Web Database Security

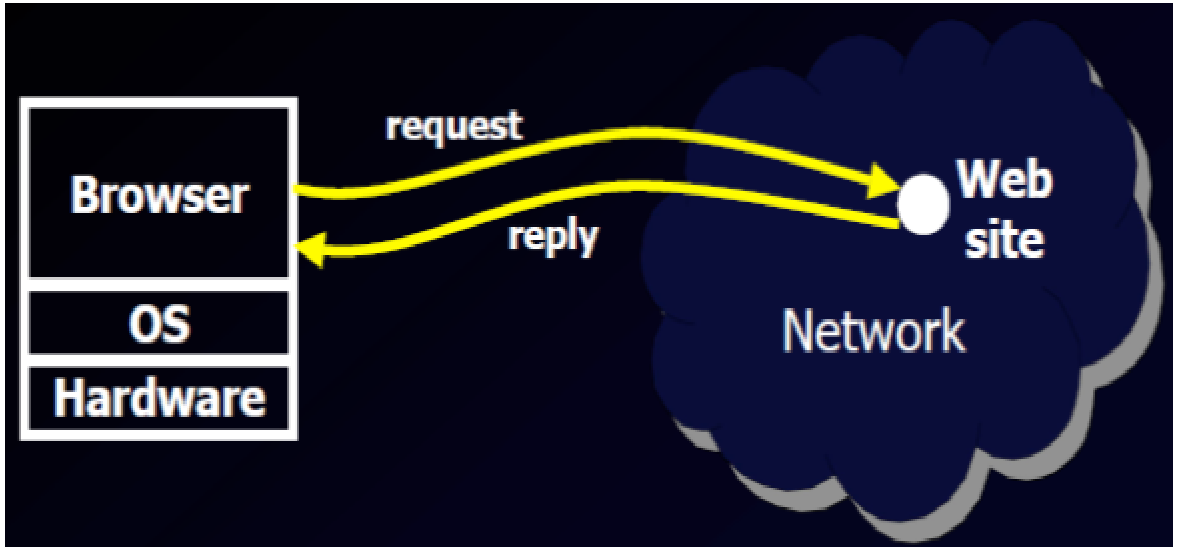
Notes-set7 Web Application Security

Web Applications

- Background on how web applications are typically developed,
- How they function, and
- The platform that they execute on

Web Applications

- Web applications are multi-tiered applications that **consist of code on a client**, which is **operated by the end-user who runs a browser**, and **a server that is operated by the web application owner.**



Hypertext Transfer Protocol

- The client and server components of a web application communicate with each other using Hypertext Transfer Protocol, often referred to as HTTP
- HTTP dates back to the 1980s, and allows the client to request a resource specified by a Uniform Resource Location, or URL.
- A URL consists of the following elements: **scheme:[//[user[:password]@]host[:port]][/path][?query][#fragment]**
- **Scheme.** A URI scheme specifies the protocol that should be used to look up the resource. The most common schemes on the web are http and https, but you have probably also encountered ftp, mailto, file, and irc.
- **User and password.** Optionally, a URI can contain a username and password for resources that require password authentication.
- **Host.** The host can be a registered name (e.g., google.com or andrew.cmu.edu) or an IP address (e.g., 28.2.42.10).
- **Port.** The network port number on which the resource can be accessed. Ports are given by 16-bit integers, and common numbers are 80 (for http resources), 443 (https), and 143 (IMAP email protocol).
- **Path.** The path component is a hierarchical, slash-separated string that typically resembles a file path.
- **Query.** Non-hierarchical data that is typically used to pass arguments to the server side of a web application in the form of key-value pairs.
    - For example, the query string q=NITK is used by Google's search engine to specify the search query, so navigating to google.com/search?q=NITK will return the search results for query "NITK".
- **Fragment.** The fragment portion of a URI can specify a secondary resource, such as a section heading within a page or a location somewhere in the middle of a file.

Hypertext Transfer Protocol

- An HTTP session is a sequence of request-response transmissions between a client-server pair. The server listens for connections on a well-known TCP port, typically 80 or 443.
- On receiving a connection and subsequent request, the server responds with astatus code, and barring an error, a message with the contents of the requested URL.
- **Requests and responses.** There are several types of requests that a client can make of a server. Two that are most often encountered in typical browser sessions are **GET** and **POST**.

Hypertext Transfer Protocol

- A **GET** request asks the server for the contents of the specified URL, and nothing else. Importantly, this request should have no other effect on the requested URL.
- A **POST** request asks the server to accept some data contained in the request for processing or annotation of the URL.
  - For example, these requests are commonly used to transmit the information entered by a user in a web form to the server
- Other types of requests include **HEAD** (return the header of a response without the body), **PUT** (store the request contents under a specified URL, or create the URL if it does not exist), **DELETE** (remove the specified resource), and **OPTIONS** (query the server for implemented methods).
- In addition to these methods, an **HTTP request may contain a number of headers** that provide additional information.
- The Internet Engineering Task Force (IETF) has standardized a number of header fields, and there are many more that are commonly used and supported by modern browsers.

HTTP

Method **File** HTTP version Headers

```
GET /default.asp HTTP/1.0
Accept: image/gif, image/x-bitmap, image/jpeg, */*
Accept-Language: en
User-Agent: Mozilla/1.22 (compatible; MSIE 2.0; Windows 95)
Connection: Keep-Alive
If-Modified-Since: Sunday, 17-Apr-96 04:32:58 GMT
```

Blank line

HTTP version **Status code** **Reason phrase**

Data – none for GET

Headers

```
HTTP/1.0 200 OK
Date: Sun, 21 Apr 1996 02:20:42 GMT
Server: Microsoft-Internet-Information-Server/5.0
Connection: keep-alive
Content-Type: text/html
Last-Modified: Thu, 18 Apr 1996 17:39:05 GMT
Content-Length: 2543

<HTML> Some data... blah, blah, blah </HTML>
```

Data

Example

```
GET index.html HTTP/1.1
Accept: text/html image/gif, image/x-bitmap, image/jpeg
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
Host: cmu.edu
Connection: keep-alive
```

Figure 1: Example HTTP **GET** request.

```
HTTP/1.1 200 OK
Date: Tue, 10 April 2018 22:38:34 GMT
Content-Type: text/html; charset=UTF-8
Content-Encoding: UTF-8
Content-Length: 138
Last-Modified: Mon, 09 April 2018 23:11:55 GMT
Server: Apache/1.3.3.7 (Unix) (Red-Hat/Linux)

<html>
<body>
  Hello World
</body>
</html>
```

Figure 2: Example HTTP response.

Maintaining state

- Most web applications are stateful, and account for the user’s interaction with a sequence of URLs across the span of a session.
- Examples of stateful behavior include authenticating users, maintaining shopping carts, remembering preferences, and tracking user behavior.
- However, the basic HTTP protocol is stateless, so some additional data **in the form of a cookie** is needed to associate requests with sessions.
- **A cookie is a small piece of data that is sent by a user’s browser to a web server**. Cookies are associated with a domain and path, so that all requests that match these elements result in the browser sending the cookie along with a request.
- Cookies can be set in HTTP responses, or from JavaScript code running in the browser.
- Although the data stored in cookies can be arbitrary (but limited in size), most of the time **they contain unique identifiers that tell the server who the user is**.

Maintaining state

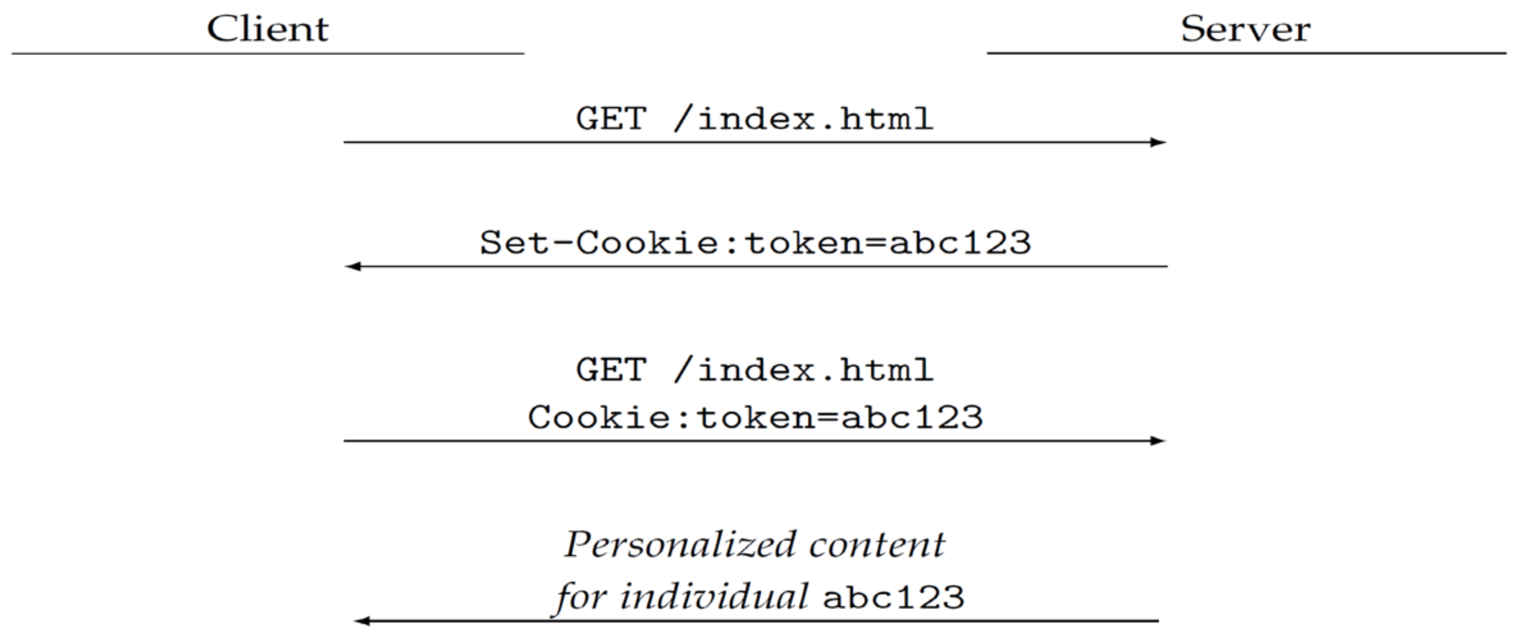
- Cookies enable applications that first authenticate users with a login challenge (e.g., apps that request a username and password).
- When the user visits the login site, they complete a form containing username and password.
- These are sent to the server-side portion of the app (e.g., using URL parameters over an encrypted connection), which checks the provided credentials against a back-end database. If the correct credentials were provided, then **the server responds by sending a response containing a session cookie** that the server-side app associates with the successfully-authenticated user.
- They can then navigate the website without having to provide credentials each time they need to view protected content.

Cookies

- There are two types of cookies: **persistent cookies** and **session cookies**
- **Persistent cookies** are stored in the browser permanently (although most browsers allow users to delete them whenever they want), and are used to track users across multiple sessions including ones that span browser shutdown.
  - Persistent cookies have a specified expiration date, after which the browser will automatically delete them.
- **Session cookies** are ephemeral, and reside in the browser's memory only for as long as the user navigates the website.
  - Session cookies have no expiration date, and disappear once the user closes the browser tab or navigates away from the website.

Simplified sequence of requests and responses to establish and utilize a cookie

- If an HTTP request does not contain a cookie header, then the server responds by setting a cookie that is remembered by the browser and included in future requests



Web apps on the client

- The **client side of a web application is run inside of a browser**, which is a native program that makes requests to remote servers, downloads the code and associated data of the web app, and renders it in a graphical interface displayed to the user
- The browser consists of a main process that manages the user interface, the (potentially) multiple tabs, and and plugins (also called extensions). This is referred to as the "browser process" or sometimes simply the browser.
- Each open tab corresponds to a separate renderer process that is ultimately responsible for running the web application code and deciding how the results should be displayed in the user interface. Finally, each tab can contain resources fetched from different sources, called frames.
  - For example, websites often include frames for advertisements, where the contents of an advertising frame come from a different domain than that of the main content.

HTML & CSS

- In the early days of the web (“Web 1.0”), the content rendered by browsers was static in the sense that it did not change and for the most part did not respond to user input.
- Web applications, which were nothing more than collections of web pages, consisted entirely of Hypertext Markup Language (HTML) documents that browsers used to interpret and compose text, images, and other content visible in the browser window.
- **HTML documents specify the structure of a web page by specifying the grouping and relative layout of a set of HTML elements**. The elements correspond to entities such as the title, header, paragraphs, and images of a page. Syntactically, elements are specified by tags given in angle brackets with tag names and attributes.
- Around the same time that HTML was proposed, **Cascading Style Sheets (CSS)** were proposed as a clean way to separate the content of a web page from its presentation.
- A CSS document specifies how the elements in an HTML document should be rendered, including aspects of layout, sizing, font, and coloring. An HTML document associates itself with a given CSS by specifying it in a tag.
- By separating content and presentation in this way, it is possible for a single HTML document to render appropriately on multiple types of devices or in several different modes.
    - For example, most websites today specify different style sheets for desktop and mobile browsers to account for differences in form factor. CSS also makes it possible for multiple HTML documents to share the same presentation style, thus eliminating redundancy that would otherwise need to exist in the HTML tag attributes and simplifying the process of changing aspects of presentation.

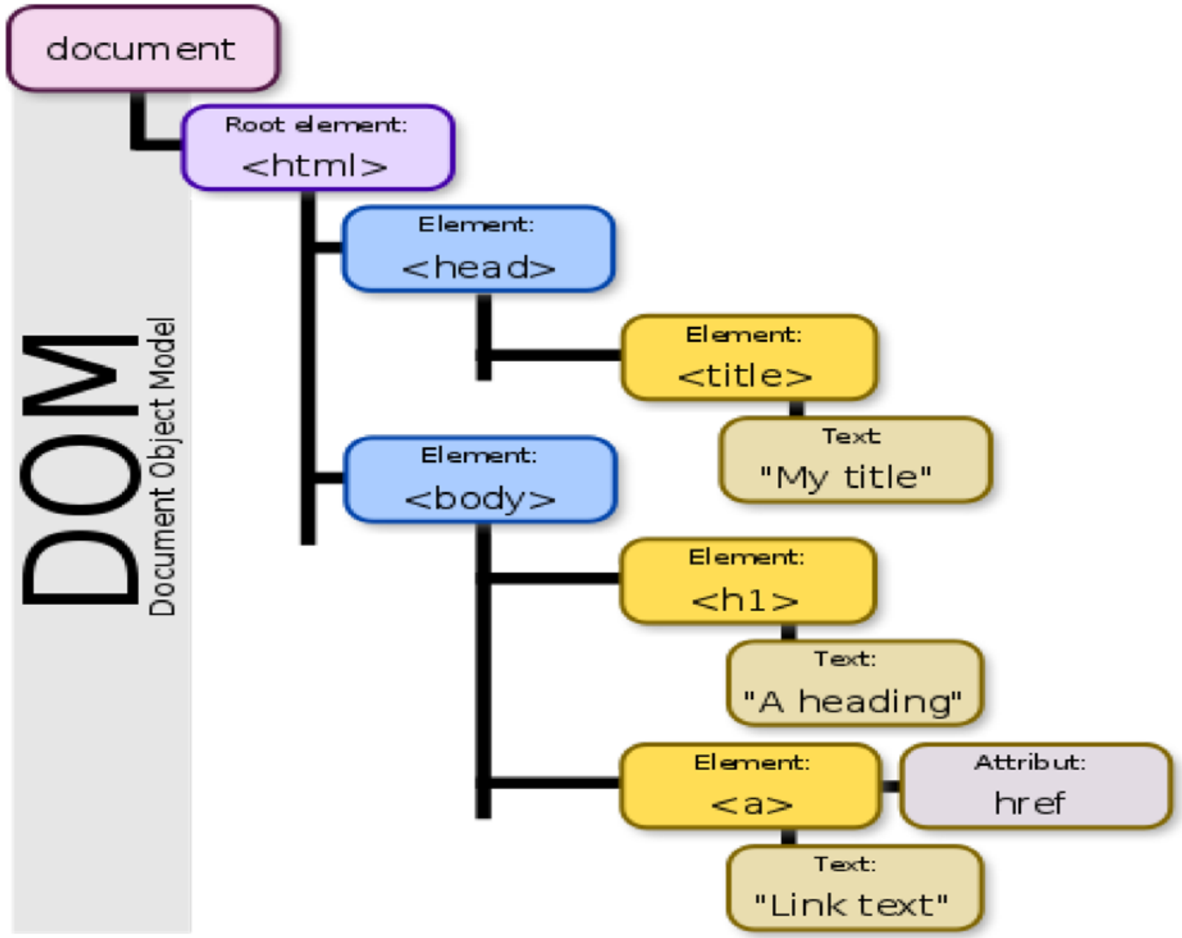
JavaScript

- Allowing them to run scripts in the context of a rendered HTML document.
- It supports APIs for dealing with strings, arrays, dates, regular expressions, and rendered HTML content (more on this later), but contains only limited facilities for I/O.
- JavaScript contains a number of unfortunate features that make it difficult to reason about both for developers and for automated tools that support safety and correctness.
- The most famous such feature is the **eval(str) function**, which takes a string argument and evaluates it as a JavaScript program. This allows programs to dynamically compute programs and run them, which causes obvious problems for static analysis.
- Another difficulty comes from **JavaScript's type coercion**, wherein data of a particular type is converted to a different type automatically

Document Object Model

- One of the key motivations for incorporating a scripting language into web pages is to allow scripts to update the rendered content programmatically and in response to events such as user interaction.
- This is supported by the **Document Object Model (DOM),** which is an API for reading and manipulating parsed HTML content.
- The DOM is actually **a language-independent API**, but we will focus on its implementation in JavaScript as it is the most relevant to web app security.
- The DOM maintains **an internal representation of an HTML document as a tree structure**. Each node corresponds to an element specified in the HTML, and the root of the tree (called the “document object”) corresponds to the top-level document containing all of the elements.
- The JavaScript program running in the context of a page can make arbitrary changes to the DOM, which the browser will then render back on the UI.
- By extension, the DOM API allows JavaScript programs to register event handler callback functions on specified elements.
- This allows websites to implement dynamic content in many ways resembling traditional desktop applications. As the user interacts with rendered elements, JavaScript event handlers are invoked which can in turn change the layout and visual appearance of the page. The combination of these technologies—HTML, JavaScript, and the DOM—are the essential client-side elements of modern web applications.

DOM



: An example of the DOM hierarchy in a simple HTML document. Image source: https://commons.wikimedia.org/wiki/File:DOM-model.svg.

Same-origin policy

- Because websites can execute scripts, there is the possibility that malicious websites could use this functionality to interfere with or spy on the content and interactions of other websites
    - For example, you may click on an untrusted link at the same time that you have a website from your bank or healthcare provider open. It would be problematic if a script running on the untrusted site were able to use the DOM API to snoop on sensitive information displayed on either page, or worse yet, invoke event handlers that cause requests to your bank!
- **To protect the secrecy and integrity of web application content and data, browsers implement the Same-origin Policy (SOP).**
- Roughly, the SOP requires that content loaded in a web page (call it “website A”) can not access data or functionality on another website (“website B”) unless A and B have the same origin.
- **An origin is defined as the combination of the URI scheme, host name, and port number from with the site was loaded**.
- Scripts running under the label for a particular (scheme, host, port) triple are not allowed to access the DOM content or script state of pages running under different triples. This is a type of information flow policy that isolates content from different sources.

Cross-origin access and embedded content

- Websites often consist of content from multiple different domains.
- One common example is **images**, which are sometimes stored on severs from different origins.
- Similarly, JavaScript code and CSS documents are frequently provided as **libraries** and there are good performance reasons for a website to use the source stored on the library vendor's servers.
- Finally, **frames** subdivide the browser window into segments with content loaded from possibly unrelated URIs; this is one common way of displaying advertisements.
- Although these do not necessarily have anything to do with JavaScript code or DOM access, these forms of embedded content seem to violate the intent of the Same-origin Policy in that they allow information to flow between different origins.
  - For example, when a page from origin A loads an image from origin B, information can leak from A to B both through the simple fact that a request is being made, as well as through the pathname of the requested image. Likewise, information can leak from B to A through any error messages that may occur (e.g., if B returns that the content is inaccessible), and through attributes such as the width and height of the returned image
- Regardless, embedding is typically allowed as it is considered essential to supporting fully-functional web applications.

Cross-origin access and embedded content

- A few case-specific details are with noting though:
  - For scripts that are embedded from outside origins with a <script src=...> tag, the code is retrieved, parsed, and then run in the context of the origin that requested the script (not the origin that it was retrieved from). This supports crossorigin third-party libraries while partially avoiding explicit cross-origin flows. Moreover error messages are only returned for scripts from the same origin as the embedding page.
  - Content embedded in an iframe can come from arbitrary domains, but is run in the context of the origin that the content is loaded from. So if a web page in origin A embeds and iframe with an advertisement from origin B, the ad will run in the context of B and the Same-origin Policy will apply more or less as though the content were loaded in a separate tab.
  - Images from different domains can be loaded and displayed on a web page, but the contents of the image itself, i.e. its pixels, cannot be read by the page loading the image.
  - Cookies use a separate definition of origins. A page can set a cookie for its own domain or any parent domain, as long as the parent domain is not a public suffix (there are only a small number of these). The browser will make a cookie available to a page in the cookie's domain, including any subdomains, no matter which protocol or port is used. Cookies can be further scoped by setting the path, thus limiting the pages to which the cookie is sent to be within specified path.
- Note: most of these SOP "exceptions" as being governed by the principle that the SOP limits the ability of pages to read content from other domains, **but not their ability to send data to other domains**.

Web apps on the server

- Web applications are “tiered” into portions consisting of the client side and server side.
- On the server tier, a common architecture divides the application into components that are responsible for the “application logic” and data storage separately.
- The application logic component takes care of network communications with the client, doing the core work of receiving requests, computing responses, and sending them back to the client.
- The data storage component is typically a traditional relational database backend that contains any data needed to service client requests.
- For traditional “static” websites, the database may not play much of a role if HTML documents are pre-generated and stored on the server’s file system. In this case, the server just parses the URI received from the client and uses the path component to locate an appropriate HTML file to send back. But it should come as no surprise that the vast majority of web applications require more than this.

Server-side scripting

- Just as client-side scripting supports dynamic content, serverside scripting languages are commonly used to generate dynamic content to send to the client in response to the details of a request.
- While on the client side JavaScript is almost exclusively used, there is no de-facto standard server-side scripting and there are several common approaches for dynamically generating content on this tier.
- Among the most common are the **Common Gateway Interface (CGI) and PHP**.
- **CGI is still used in web applications**, but it is much more common to implement dynamic server-side functionality in a language like PHP.
- **PHP in some ways simplifies the dynamic creation of HTML by allowing developers to write a template HTML document with portions that contain embedded PHP code**. When the client requests a PHP file, the web server invokes the PHP interpreter passing the client's arguments, and the embedded PHP elements are evaluated into strings within the HTML template.
  - The end result is an HTML document corresponding to the original template, with specific portions having content that was dynamically computed based on the client's request.

Problem Parameters

- HTTP is a stateless protocol
  - Each request is independent of previous request
- Servers have little information about where a request comes from
- Web site software is extremely loosely coupled
  - Coupled through the Internet – separated by space
  - Coupled to diverse hardware devices
  - Written in diverse software languages

Separation of Concerns in Web Apps

- Presentation layer $\longrightarrow$ HTML, output and UI

- Data content layer $\longrightarrow$ Computation, data access

---

- Data representation layer $\longrightarrow$ In-memory data storage

---

- Data storage layer $\longrightarrow$ Permanent data storage

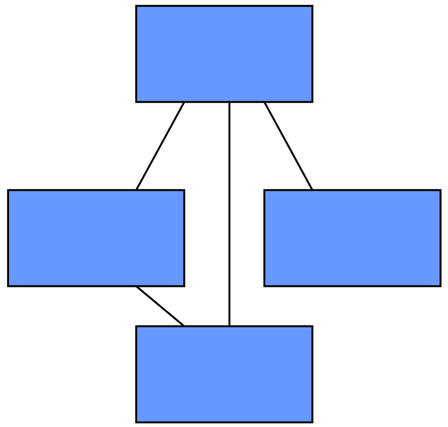
Differences in Testing Web Software

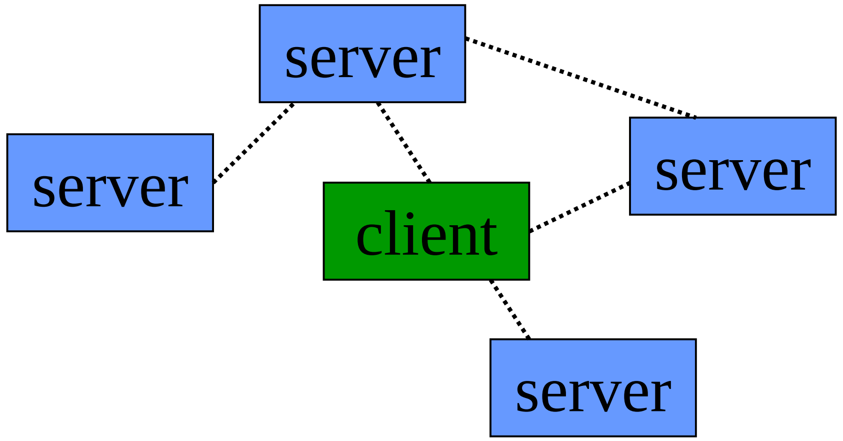
- Traditional graphs do not apply
  - Control flow graph
  - Call graph
- State behavior is hard to model and describe
- All inputs go through the HTML UI – low controllability
- Hard to get access to server-side state (memory, files, database) – low observability
- Not clear what logic predicates can be effectively used
- No model for mutation operators on web software

New Essential Problems of Web Apps

1. Web site applications feature distributed integration and are extremely loosely coupled
   - Internet and diverse hardware / software
2. HTML forms are created dynamically by web applications
   - UI created on demand and can vary by user and time
3. Users can change the flow of control arbitrarily
   - back button, forward button, URL rewriting, refresh
4. Dynamic integration of new software components
   - new components can be added during execution

Problem 1: Loosely Coupled





<u>**Traditional software**</u>

Connected by calls and message passing

High and moderate coupling

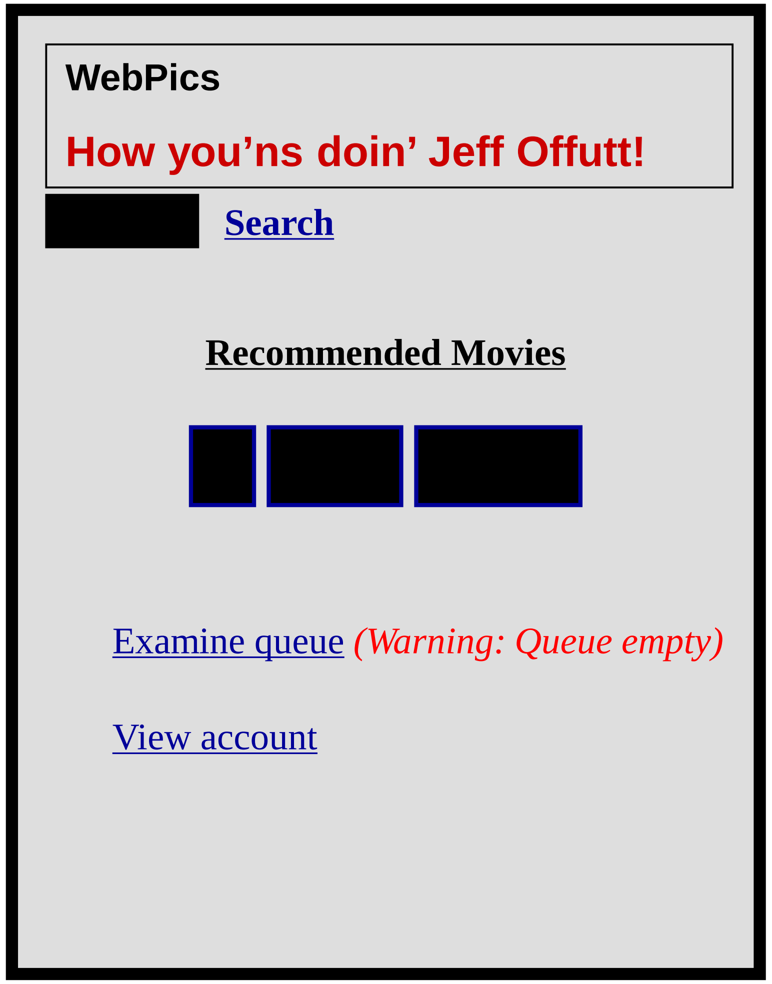
<u>**Web-based software**</u>

Connected with HTTP and XML

Loose, *extremely* loose, <u>distributed</u> coupling

How can we ensure the reliability of this type of software?

Problem 2: Dynamic Flow of Control


WebPics

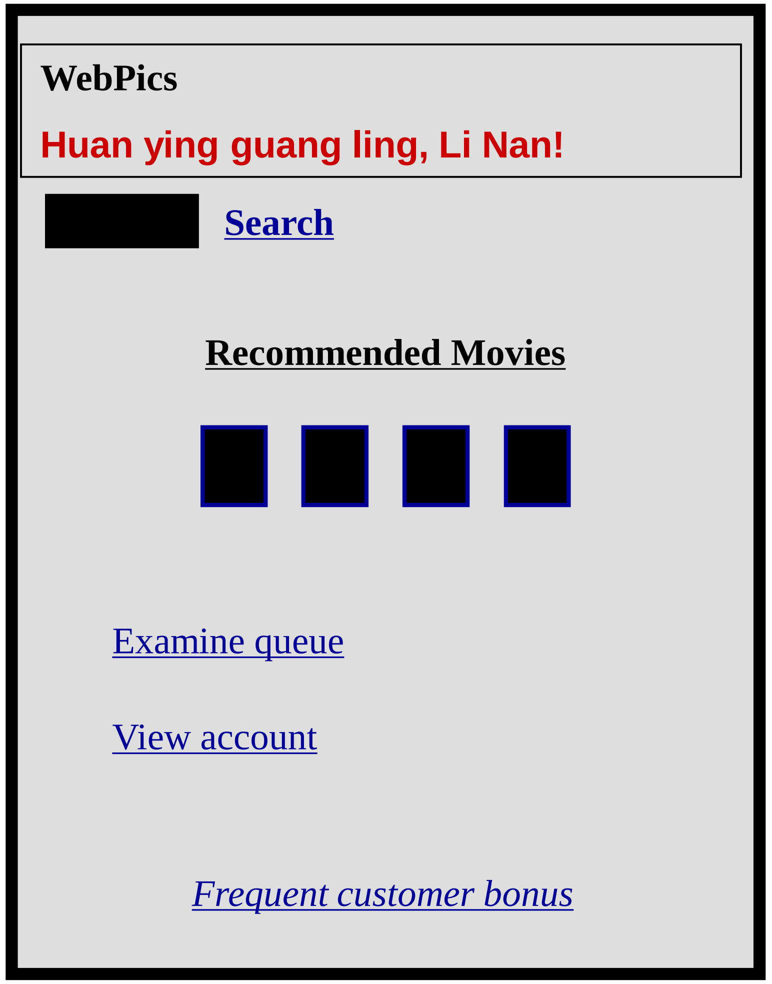
How you'ns doin' Jeff Offutt!

Search

Recommended Movies

Examine queue *(Warning: Queue empty)*

View account


WebPics

Huan ying guang ling, Li Nan!

Search

Recommended Movies

Examine queue

View account

Frequent customer bonus

How can we ensure the reliability of this type of system?

Dynamic Execution of Web Apps

- Parts of the program are generated dynamically
- Dynamic web pages are created when users make requests
- Different users will see different programs !
- The potential control, ala the traditional control flow graph, cannot be known ahead of time

The potential flow of control cannot be known statically

Problem 3: User Control Flow

- Users can make unexpected changes to the flow of control
  - *Operational transitions* are NOT based on an HTML link or program statement
  - Back button, forward button, refreshing, caching, URL rewriting
- Web software platforms offer new control flow mechanisms
  - Forward, redirect, asynchronous message passing (Ajax)
- State is stored in the server and in the HTML in the client's browser
- These transitions can cause unanticipated changes to the state of the web application

How can we ensure the reliability of this type of software?

Problem 4: Dynamic Integration

- Software modules can dynamically integrate with others if they use the same data structures

- EJBs can be inserted into web applications, which can immediately start using them

- Web services find and bind to other web services dynamically