Stony Brook University

CSE392/ISE331

Web Security Goals
&
Workings of the Web

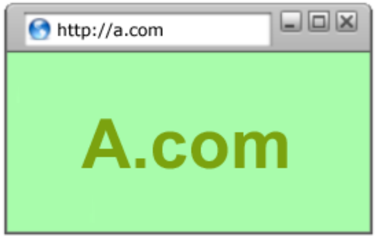
Nick Nikiforakis
nick@cs.stonybrook.edu

Goals of Web Security

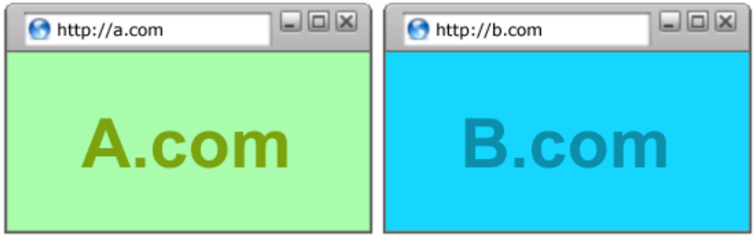
- Safely browse the Web
  - A malicious website cannot steal information from or modify legitimate sites or otherwise harm the user…
  - … even if visited concurrently with a legitimate site - in a separate browser window, tab, or even iframe on the same webpage
- Support secure Web applications
  - 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

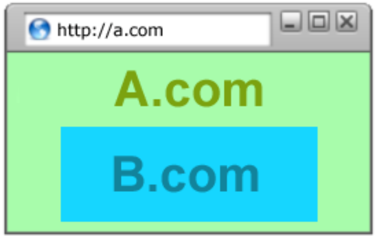
- Safe to visit an evil website



- Safe to visit two pages at the same time

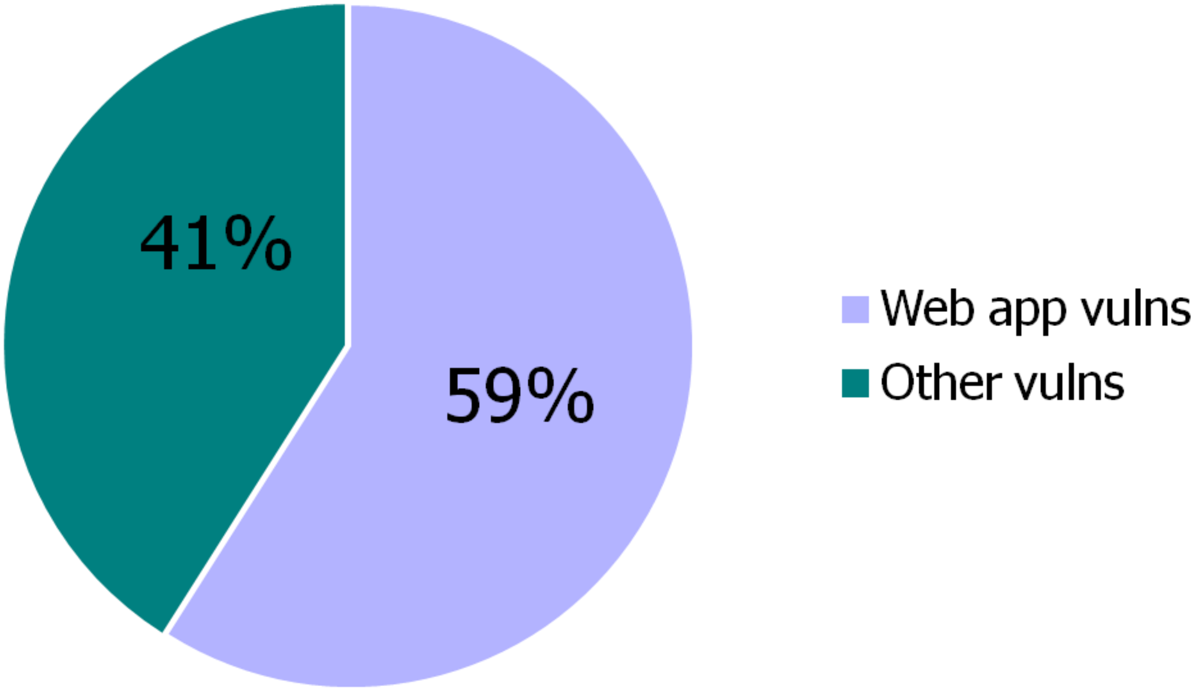


- Safe delegation



Security Vulnerabilities in 2011

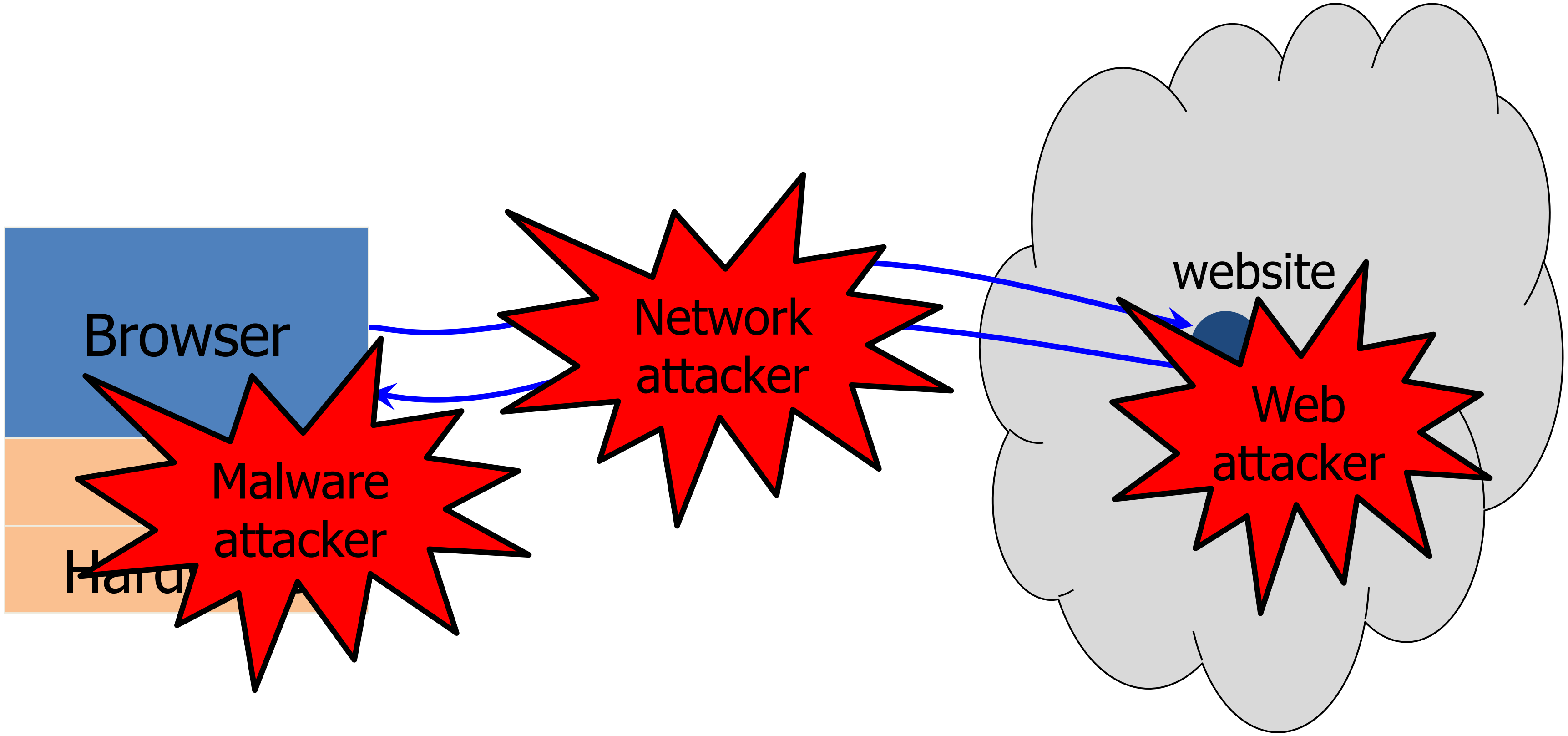
Source: IBM X-Force



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, CSRF, SQL injection

Where Does the Attacker Live?



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.

Dangerous Websites

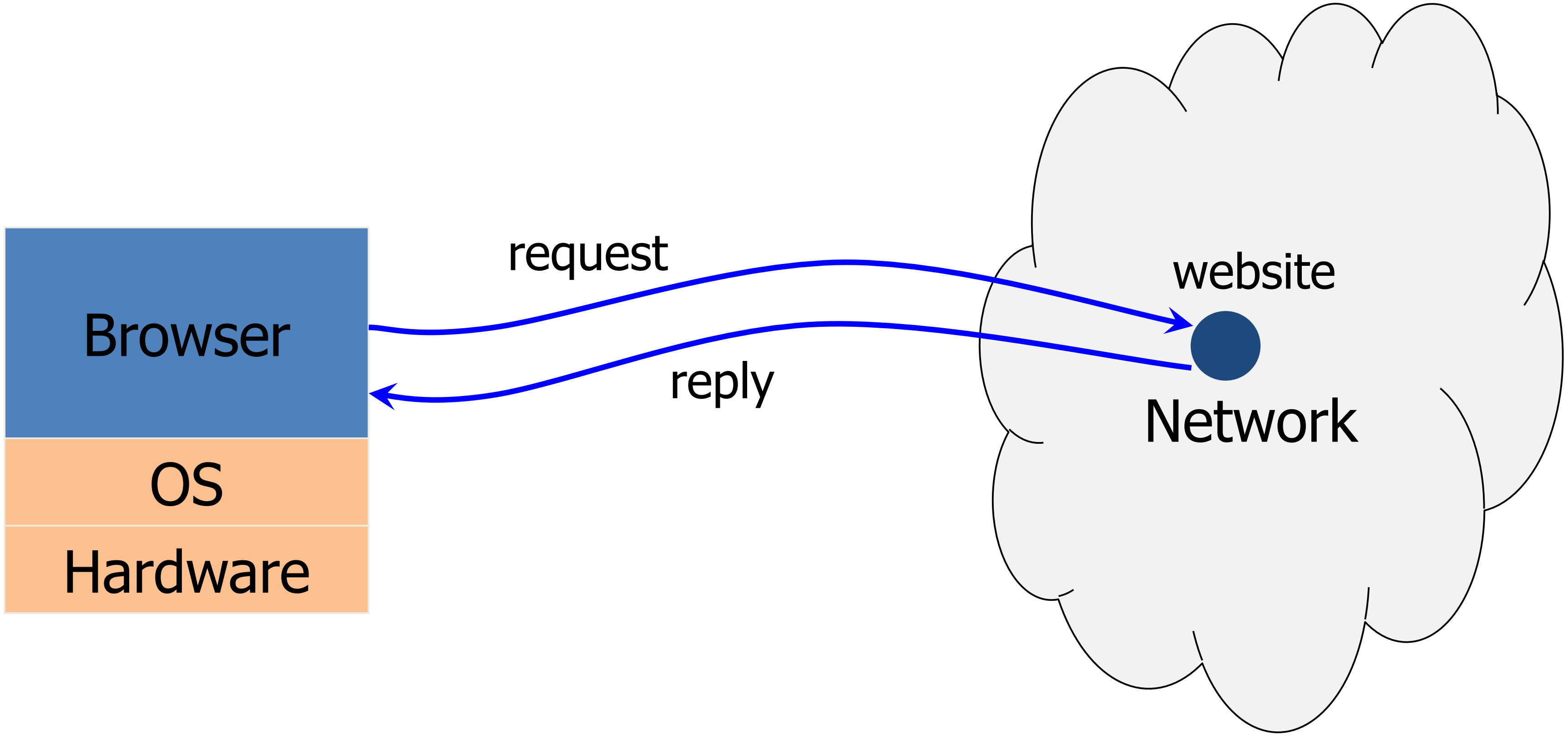
- Microsoft’s 2006 “Web patrol” study identified hundreds of URLs that could successfully exploit unpatched Windows XP machines
  - Many interlinked by redirection and controlled by the same major players
- “But I never visit risky websites”
  - 11 exploit pages are among top 10,000 most visited
  - 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

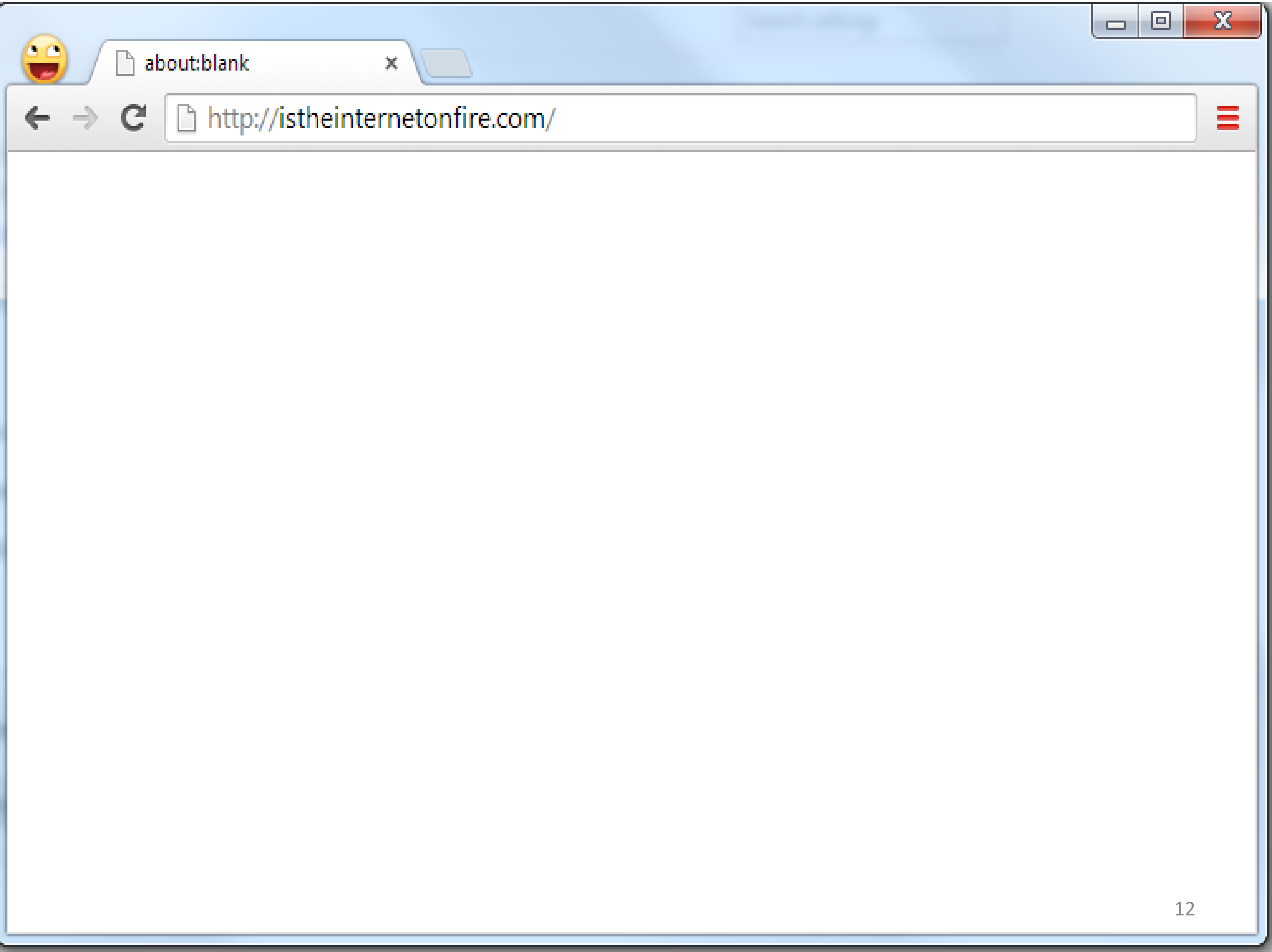
Before we break the web

- We must first understand how it works

- Questions that need to be answered:
  - How the browser works?
  - What happens when we type a URL and hit "Enter"?
  - How does Facebook remember who I am?
  - …

Browser and Network





http://istheinternetonfire.com/

DNS

- istheinternetonfire.com does not mean anything to a computer

- So first your browser needs to find the IP address belonging to that domain name
  - Exact DNS workings are outside the scope of this lecture (will come back to it in the future)
  - That said, the resolution of a domain name is an iterative procedure starting from your local machine potentially reaching to the DNS root servers that hold the Internet together

The answer from nslookup

```
 nslookup istheinternetonfire.com
Server:         97.107.133.4
Address:         97.107.133.4#53

Non-authoritative answer:
Name:   istheinternetonfire.com
Address: 166.84.7.99
```

Next step

- Now that your browser knows the address, it can open a socket to that IP address and start sending information

- What port is the webserver listening on?
  - By default port 80
- What kind of information do we send the server?
  - We send a request for the main page using the HTTP protocol and the GET method

HTTP request

```
GET / HTTP/1.1
Host: istheinternetonfire.com
Proxy-Connection: keep-alive
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/38.0.2125.104 Safari/537.36
Accept-Encoding: gzip,deflate,sdch
Accept-Language: en-US,en;q=0.8
```

HTTP Requests

- A request has the form:

```
<METHOD>  /path/to/resource?query_string  HTTP/1.1
<header>*

<BODY>
```

- HTTP supports a variety of methods, but only two matter in practice:
  - GET: intended for information retrieval
    - Typically the BODY is empty
  - POST: intended for submitting information
    - Typically the BODY contains the submitted information

HTTP response

```
HTTP/1.1 200 OK
Date: Tue, 21 Oct 2014 16:21:44 GMT
Server: Apache/2.2.25 (Unix) mod_ssl/2.2.25 OpenSSL/1.0.1h PHP/5.2.17
Last-Modified: Tue, 21 Oct 2014 15:37:09 GMT
ETag: "3aaa5c-850-505f09ab7f211"
Accept-Ranges: bytes
Content-Length: 2128
Content-Type: text/html

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html><head>

  <title>Is The Internet On Fire?</title>
  <meta http-equiv="content-type" content="text/html; charset=UTF-8">
  <link rev="made" href="mailto:jschauma@netmeister.org">
```

HTTP Responses

- A response has the form

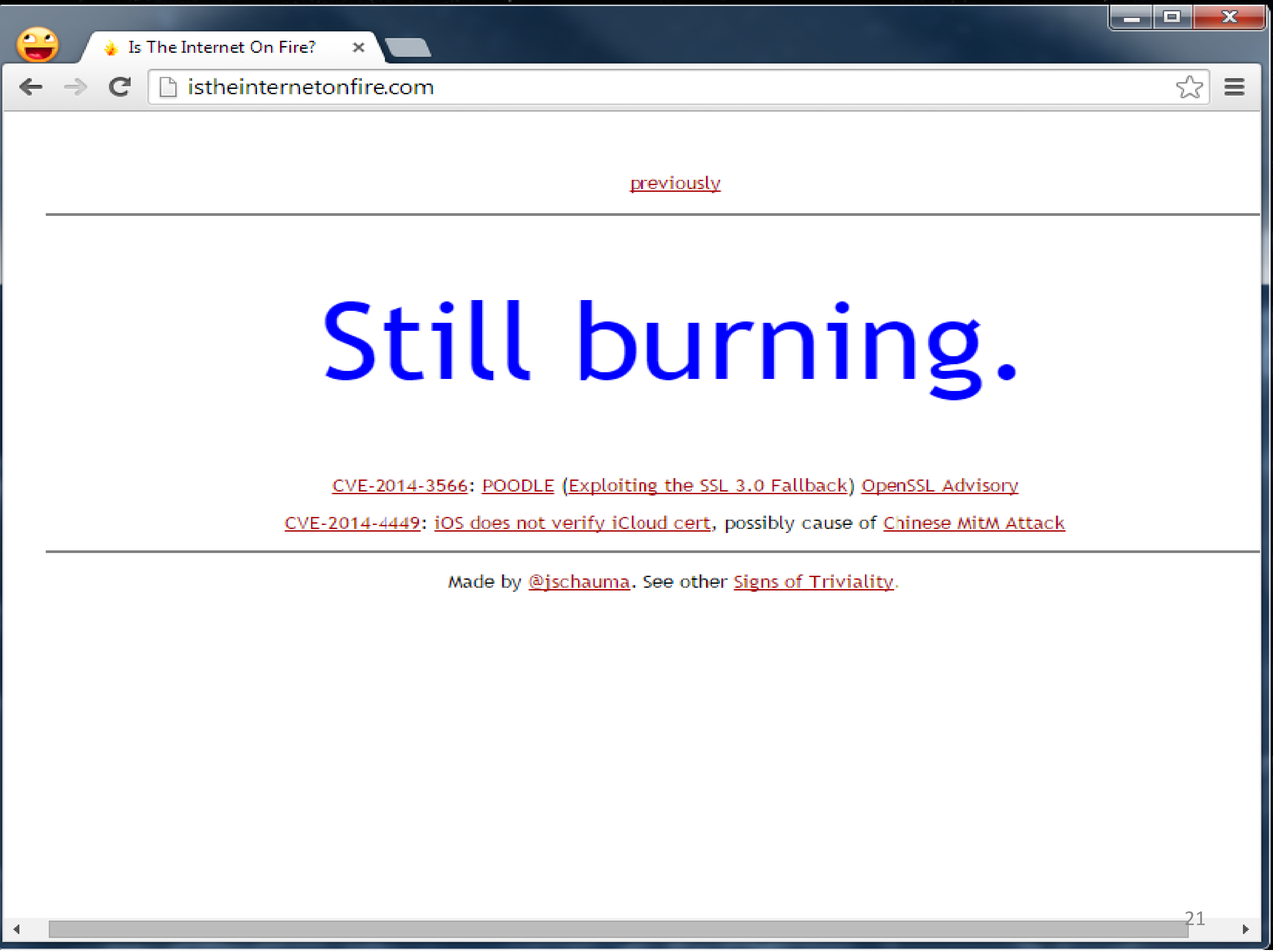
```
HTTP/1.1 <STATUS CODE> <STATUS MESSAGE>
<header>*

<BODY>
```

- Important response codes:
  - 2XX: Success, e.g. 200 OK
  - 3XX: Redirection, e.g. 301 Moved Permanently
  - 4XX: Client side error, e.g. 404 Not Found
  - 5XX: Server side error, e.g. 500 Internal Server Error

Browser consumption of response

- The browser gets the response and starts consuming it
  - Drawing on the screen according to the HTML code that was present in the response from the web server

- <u>Lalala</u>
- <hr>
- <a href=“http://a.com”> Cool site! </a>



previously

Still burning.

CVE-2014-3566: POODLE (Exploiting the SSL 3.0 Fallback) OpenSSL Advisory

CVE-2014-4449: iOS does not verify iCloud cert, possibly cause of Chinese MitM Attack

Made by @jschauma. See other Signs of Triviality.

Automatic loading of remote resources

- As the browser is parsing the HTML, whenever it finds a reference to a remote resource, it will **automatically** make a request to get it:

- Images
  - <img src=http://foo.com/a.jpg/>
- Cascading Style sheets
  - <link rel="stylesheet" type="text/css" href="default.css"/>
- Scripts (more on that later)
  - <script src=“http…”></script>
- Frames/iframes
  - <iframe src=“http…”></iframe>

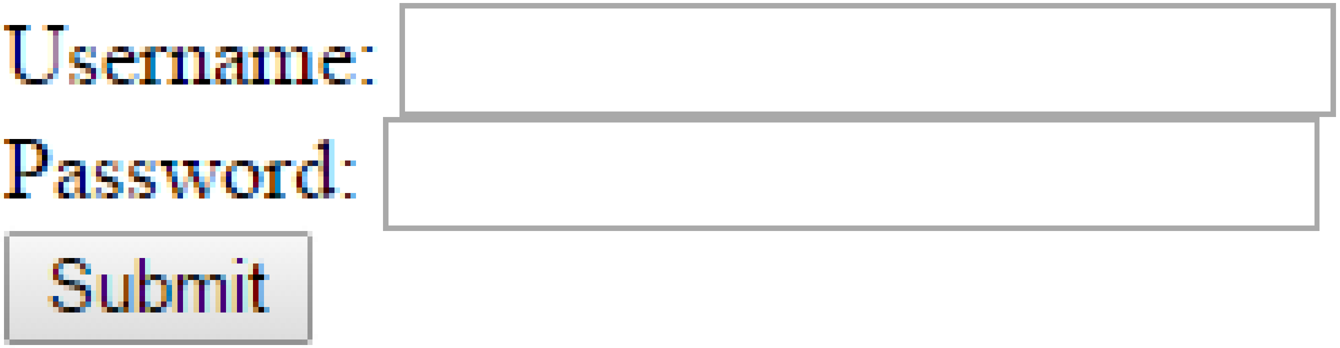
Where are we at?

- We can ask for pages, and we get back responses
- We can click on links, which will generate GET requests, and navigate around
- Question
  - How about personalization?
  - How does a site know that we are logged in?

Let’s look at a login form

```
<form method=“POST” action=“login.php”>
Username:
<input type=“text” name=“username”/>
Password:
<input type=“password” name=“password”/>
<input type=“submit”/>
</form>
```

Let’s look at a login form



- Let’s assume that the user is typing “admin” for username and “letmein” for a password

- The browser will emmit a “POST” request, as Instructed by the programmer

HTTP POST request

```
POST /login.php HTTP/1.1
Host: in.gr
Proxy-Connection: keep-alive
Content-Length: 31
Cache-Control: max-age=0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Origin: null
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/38.0.2125.104 Safari/537.36
Content-Type: application/x-www-form-urlencoded
Accept-Encoding: gzip,deflate
Accept-Language: en-US,en;q=0.8

username=admin&password=letmein
```

Server-side

- The webserver receives this request, passes it to a web application and then the web application checks to see whether such a user really exists
  - Typically in a database, present on the same machine or on other dedicated servers

- Assume that the username and password are correct. Now what?
  - We will give the proper page to the user (e.g. wall/list of emails/banking page,etc.)
  - How will we remember the user in the next request?

No state

- HTTP is, by design, stateless
  - There’s nothing baked in the protocol to identify one request as part of sequence of other requests

- You can try to do it by IP address, but that’s not going to work well
  - Network Address Translation
  - DHCP
  - Mobile devices

Let’s have some state

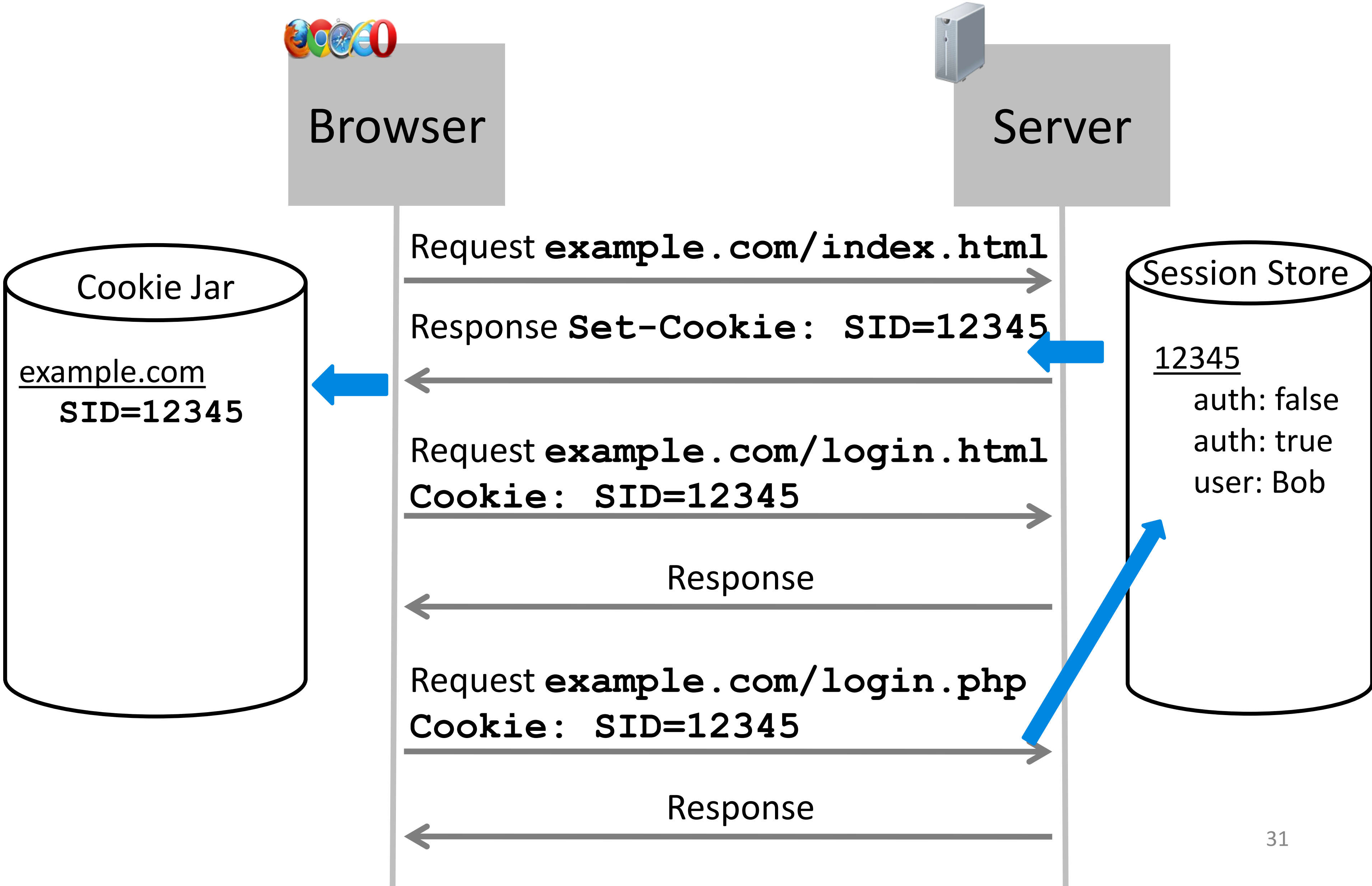
- There are more than one ways of introducing state, but the most popular (by far) is through the use of Cookies

- Cookie: A small piece of data sent by the webserver to browsers, which the browsers are to include to **all** their subsequent requests to that server.

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

Cookie-based Session Management



Browser

Server

Cookie Jar

example.com
SID=12345

Request **example.com/index.html**

Response **Set-Cookie: SID=12345**

Request **example.com/login.html**
Cookie: SID=12345

Response

Request **example.com/login.php**
Cookie: SID=12345

Response

Session Store

12345
auth: false
auth: true
user: Bob

Sessions

- As long as different users have different session identifiers (present in their cookies), the web server will be able to tell them apart
  - Regardless of their IP address
- When users delete their cookies, the browsers no longer send out the appropriate session identifier, and thus the web server “forgets” about them

Session Identifiers

- Long pseudo-random strings
- Unique per visiting client
- Each identifier is associated with a specific visitor
  - ID A -> User A
- As sensitive as credentials (per session)

One missing piece

- We can create websites
- And we can have state, enabling us to have a personalized web
  - Banking ,Email, Social networks, etc.
- But our pages are still static
  - The server sent some HTML, the browser drew it on the screen, and that’s it

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

```
<a href="http://www.yahoo.com"
  onmouseover="alert('hi');">
```

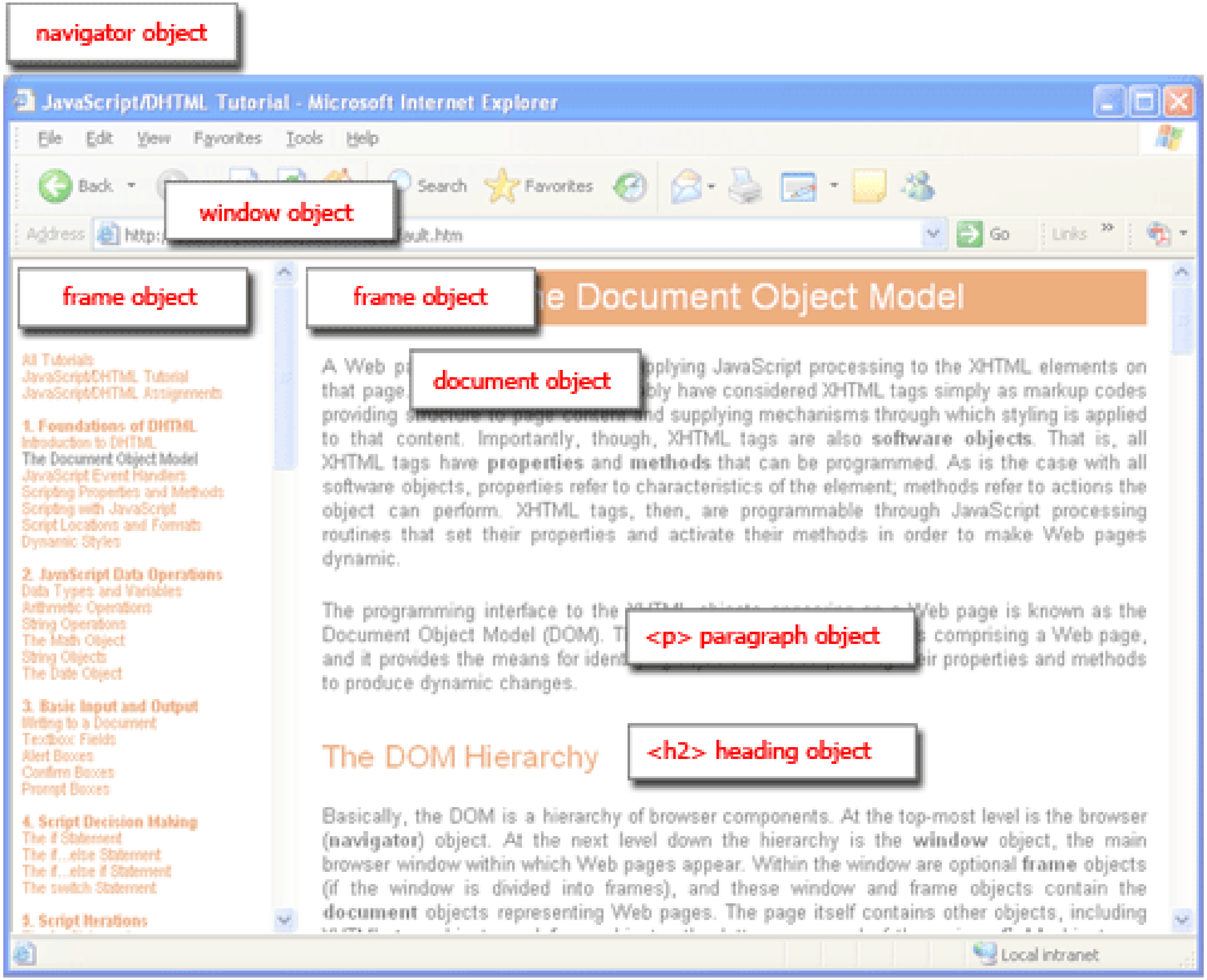
- Pseudo-URL referenced by a link

```
<a href=“JavaScript: alert(‘You clicked’);”>Click me</a>
```

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



navigator object

window object

frame object

frame object

document object

<p> paragraph object

<h2> heading object

All Tutorials
JavaScript/DHTML Tutorial
JavaScript/DHTML Assignments

1. Foundations of DHTML
Introduction to DHTML
The Document Object Model
JavaScript Event Handlers
Scripting Properties and Methods
Scripting with JavaScript
Script Locations and Formats
Dynamic Styles

2. JavaScript Data Operations
Data Types and Variables
Arithmetic Operations
String Operations
The Math Object
String Objects
The Date Object

3. Basic Input and Output
Writing to a Document
TextBox Fields
Alert Boxes
Confirm Boxes
Prompt Boxes

4. Script Decision Making
The if Statement
The if...else Statement
The if...else if Statement
The switch Statement

5. Script Iterations

e Document Object Model

A Web pa pplying JavaScript processing to the XHTML elements on that page. bly have considered XHTML tags simply as markup codes providing d supplying mechanisms through which styling is applied to that content. Importantly, though, XHTML tags are also **software objects**. That is, all XHTML tags have **properties** and **methods** that can be programmed. As is the case with all software objects, properties refer to characteristics of the element; methods refer to actions the object can perform. XHTML tags, then, are programmable through JavaScript processing routines that set their properties and activate their methods in order to make Web pages dynamic.

The programming interface to the Veb page is known as the Document Object Model (DOM). T comprising a Web page, and it provides the means for iden ir properties and methods to produce dynamic changes.

The DOM Hierarchy

Basically, the DOM is a hierarchy of browser components. At the top-most level is the browser (**navigator**) object. At the next level down the hierarchy is the **window** object, the main browser window within which Web pages appear. Within the window are optional **frame** objects (if the window is divided into frames), and these window and frame objects contain the **document** objects representing Web pages. The page itself contains other objects, including

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 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 "

Page Manipulation with JavaScript

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

Sample HTML

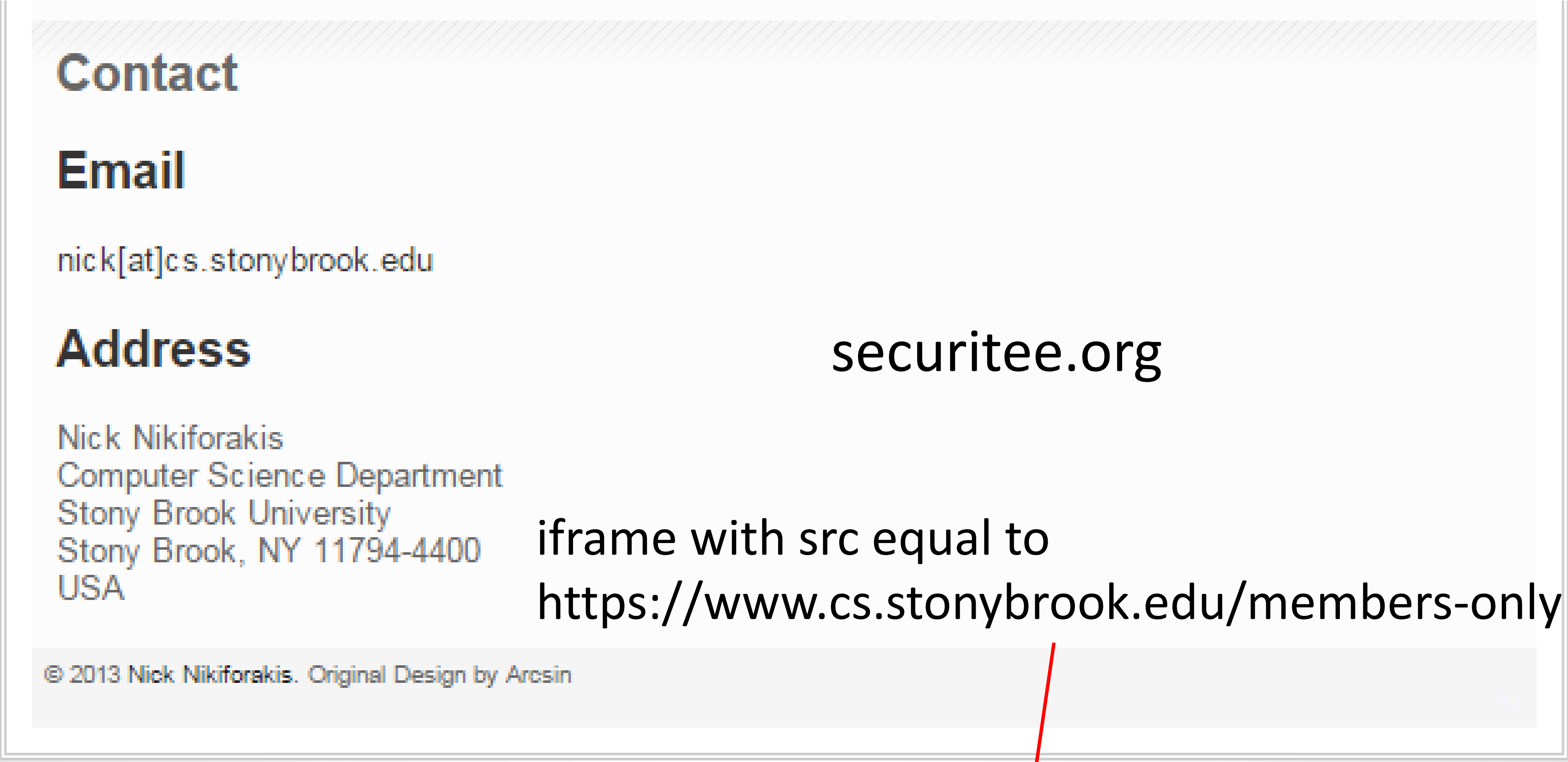
```
<ul id="t1">
<li> Item 1 </li>
</ul>
```

All the functional pieces are in place

- Now we can create personalized and dynamic websites. Yay!

- But what about security?
  - How do we stop websites from snooping around in each other’s business?

- An example with frames



Contact

Email

nick[at]cs.stonybrook.edu

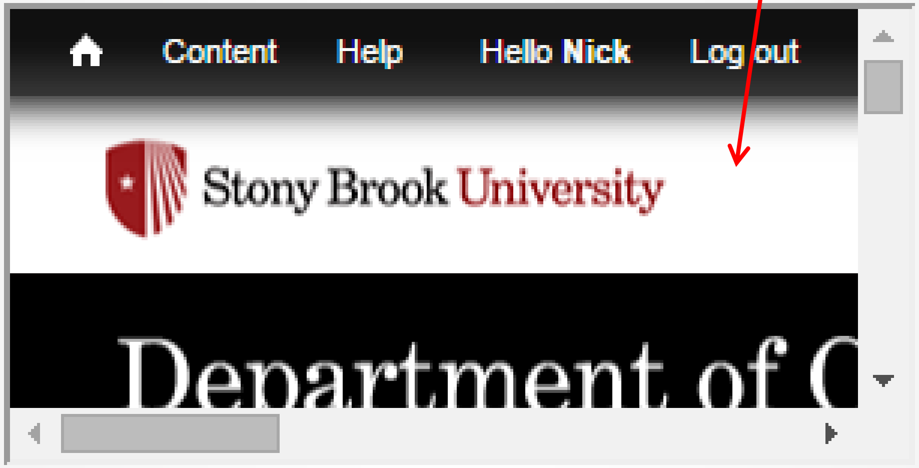
Address

Nick Nikiforakis
Computer Science Department
Stony Brook University
Stony Brook, NY 11794-4400
USA

© 2013 Nick Nikiforakis. Original Design by Arcsin

securitee.org

iframe with src equal to https://www.cs.stonybrook.edu/members-only



Problems?

- If there are no restrictions, securitee.org could use the DOM to dive into https://www.cs.stonybrook.edu/members-only and
  1. Extract details
  2. Make requests in the name of the user
  3. Inspect the responses

Content in the Browser

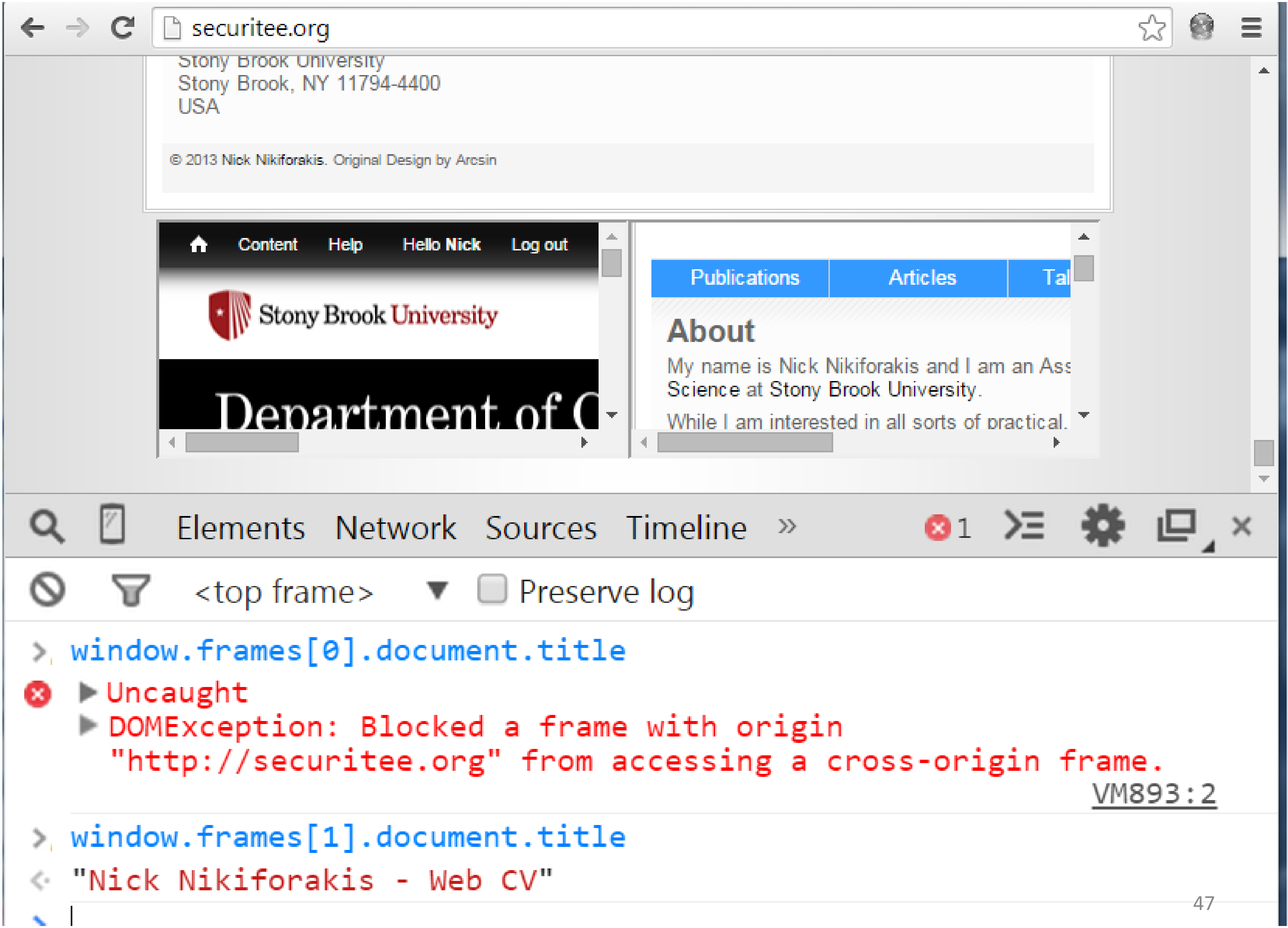
- Origin-based separation of documents
  - Naturally enforced by the Same-Origin Policy
  - Allows you to separate sensitive parts and non-sensitive parts
  - Prevents unintended sharing of information
  - Prevents escalation of successful attack

ORIGIN

The triple <scheme, host, port> derived from the document's URL. For *http://example.org/forum/*, the origin is *<http, example.org, 80>*

SAME-ORIGIN POLICY

Content retrieved from one origin can freely interact with other content from that origin, but interactions with content from other origins are restricted



Stony Brook University
Stony Brook, NY 11794-4400
USA

© 2013 Nick Nikiforakis. Original Design by Arcsin

About

My name is Nick Nikiforakis and I am an Ass
Science at Stony Brook University.

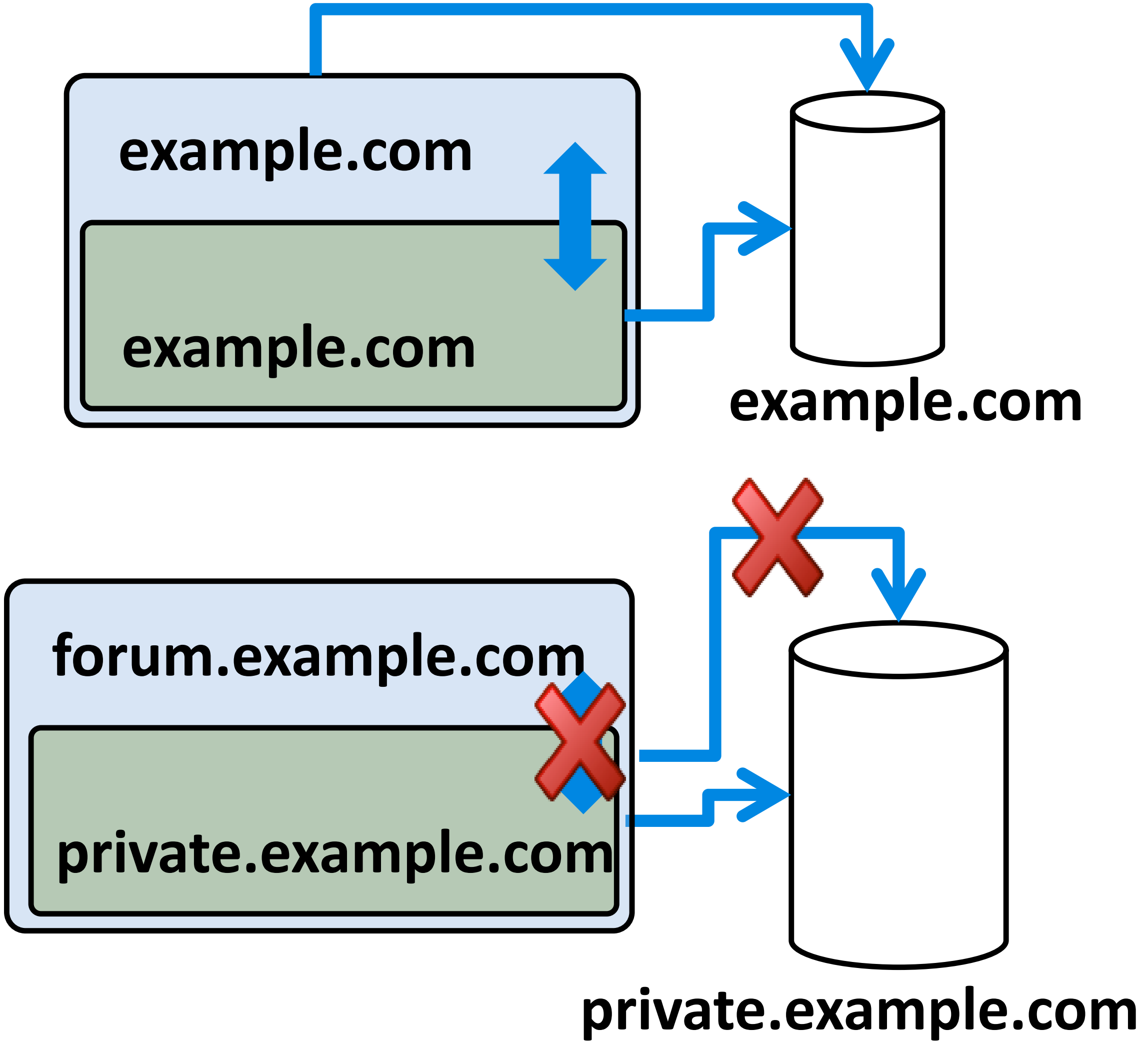
While I am interested in all sorts of practical

```
> window.frames[0].document.title
✖ ▶ Uncaught
  ▶ DOMException: Blocked a frame with origin
    "http://securitee.org" from accessing a cross-origin frame.
                                                        VM893:2
> window.frames[1].document.title
< "Nick Nikiforakis - Web CV"
```

Examples of the Same-Origin Policy

SAME-ORIGIN POLICY

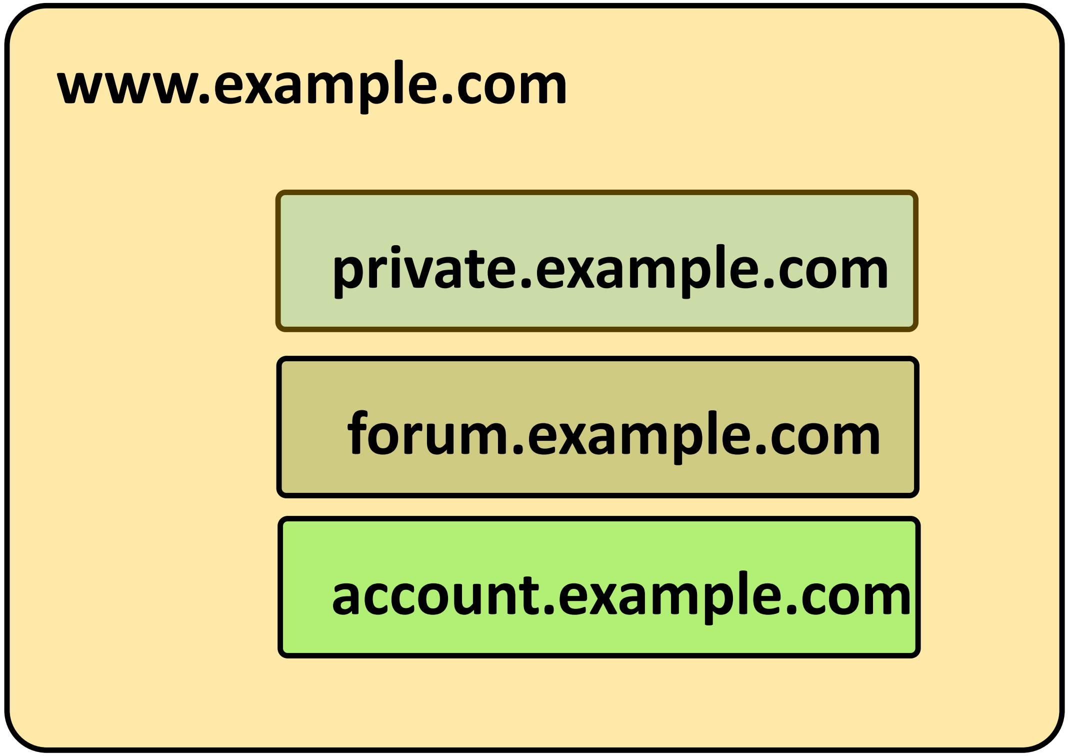
Content retrieved from one origin can freely interact with other content from that origin, but interactions with content from other origins are restricted



Domains vs Subdomains

- Subdomains
  - E.g. ***private.example.com*** vs ***forum.example.com***
  - Considered different origin
  - Possibility to relax the origin to ***example.com*** using *document.domain*
  - Possibility to use cookies on ***example.com***
- Completely separate domains
  - E.g. ***private.example.com*** vs ***exampleforum.com***
  - Considered different origin, without possibility of relaxation
  - No possibility of shared cookies

Subdomains and Domain Relaxation



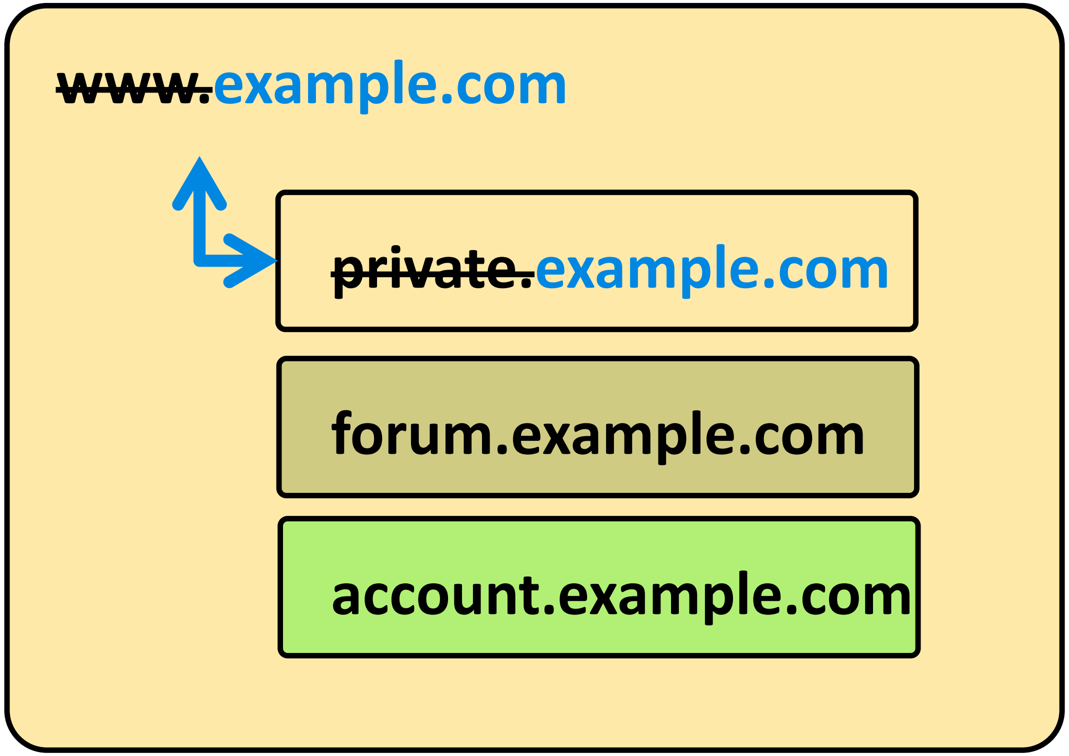
www.example.com

private.example.com

forum.example.com

account.example.com

Subdomains and Domain Relaxation



~~www.~~example.com

~~private.~~example.com

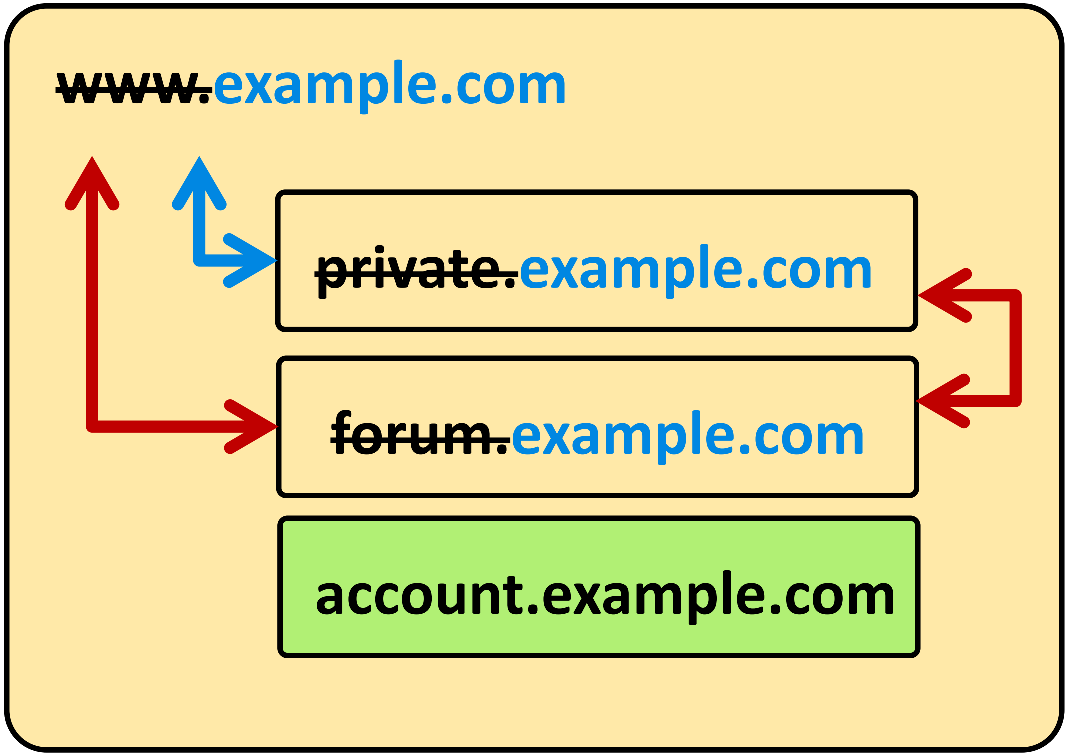
forum.example.com

account.example.com

DOMAIN RELAXATION

```
document.domain = "example.com";
```

Subdomains and Domain Relaxation



DOMAIN RELAXATION

```
document.domain = “example.com”;
```

So, what’s left?

- Same-Origin Policy has our backs, right?
  - It will stop **attacker.com** from looking into the DOM, requests, and responses.
  - No malicious website can steal a user’s data, right?

- Wrong!