

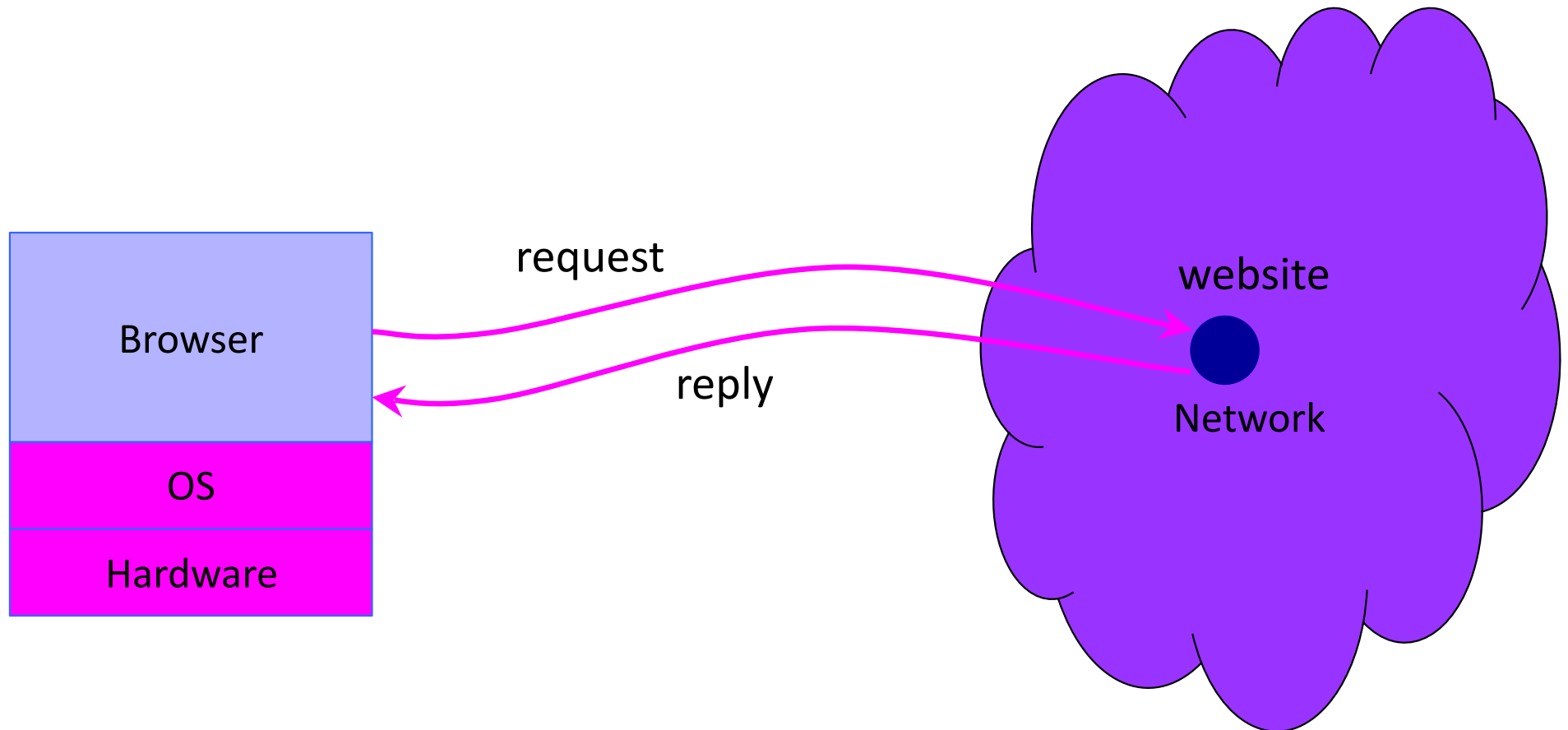
Web Security Model and JavaScript Rootkits

Yan Huang



Credits: slides adapted from
Stanford and Cornell Tech

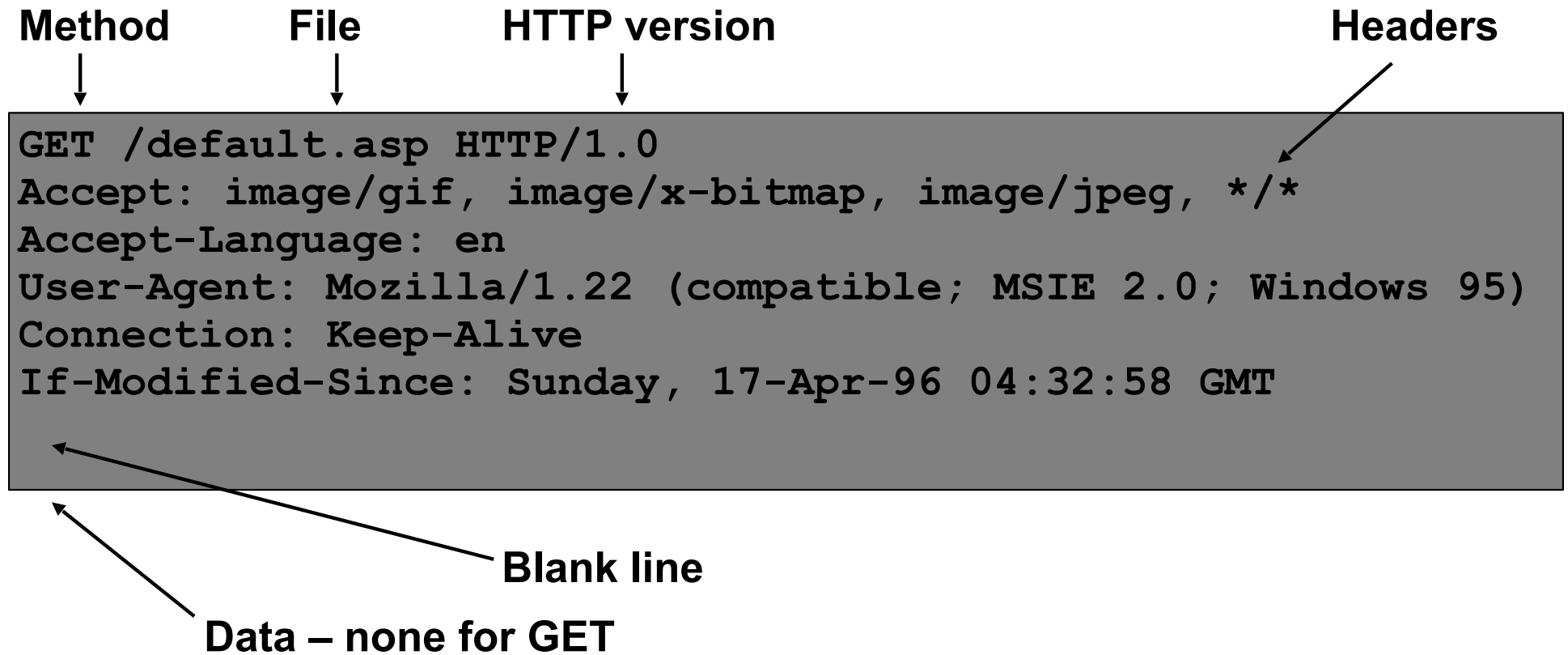
Browser and Network



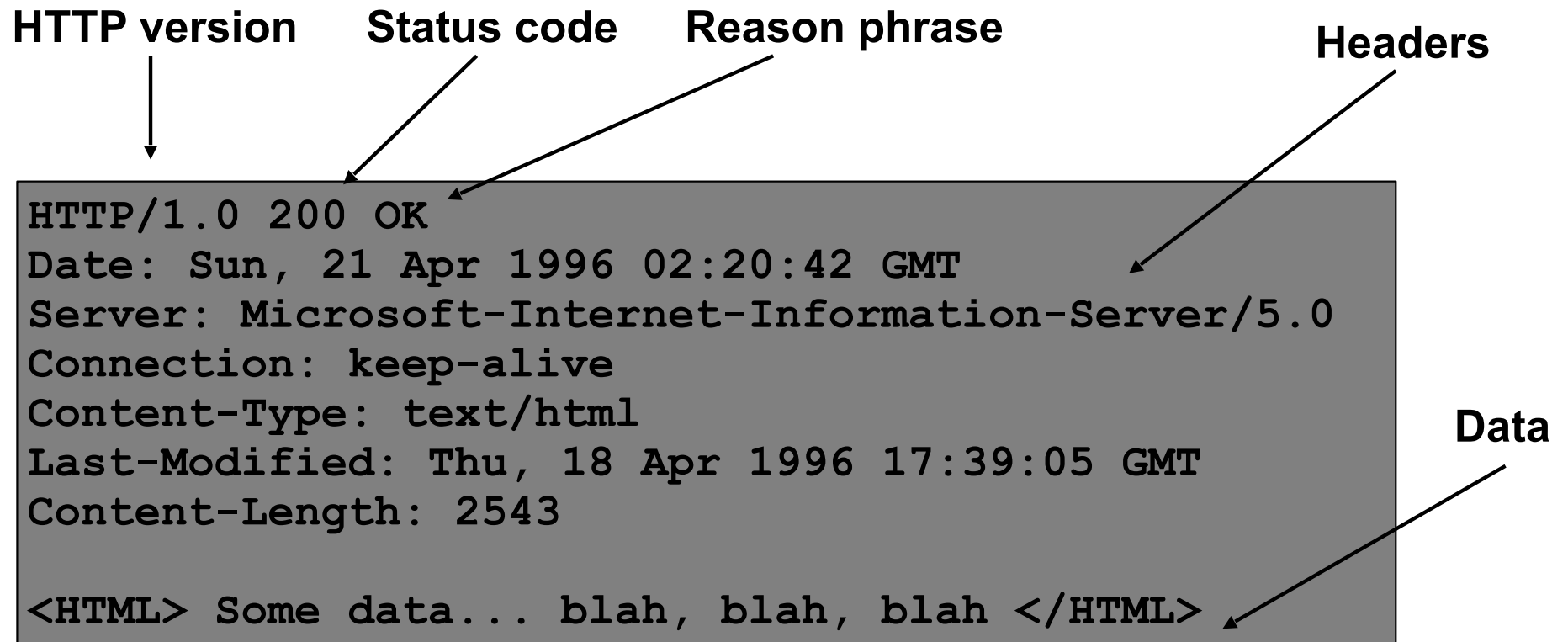
HTTP: HyperText Transfer Protocol

- ◆ 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

HTTP Request

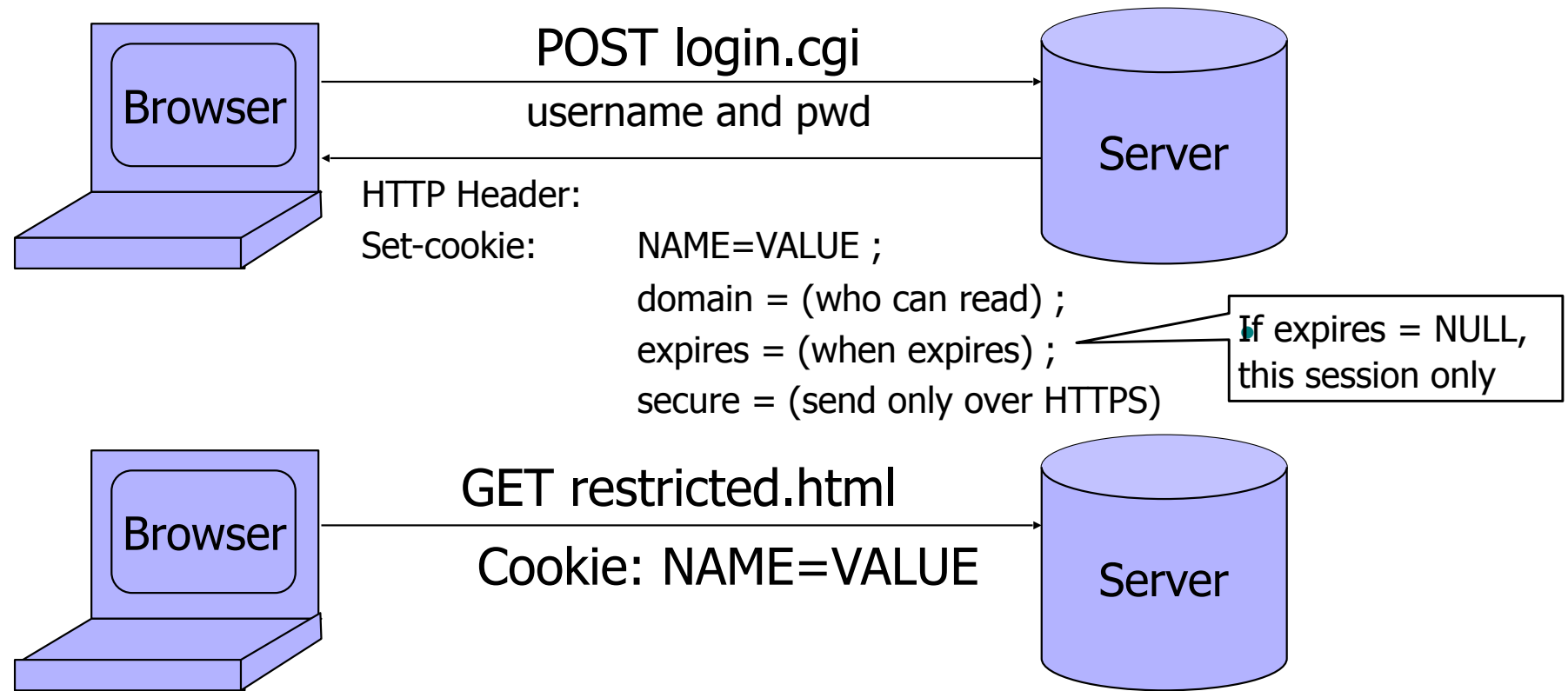


HTTP Response



Website Storing Info In Browser

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

- Recognize the user from a previous visit and customize the web pages

◆ Tracking

- Follow the user from site to site; learn his/her browsing behavior, preferences, and so on

Goals of Web Security

◆ Browse the Web **Safely**

- A malicious website cannot steal or modify information from legitimate sites to harm the users ...
- ... even if visited concurrently with a legitimate site - in separate browser windows, tabs, or iframes

◆ Support secure Web applications

- Applications delivered over the Web should have the same security properties we require for standalone applications (what are these properties?)

All of These Should Be Safe

◆ Safe to visit an evil website



◆ Safe to visit two pages
at the same time



◆ Safe delegation



Two Sides of Web Security

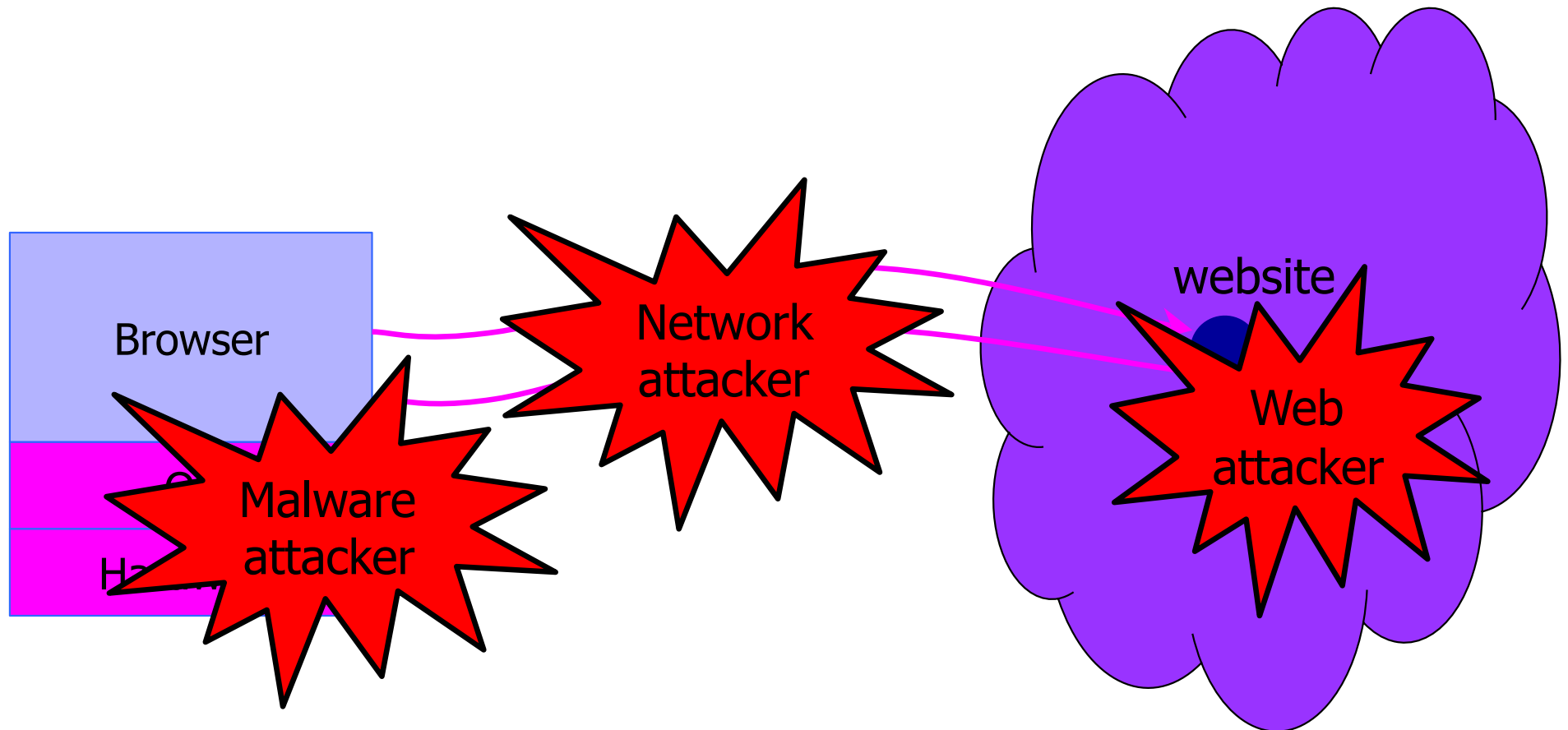
◆ Web browser

- 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?



Web Attacker

- ◆ Controls a malicious website (attacker.com)
 - Can even obtain an SSL/TLS certificate for his site (\$0)
- ◆ Attracting user visits to attacker.com
 - 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.

Dangerous Websites

- ◆ Microsoft's 2006 "Web patrol" study identified hundreds of URLs that could successfully exploit unpatched Windows XP machines
 - Many 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 then redirects to the exploit site
 - One of the malicious sites was providing exploits to 75 "innocuous" sites focusing on (1) celebrities, (2) song lyrics, (3) wallpapers, (4) video game cheats, and (5) wrestling

OS vs. Browser Analogies

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 and localStorage

◆ Principals: “Origins”

- Mandatory access control

◆ Vulnerabilities

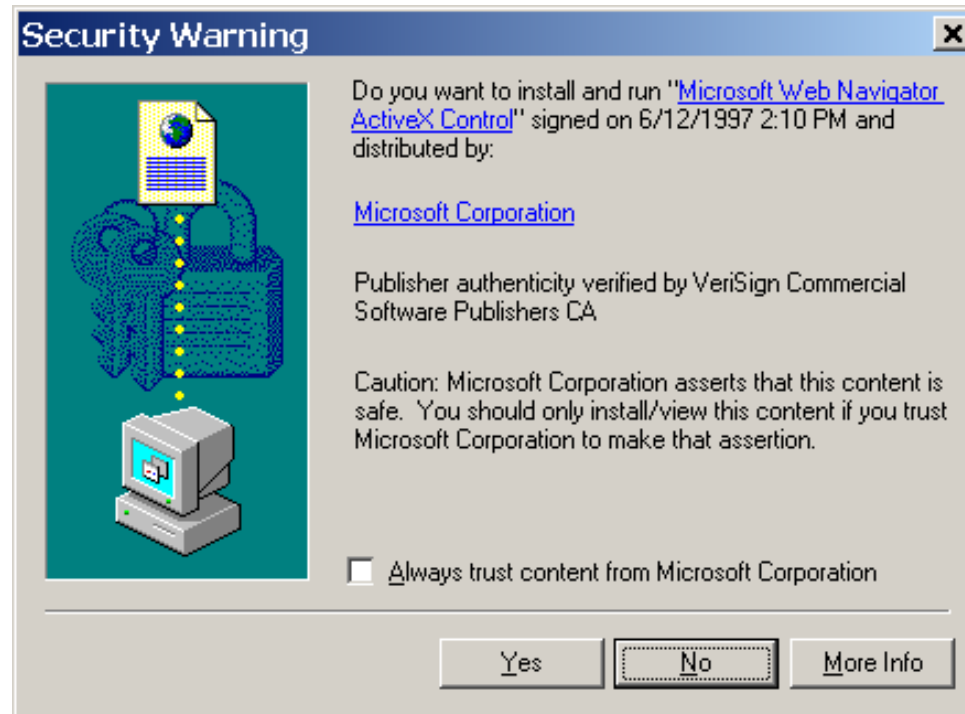
- Cross-site scripting
- Universal scripting

ActiveX

- ◆ ActiveX “controls” are compiled binaries that reside on the client machine
 - Downloaded and installed, like any other executable
 - Activated by an HTML object tag on the page
 - Run as native binaries, not interpreted by the browser
- ◆ Security model relies on three components
 - Digital signatures to verify the source of the control
 - Browser policy can reject controls from network zones
 - Controls can be marked by author as “safe for initialization” or “safe for scripting”

Once accepted, installed and started, no control over execution!

Installing ActiveX Controls



If you install and run, no further control over the code,
same access as any other program you installed

ActiveX Risks

◆ From MSDN:

- “An ActiveX control can be an extremely insecure way to provide a feature. Because it is a Component Object Model (COM) object, it can do anything the user can do from that computer. It can read from and write to the registry, and it has access to the local file system. From the moment a user downloads an ActiveX control, the control may be vulnerable to attack because any Web application on the Internet can repurpose it, that is, use the control for its own ends whether sincere or malicious.”

◆ How can a control be “repurposed?”

- Once a control is installed, any webpage that knows the control’s class identifier (CLSID) can access it using an HTML object tag embedded in the page

Browser: Basic Execution Model

◆ Each browser window or frame:

- Loads content
- Renders
 - Processes HTML and executes scripts to display the page
 - May involve images, subframes, etc.
- Responds to **events**

◆ Events

- User actions: OnClick, OnMouseover
- Rendering: OnLoad, OnUnload
- Timing: setTimeout(), clearTimeout()

HTML and 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)
```

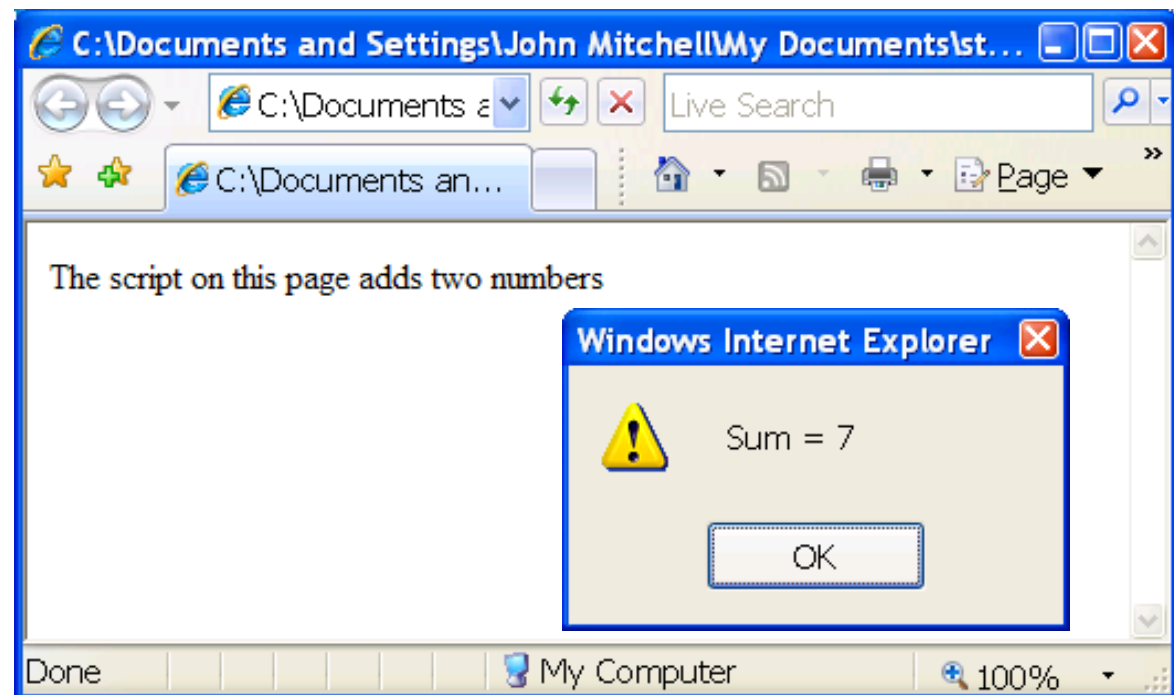
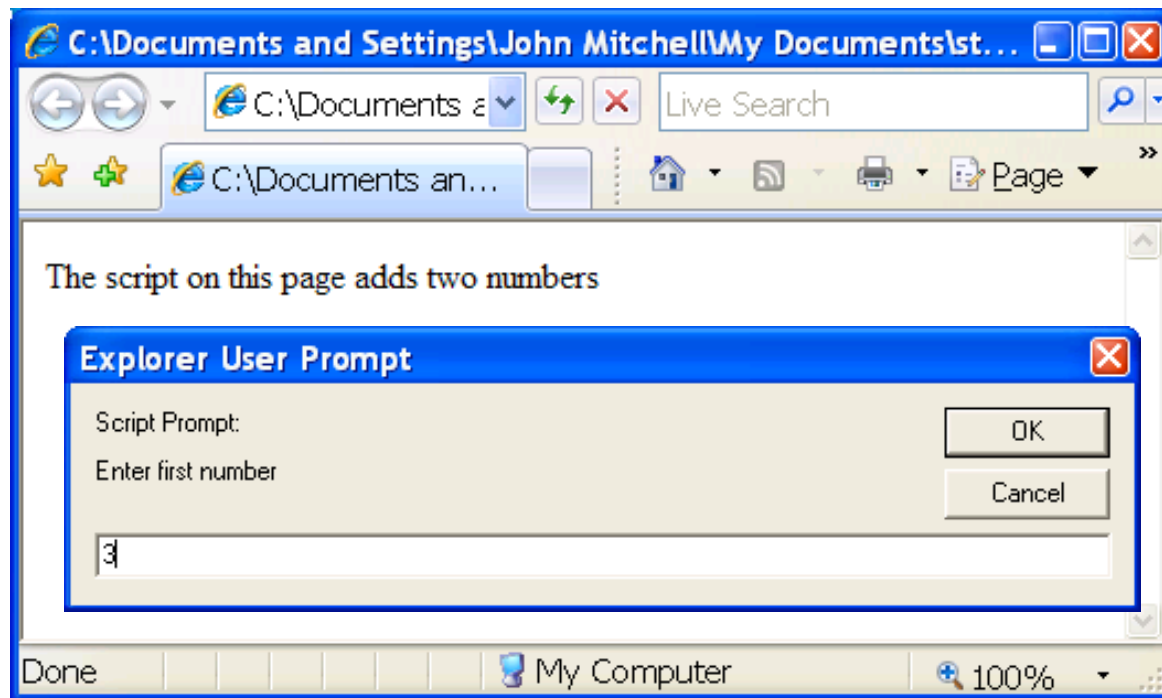
```
    alert("Sum = " + sum)
```

```
</script>
```

```
...
```

```
</html>
```


Browser receives content,
displays HTML and executes scripts



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>
```

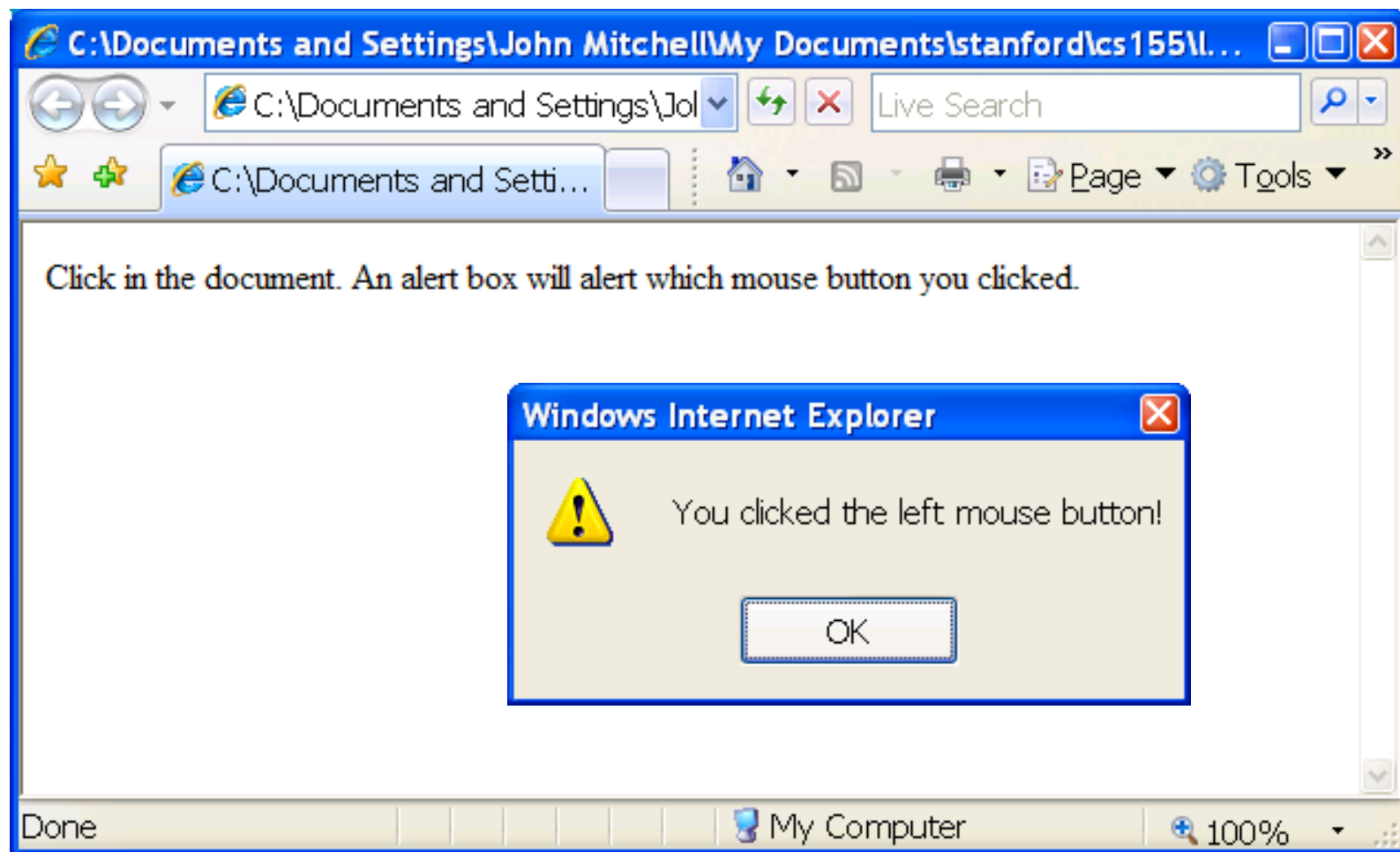
Script defines a
page-specific function



```
...  
<body onmousedown="whichButton(event)">  
...  
</body>
```

Function gets executed
when some event happens

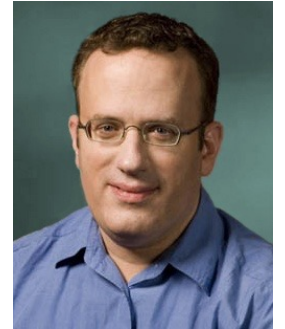




JavaScript

- ◆ “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 ...

JavaScript in Webpages

◆ Embedded in HTML as a `<script>` element

- Written directly inside a `<script>` element
 - `<script> alert("Hello World!") </script>`
- In a file linked as `src` attribute of a `<script>` element
`<script type="text/JavaScript" src="functions.js"></script>`

◆ Event handler attribute

``

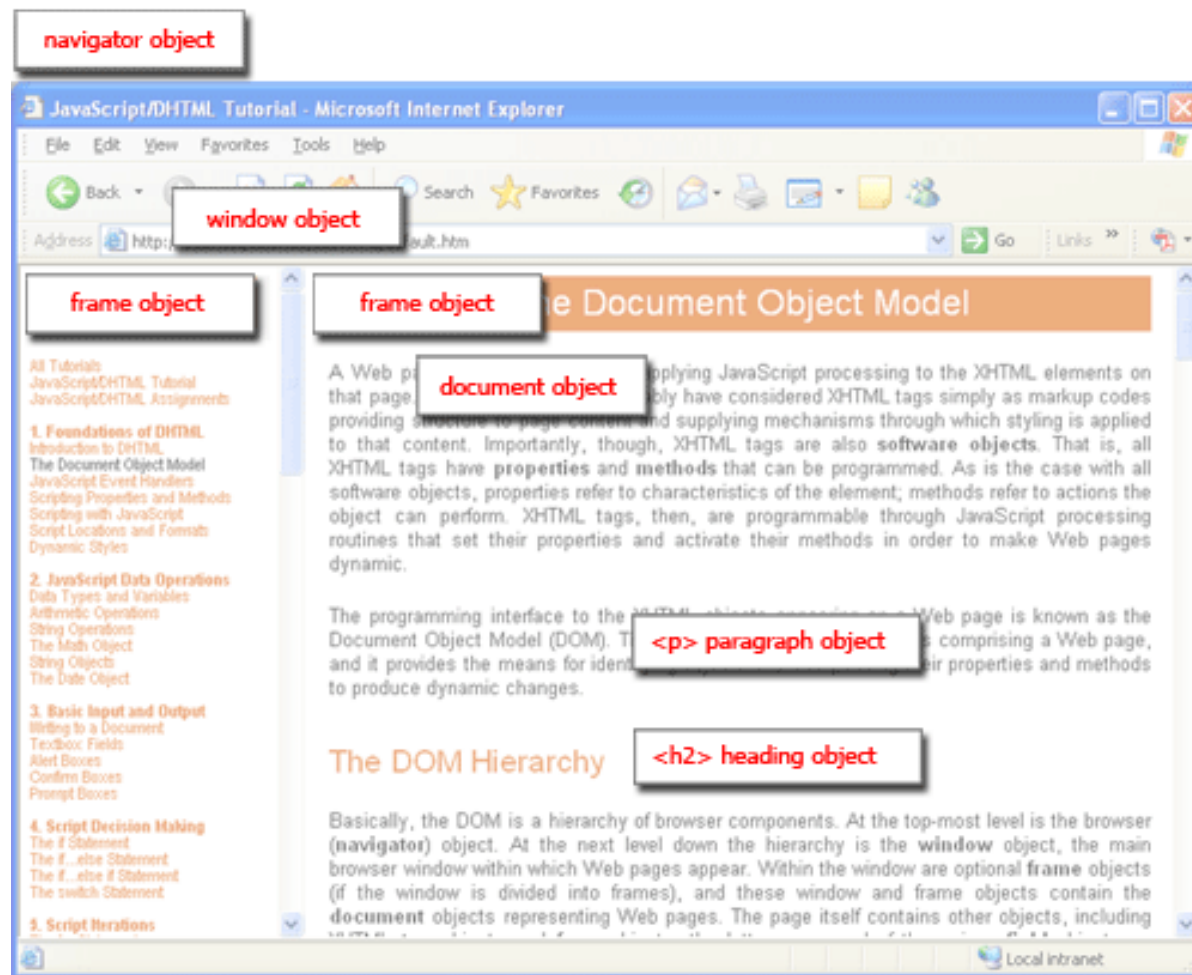
◆ Pseudo-URL referenced by a link

`Click me`

Document Object Model (DOM)

- ◆ HTML page is structured data
- ◆ DOM is object-oriented representation of the hierarchical HTML structure
 - **Properties:** `document.alinkColor`, `document.URL`, `document.forms[]`, `document.links[]`, ...
 - **Methods:** `document.write(document.referrer)`
 - These change the content of the page!
- ◆ Also Browser Object Model (BOM)
 - `Window`, `Document`, `Frames[]`, `History`, `Location`, `Navigator` (type and version of browser)

Browser and Document Structure



W3C standard differs from models supported in existing browsers

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`

- 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 "

Sample HTML

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

Page Manipulation with JavaScript

◆ Some possibilities

- `createElement(elementName)`
- `createTextNode(text)`
- `appendChild(newChild)`
- `removeChild(node)`

Sample HTML

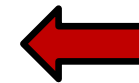
```
<ul id="t1">  
<li> Item 1 </li>  
</ul>
```

◆ Example: add a new list item

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

JavaScript Bookmarks (Favelets)

- ◆ Script stored by the browser as a bookmark
- ◆ Executed in the context of the current webpage
- ◆ Typical uses:
 - Submit the current page to a blogging or bookmarking service
 - Query a search engine with highlighted text
 - Password managers
 - One-click sign-on
 - Automatically generate a strong password
 - Synchronize passwords across sites



Must execute
only inside the
"right" page

Root-Kits

A ***rootkit*** modifies the user-program-accessible behavior of the operating system and escapes detection by interception of the operating system's reflection APIs

- E.g., removing itself from the operating system's list of running processes

JavaScript Rootkits?

A JavaScript “Rootkit”

[“Rootkits for JavaScript environments”]

```
if (window.location.host == "bank.com")  
    doLogin(password);
```

JavaScript bookmark

Malicious page defines a global variable named “window” whose value is a fake “location” object

```
var window = { location: { host: "bank.com" } };
```



Browsers let web pages override native objects to help with compatibility.

A malicious webpage

Let's Detect Fake Objects

["Rootkits for JavaScript environments"]

```
window.location = "#";
```

If window.location is a native object,
new value will be "https://bank.com/login#"

JavaScript bookmark

```
window.__defineGetter__("location",  
    function () { return "https://bank.com/login#"; });  
window.__defineSetter__("location", function (v) { });
```



A malicious webpage

Let's Detect Emulation

[“Rootkits for JavaScript environments”]

Use reflection API

```
typeof  
obj.__lookupGetter__(propertyName) !==  
"undefined"
```

typeof and !== avoid asking for the value of
“undefined” (could be redefined by attacker!)

JavaScript bookmark

Attacker emulates reflection API itself!

```
Object.prototype.__lookupGetter__ =  
function() { ... };
```



A malicious webpage

Defenses to Javascript Rootkits

- Store a short *master secret* in a Secure cookie for pwdmgr.com
- The bookmarklet initiates a network request to <https://pwdmgr.com> by adding a `<script>` tag to the current page
- Must authenticate the web site receiving the password
 - Referral header through HTTPS