port)

Same Origin Policy (SOP) for cookies:
Generally, based on ([protocol], domain, path)
optional

Frame and iFrame

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

Analogy

Operating system
- Primitives
  - Processes
  - System calls
  - File system
- Principals: Users (ID)
  - Discretionary access control
- Vulnerabilities
  - Buffer overflow
  - Root exploit

Web browser
- Primitives
  - Frames
  - Content (including JavaScript)
  - Document object model, Cookies / localStorage
- Principals: “Origins”
  - Mandatory access control
- Vulnerabilities
  - Cross-site scripting
  - Cross-site request forgery
  - Cache history attacks
Policy Goals

- Safe to visit an evil web site
- Safe to visit two pages at the same time
  - Address bar distinguishes them
- Allow safe delegation

Browser security mechanism

- Each frame of a page has an origin
  - Origin = protocol://host:port
- Frame can access its own origin
  - Network access, Read/write DOM, Storage (cookies)
- Frame cannot access data associated with a different origin
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?

Domain Relaxation

- Origin: scheme, host, (port), hasSetDomain
- Try document.domain = document.domain
## Additional mechanisms

- Cross-origin network requests
  - Access-Control-Allow-Origin: <list of domains>
  - Access-Control-Allow-Origin: *
- Cross-origin client side communication
  - Client-side messaging via navigation (older browsers)
  - postMessage (newer browsers)

## Library import excluded from SOP

Same origin policy does not apply to directly included scripts (not enclosed in an iframe)

```html
<script src=https://seal.verisign.com/getseal?
    host_name=a.com></script>
```

- Script has privileges of imported page, NOT source server.
- Can script other pages in this origin, load more scripts
COMMUNICATION

**window.postMessage**

- New API for inter-frame communication
  - Supported in latest betas of many browsers
  - A network-like channel between frames

[Image of contact sharing]
**postMessage syntax**

```javascript
frames[0].postMessage("Attack at dawn!", "http://b.com/");

window.addEventListener("message", function (e) {
  if (e.origin == "http://a.com") {
    ... e.data ... 
  }
}, false);
```

**Why include “targetOrigin”?**

- What goes wrong?
  
  ```javascript
  frames[0].postMessage("Attack at dawn!");
  ```

- Messages sent to *frames*, not principals
- When would this happen?
NAVIGATION

A Guninski Attack

```javascript
window.open("https://attacker.com/", "awglogin");
```
What should the policy be?

Legacy Browser Behavior

<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>
Window Policy Anomaly

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
  - Navigate a frame if can draw over the frame
Solution: Descendant Policy

- Best security/compatibility trade-off
  - Security: Respect pixel delegation
  - Compatibility: Least restrictive such policy

- Implementation
  - Wrote patches for Firefox and Safari
  - Wrote over 1000 lines of regression tests

- Deployment
  - Apple released patch as security update
  - Mozilla included policy in Firefox 3

Adoption of Descendant Policy

<table>
<thead>
<tr>
<th>Browser</th>
<th>Policy</th>
</tr>
</thead>
<tbody>
<tr>
<td>IE7 (no Flash)</td>
<td>Descendant</td>
</tr>
<tr>
<td>IE7 (with Flash)</td>
<td>Descendant</td>
</tr>
<tr>
<td>Firefox 3</td>
<td>Descendant</td>
</tr>
<tr>
<td>Safari 3</td>
<td>Descendant</td>
</tr>
<tr>
<td>Opera 9</td>
<td>(many policies)</td>
</tr>
<tr>
<td>HTML 5</td>
<td>Descendant</td>
</tr>
</tbody>
</table>
When is it safe to type my password?

SECURITY USER INTERFACE

Safe to type your password?
Safe to type your password?

Safe to type your password?
Safe to type your password?

BANK OF THE WEST
Portfolio Online

Welcome to Portfolio Online!
Please enter your access ID and click "Continue."

Access ID:

Terms and Conditions
please read our Terms & Conditions.

Bankofonline.malavita.com

Safe to type your password?
Mixed Content: HTTP and HTTPS

- **Problem**
  - Page loads over HTTPS, but has HTTP content
  - Network attacker can control page
  - **IE:** displays mixed-content dialog to user
    - Flash files over HTTP loaded with no warning (!)
    - **Note:** Flash can script the embedding page
  - **Firefox:** red slash over lock icon (no dialog)
    - Flash files over HTTP do not trigger the slash
  - **Safari:** does not detect mixed content
Mixed content and network attacks

- banks: after login all content over HTTPS
  - Developer error: Somewhere on bank site write
    \[
    \text{<script src='http://www.site.com/script.js'> </script>}
    \]
  - Active network attacker can now hijack any session

- Better way to include content:
  \[
  \text{<script src='//www.site.com/script.js'> </script>}
  \]
  - served over the same protocol as embedding page

Lock Icon 2.0

- Extended validation (EV) certificates

  - Prominent security indicator for EV certificates
  - note: EV site loading content from non-EV site does not trigger mixed content warning
Finally: the status Bar

- Trivially spoofable

<a href="http://www.paypal.com/">
 onclick="this.href = 'http://www.evil.com/';">PayPal</a>

COOKIES: CLIENT STATE
Cookie authentication

Browser

POST login.cgi
Username & pwd
Set-cookie: auth=val

Web Server

Validate user
auth=val

Store val

Auth server

GET restricted.html
Cookie: auth=val
If YES,
restricted.html

Check val
YES/NO

Cookie Security Policy

- Browser will store:
  - At most 20 cookies/site, 3 KB / cookie
  - Origin is the tuple <domain, path>
    - Can set cookies valid across a domain suffix
Cookie Classification

- Based on origin
  - First-party cookie
  - Third-party cookie
- Based on lifetime
  - session cookie
  - persistent cookie
- Secure Cookie
- HTTPOnly Cookie

Secure Cookies

- A secure cookie is encrypted when transmitting from client to server
- Provides confidentiality against network attacker
  - Browser will only send cookie back over HTTPS
httpOnly Cookies

- Cookie sent over HTTP(s), but not accessible to scripts
  - cannot be read via `document.cookie`
  - Helps prevent cookie theft via XSS

... but does not stop most other risks of XSS bugs