### CSci 5271 Introduction to Computer Security Web security, part 1

Stephen McCamant University of Minnesota, Computer Science & Engineering

### Outline

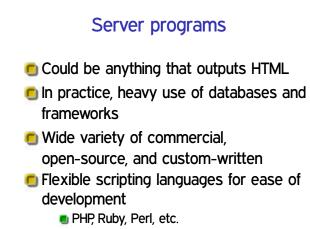
### The web from a security perspective

- Announcements intermission
- SQL injection
- Web authentication failures
- Cross-site scripting

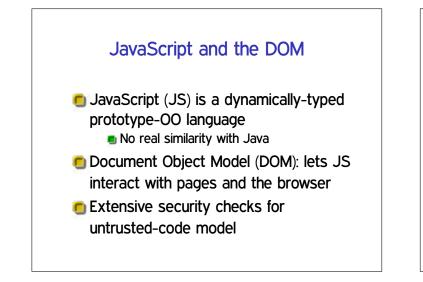
## Once upon a time: the static web HTTP: stateless file download protocol TCP, usually using port 80 HTML: markup language for text with formatting and links All pages public, so no need for authentication or encryption

### Web applications

- The modern web depends heavily on active software
- Static pages have ads, paywalls, or "Edit" buttons
- Many web sites are primarily forms or storefronts
- Web hosted versions of desktop apps like word processing

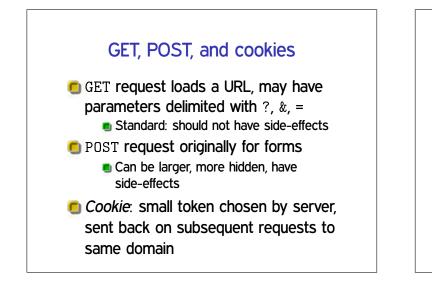


- Client-side programming
- Java: nice language, mostly moved to other uses
- ActiveX: Windows-only binaries, no sandboxing
  - Glad to see it on the way out
- Flash and Silverlight: most important use is DRM-ed video
- Core language: JavaScript



### Same-origin policy

- Origin is a tuple (scheme, host, port)
   E.g., (http, www.umn.edu, 80)
- Basic JS rule: interaction is allowed only with the same origin
- Different sites are (mostly) isolated applications



### User and attack models

- Web attacker" owns their own site (www.attacker.com)
  And users sometimes visit it
  - Realistic reasons: ads, SEO
- Network attacker" can view and sniff unencrypted data
  - Unprotected coffee shop WiFi

### Outline

The web from a security perspective

Announcements intermission

SQL injection

Web authentication failures

Cross-site scripting

### Note to early readers

- This is the section of the slides most likely to change in the final version
- If class has already happened, make sure you have the latest slides for announcements

### Outline

The web from a security perspective

Announcements intermission

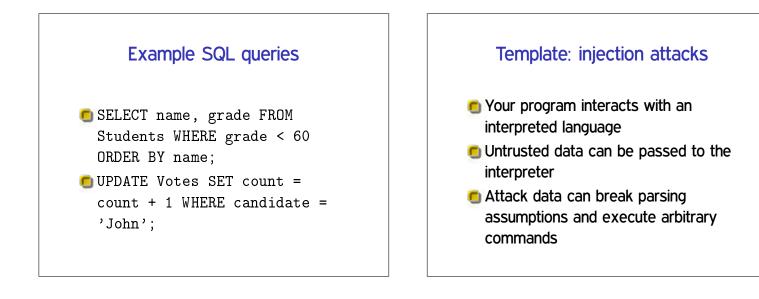
### SQL injection

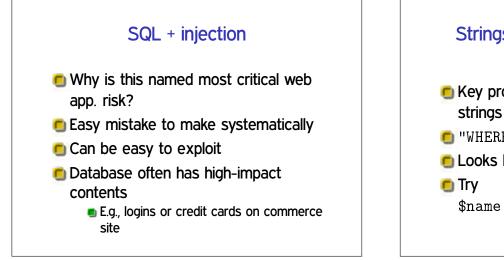
Web authentication failures

Cross-site scripting

### Relational model and SQL

- Relational databases have tables with rows and single-typed columns
- Used in web sites (and elsewhere) to provide scalable persistent storage
- Allow complex *queries* in a declarative language SQL





### Strings do not respect syntax • Key problem: assembling commands as

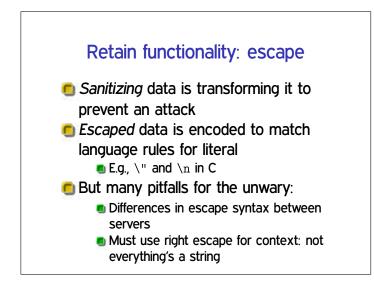
- strings
- 🗐 "WHERE name = '\$name';"
- **books like** \$name is a string

\$name = "me' OR grade > 80; --"

### Using tautologies Tautology: formula that's always true Often convenient for attacker to see a whole table Classic: OR 1=1

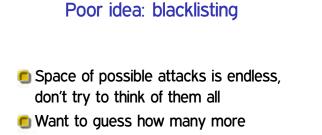
### Non-string interfaces

- Best fix: avoid constructing queries as strings
- SQL mechanism: prepared statement
   Original motivation was performance
- Web languages/frameworks often provide other syntax



### Lazy sanitization: whitelisting

- Allow only things you know to be safe/intended
- 🖲 Error or delete anything else
- Short whitelist is easy and relatively easy to secure
- E.g., digits only for non-negative integer
- But, tends to break benign functionality



- comment formats SQL has?
- Particularly silly: blacklisting 1=1

### Attacking without the program

Often web attacks don't get to see the program

Not even binary, it's on the server

- Surmountable obstacle:
  - Guess natural names for columns
  - Harvest information from error messages

### Blind SQL injection

- Attacking with almost no feedback
- Common: only "error" or "no error"
- One bit channel you can make yourself: if (x) delay 10 seconds
- Trick to remember: go one character at a time

### Injection beyond SQL

- SPath/XQuery: queries on XML data
- LDAP: queries used for authentication
- Shell commands: example from Ex. 1
- More web examples to come

### Outline

The web from a security perspective

Announcements intermission

SQL injection

Web authentication failures

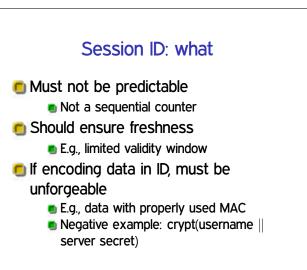
**Cross-site scripting** 

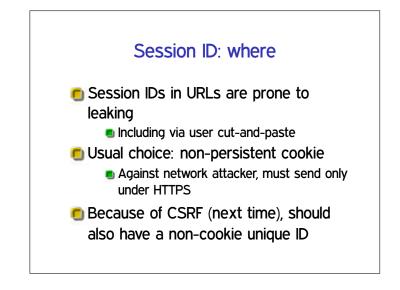
### Per-website authentication

- Many web sites implement their own login systems
  - If users pick unique passwords, little systemic risk
  - Inconvenient, many will reuse passwords
  - Lots of functionality each site must implement correctly
  - Without enough framework support, many possible pitfalls

### Building a session

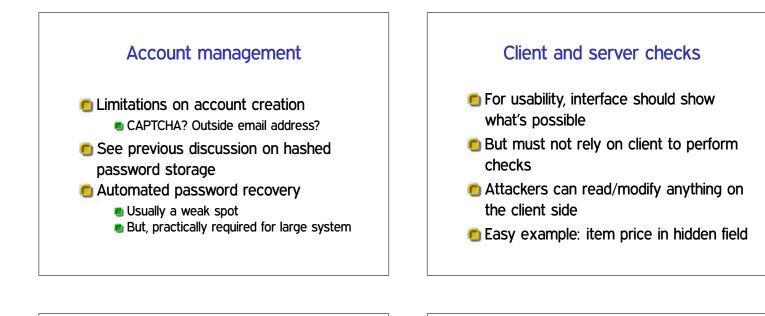
- HTTP was originally stateless, but many sites want stateful login sessions
- Building by tying requests together with a shared session ID
- Must protect confidentiality and integrity

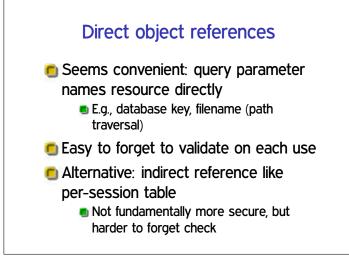




### Session management

- Create new session ID on each login
- 🖲 Invalidate session on logout
- 🖲 Invalidate after timeout
  - Usability / security tradeoff
  - Needed to protect users who fail to log out from public browsers





### Function-level access control

- E.g. pages accessed by URLs or interface buttons
- Must check each time that user is authorized
  - Attack: find URL when authorized, reuse when logged off
- Helped by consistent structure in code

### Outline

The web from a security perspective

Announcements intermission

SQL injection

Web authentication failures

Cross-site scripting

### XSS: HTML/JS injection

- Note: CSS is "Cascading Style Sheets"
- Another use of injection template
- Attacker supplies HTML containing JavaScript (or occasionally CSS)
- OWASP's most prevalent weakness
  - A category unto itself
  - Easy to commit in any dynamic page construction

# Why XSS is bad (and named that) attacker.com can send you evil JS directly But XSS allows access to bank.com data Violates same-origin policy Not all attacks actually involve multiple sites

### **Reflected XSS**

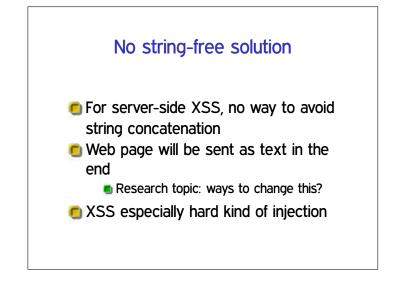
- Injected data used immediately in producing a page
- Commonly supplied as query/form parameters
- Classic attack is link from evil site to victim site

### Persistent XSS

- Injected data used to produce page later
- For instance, might be stored in database
- Can be used by one site user to attack another user
  - E.g., to gain administrator privilege

### DOM-based XSS

- Injected occurs in client-side page construction
- Flaw at least partially in code running on client
- Many attacks involve mashups and inter-site communication

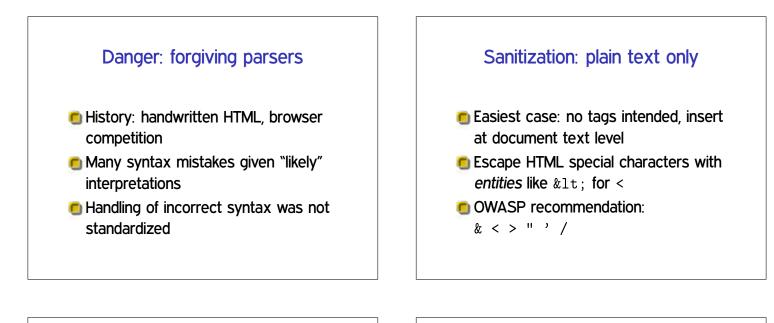


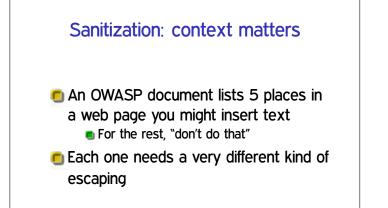
### Danger: complex language embedding

- JS and CSS are complex languages in their own right
- Can appear in various places with HTML

But totally different parsing rules

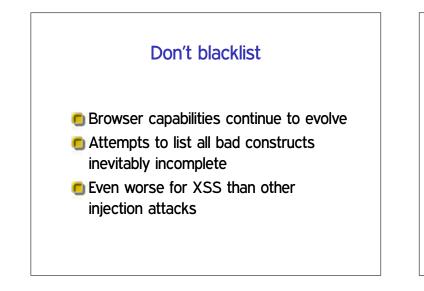
- Example: "...." used for HTML attributes and JS strings
  - What happens when attribute contains JS?





### Sanitization: tag whitelisting

- In some applications, want to allow benign markup like <b>
- But, even benign tags can have JS attributes
- Handling well essentially requires an HTML parser
  - But with an adversarial-oriented design



### Filter failure: one-pass delete

- Simple idea: remove all occurrences of <script>
- What happens to <scr<script>ipt>?



You may have heard of UTF-8

 Encode Unicode as 8-bit bytes

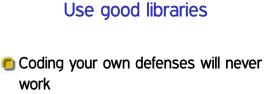
 UTF-7 is similar but uses only ASCII
 Encoding can be specified in a <meta> tag, or some browsers will guess

**5**+ADw-script+AD4-

### Filter failure: event handlers

<IMG onmouseover="alert('xss')">

- Put this on something the user will be tempted to click on
- There are more than 100 handlers like this recognized by various browsers



- Take advantage of known good implementations
- Best case: already built into your framework
  - Disappointingly rare

