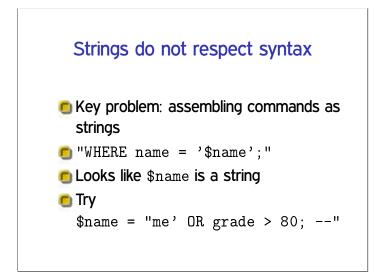
CSci 5271 Introduction to Computer Security Day 20: Web security, part 2

Stephen McCamant University of Minnesota, Computer Science & Engineering

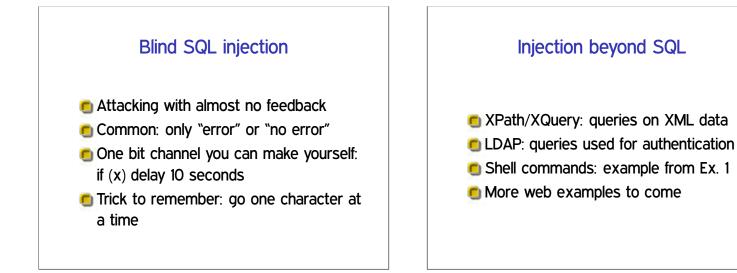
Outline

SQL injection (A1) Web authentication failures Announcements intermission Cross-site scripting (A3) More cross-site risks Confidentiality and privacy Even more risks



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



Outline

SQL injection (A1)

Web authentication failures

Announcements intermission

- Cross-site scripting (A3)
- More cross-site risks
- Confidentiality and privacy
- Even more risks

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

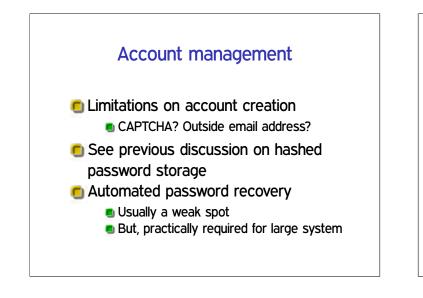


Session ID: where

- Session IDs in URLs are prone to leaking
 - Including via user cut-and-paste
- Usual choice: non-persistent cookie
 - Against network attacker, must send only under HTTPS
- Because of CSRF (coming up), should also have a non-cookie unique ID

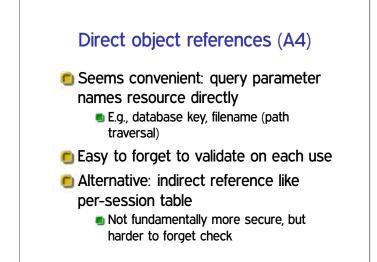
Session management (A2)

- 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



Client and server checks

- For usability, interface should show what's possible
- But must not rely on client to perform checks
- Attackers can read/modify anything on the client side
- Easy example: item price in hidden field



Function-level access control (A7)

- 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

SQL injection (A1)

Web authentication failures

Announcements intermission

Cross-site scripting (A3)

More cross-site risks

Confidentiality and privacy

Even more risks

Hands-on Assignment 2

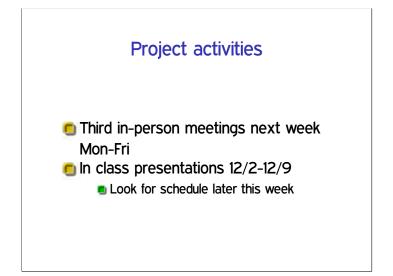
- Full version (6 questions) now posted
- 🖲 Email Yang if you didn't yet get a VM
- Server VMs update (for Q5,6) rolling out soon
- 🖲 Due a week from Friday

HA 2 questions

- 1. Network sniffing
- 2. Offline dictionary attack
- 3. Forging predictable cookies
- 4. SQL injection
- 5. Cross-site scripting
- 6. Crypto. attack against a poor MAC

Exercise sets

- Exercise set 3 should be graded by Thursday
- Exercise set 4 out, due week from Thursday



Outline

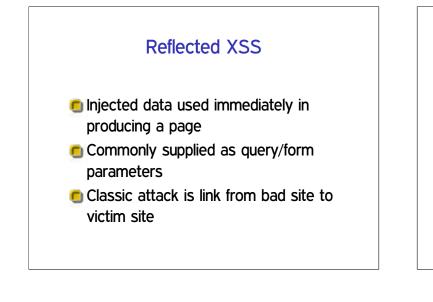
SQL injection (A1) Web authentication failures Announcements intermission Cross-site scripting (A3) More cross-site risks Confidentiality and privacy Even more risks

XSS: HTML/JS injection (A3)

- 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

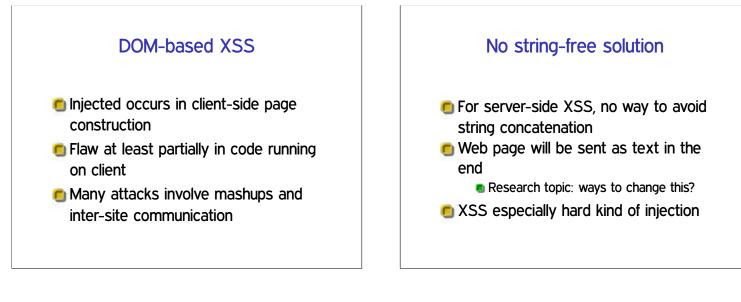
Why XSS is bad (and named that)

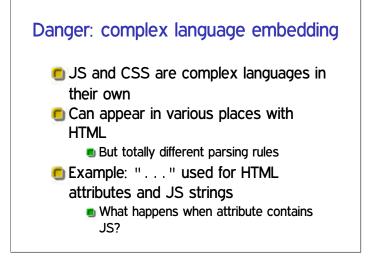
- attacker.com can send you evil JS directly
- But XSS allows access to bank.com data
- 🖲 Violates same-origin policy
- Not all attacks actually involve multiple sites



Persistent XSS

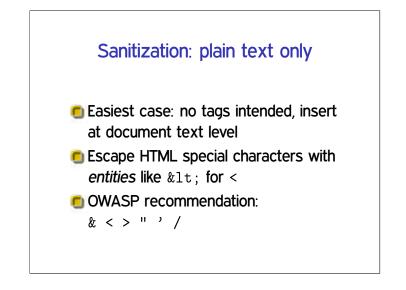
- Injected data used to produce page later
- For instance, might be stored in database
- Can be used by one site user to attack another user
 - E.g., to gain administrator privilege





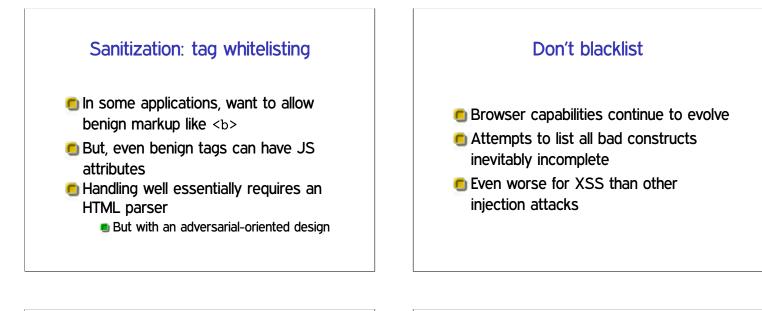
Danger: forgiving parsers

- History: handwritten HTML, browser competition
- Many syntax mistakes given "likely" interpretations
- Handling of incorrect syntax was not standardized

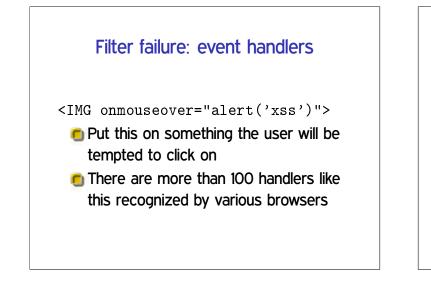


Sanitization: context matters

- An OWASP document lists 5 places in a web page you might insert text
 For the rest, "don't do that"
- Each one needs a very different kind of escaping







Use good libraries

- Coding your own defenses will never work
- Take advantage of known good implementations
- Best case: already built into your framework
 - Disappointingly rare

Content Security Policy

- New HTTP header, W3C candidate recommendation
- Lets site opt-in to stricter treatment of embedded content, such as:
 - No inline JS, only loaded from separate URLs
 - Disable JS eval et al.
- Has an interesting violation-reporting mode

Outline

SQL injection (A1)

- Web authentication failures
- Announcements intermission
- Cross-site scripting (A3)
- More cross-site risks
- Confidentiality and privacy
- Even more risks

HTTP header injection

- Untrusted data included in response headers
- Can include CRLF and new headers, or premature end to headers
- 🖲 AKA "response splitting"

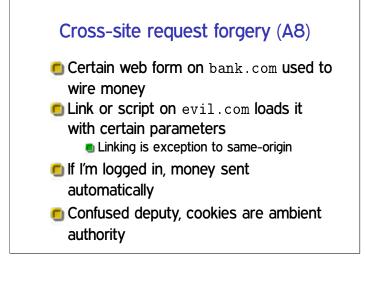
Content sniffing

Browsers determine file type from headers, extension, and content-based guessing

Latter two for ~ 1% server errors

- Many sites host "untrusted" images and media
- Inconsistencies in guessing lead to kind of XSS

E.g., "chimera" PNG-HTML document



CSRF prevention

Give site's forms random-nonce tokens

 E.g., in POST hidden fields
 Not in a cookie, that's the whole point

 Reject requests without proper token

 Or, ask user to re-authenticate

 XSS can be used to steal CSRF tokens

Open redirects (A10)

- Common for one page to redirect clients to another
- Target should be validated
 - With authentication check if appropriate
- Open redirect: target supplied in parameter with no checks
 - Doesn't directly hurt the hosting site
 - But reputation risk, say if used in phishing
 - We teach users to trust by site

Outline

SQL injection (A1)

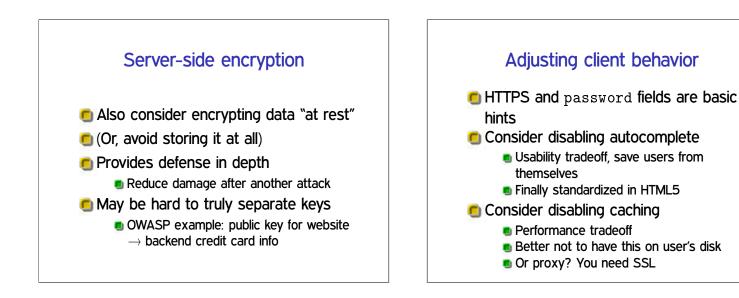
- Web authentication failures
- Announcements intermission
- Cross-site scripting (A3)
- More cross-site risks
- Confidentiality and privacy
- Even more risks

Site perspective (A6)

- Protect confidentiality of authenticators
 - Passwords, session cookies, CSRF tokens
- Duty to protect some customer info
 - Personally identifying info ("identity theft")
 - Credit-card info (Payment Card Industry Data Security Standards)
 - Health care (HIPAA), education (FERPA)
 - Whatever customers reasonably expect

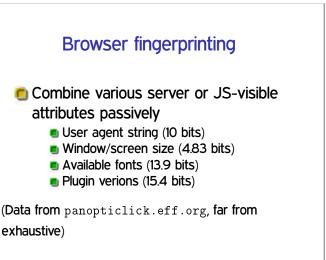
You need to use SSL

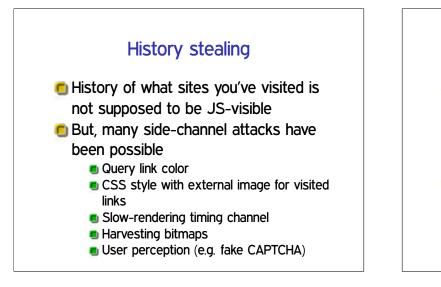
- Finally coming around to view that more sites need to support HTTPS
 Special thanks to WiFi, NSA
- If you take credit cards (of course)
- 🗐 If you ask users to log in
 - Must be protecting something, right?
 Also important for users of Tor et al.











Browser and extension choices

- More aggressive privacy behavior lives in extensions
 - Disabling most JavaScript (NoScript)
 - HTTPS Everywhere (whitelist)
 - Tor Browser Bundle
- Default behavior is much more controversial
 - Concern not to kill advertising support as an economic model

Outline

SQL injection (A1)

Web authentication failures

Announcements intermission

Cross-site scripting (A3)

More cross-site risks

Confidentiality and privacy

Even more risks

Misconfiguration problems (A5)

- 🖲 Default accounts
- Unneeded features
- Framework behaviors
 - Don't automatically create variables from query fields

Openness tradeoffs

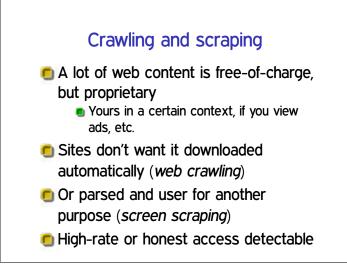
Error reporting

- Few benign users want to see a stack backtrace
- Directory listings
 - Hallmark of the old days
- Readable source code of scripts
 - Doesn't have your DB password in it, does it?

Using vulnerable components (A9) Large web apps can use a lot of third-part code

- Convenient for attackers too
 - OWASP: two popular vulnerable components downloaded 22m times
- Hiding doesn't work if it's popular
- Stay up to date on security announcements





Next time
Firewalls, NATs, and network intrusion detection