CSci 5271 Introduction to Computer Security Web security, combined slides

Stephen McCamant University of Minnesota, Computer Science & Engineering

Outline

The web from a security perspective (cont'd) SQL injection Announcements intermission Web authentication failures Cross-site scripting More risks Confidentiality and privacy

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

GET, POST, and cookies

- GET request loads a URL, may have parameters delimited with ?, &, =
 - Standard: should not have side-effects
- POST request originally for forms Can be larger, more hidden, have side-effects
- Cookie: small token chosen by server, sent back on subsequent requests to same domain

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

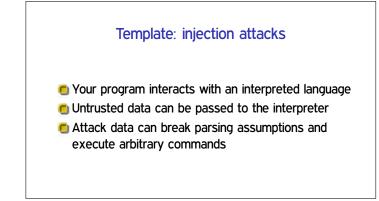
The web from a security perspective (cont'd) SQL injection Announcements intermission Web authentication failures Cross-site scripting More risks Confidentiality and privacy

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

Example SQL queries

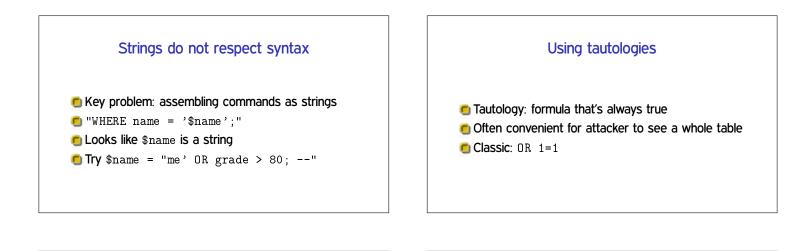
- SELECT name, grade FROM Students WHERE grade < 60 ORDER BY name;</p>
- UPDATE Votes SET count = count + 1 WHERE candidate = 'John';



SQL + injection

Why is this named most critical web app. risk?
Easy mistake to make systematically
Can be easy to exploit
Database often has high-impact contents

E.g., logins or credit cards on commerce site



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

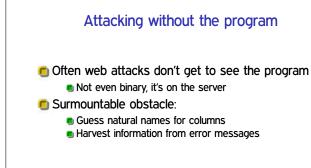
Retain functionality: escape

- Sanitizing data is transforming it to prevent an attack
 Escaped data is encoded to match language rules
 - for literal E.g., \" and \n in C
- But many pitfalls for the unwary:
 - Differences in escape syntax between servers
 - Must use right escape for context: not everything's a string



Poor idea: blacklisting

- Space of possible attacks is endless, don't try to think of them all
- Want to guess how many more comment formats SQL has?
- Particularly silly: blacklisting 1=1



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

XPath/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 (cont'd) SQL injection Announcements intermission Web authentication failures Cross-site scripting More risks Confidentiality and privacy

Hands-on assignment 2 questions

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

Outline

The web from a security perspective (cont'd) SQL injection Announcements intermission Web authentication failures Cross-site scripting More risks Confidentiality and privacy

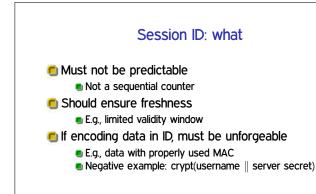
Per-website authentication

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
- Built by tying requests together with a shared session ID
- Must protect confidentiality and integrity



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 (next time), should also have a non-cookie unique ID

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



- Limitations on account creation CAPTCHA? Outside email address?
- See previous discussion on hashed password storage
- Automated password recovery
 - Usually a weak spot
 - But, practically required for large system

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

Direct object references

- Seems convenient: query parameter names resource directly
 - E.g., database key, filename (path traversal)
- Easy to forget to validate on each use
- Alternative: indirect reference like per-session table
 Not fundamentally more secure, but harder to forget check

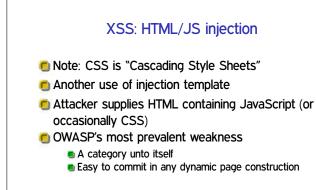
Function-level access control

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 (cont'd) SQL injection Announcements intermission Web authentication failures **Cross-site scripting** More risks Confidentiality and privacy



Why XSS is bad (and named that)

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

Reflected XSS

Injected data used immediately in producing a page
Ormmonly 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

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

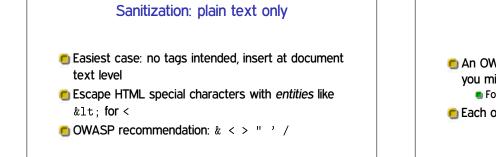
No string-free solution

- For server-side XSS, no way to avoid string concatenation
- Web page will be sent as text in the end Research topic: ways to change this?
- XSS especially hard kind of injection

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?

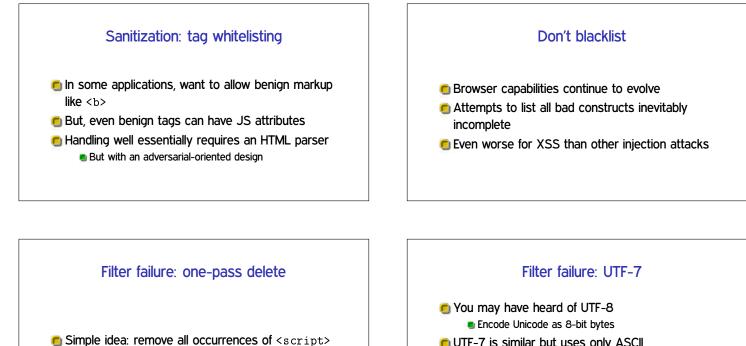
Danger: forgiving parsers

- 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



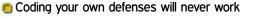
- UTF-7 is similar but uses only ASCII
- Encoding can be specified in a <meta> tag, or some browsers will guess
- 🖲 +ADw-script+AD4-

Filter failure: event handlers

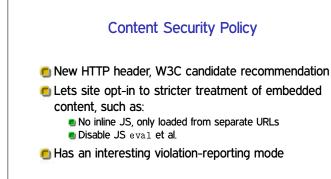
E) What happens to <scr<script>ipt>?

- 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



Outline

The web from a security perspective (cont'd) SQL injection Announcements intermission Web authentication failures Cross-site scripting More risks Confidentiality and privacy



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 a kind of XSS E.g., "chimera" PNG-HTML document

Cross-site request forgery

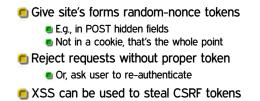
Ortain web form on bank.com used to wire money

Link or script on evil.com loads it with certain parameters

Linking is exception to same-origin

- If I'm logged in, money sent automatically
- Confused deputy, cookies are ambient authority

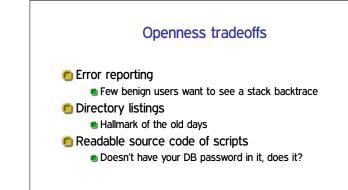
CSRF prevention



Open redirects 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

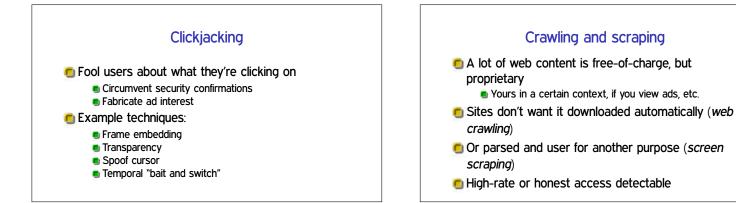
Misconfiguration problems

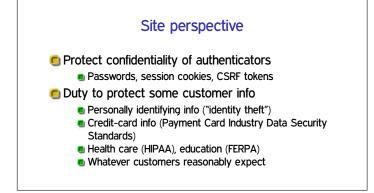
- 🖲 Default accounts
- Unneeded features
- Framework behaviors
 - Don't automatically create variables from query fields

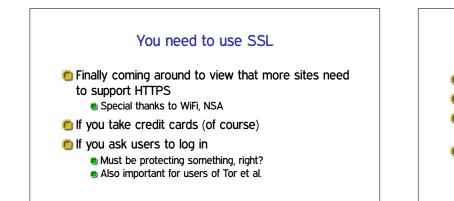


Using vulnerable components

- Large web apps can use a lot of third-party 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







Outline

The web from a security perspective (cont'd)

SQL injection

More risks

Announcements intermission

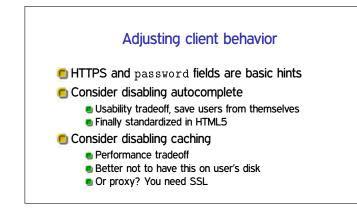
Web authentication failures

Confidentiality and privacy

Cross-site scripting

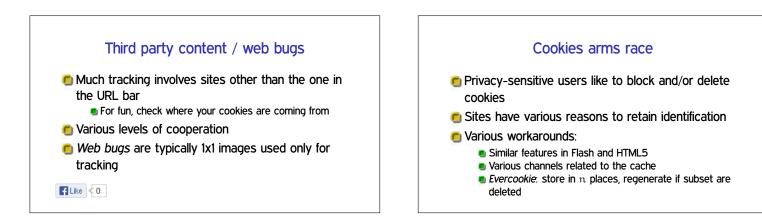


- Also consider encrypting data "at rest"
- 🖲 (Or, avoid storing it at all)
- 🖲 Provides defense in depth
 - Reduce damage after another attack
- May be hard to truly separate keys
 - \blacksquare OWASP example: public key for website \rightarrow backend credit card info



User vs. site perspective

- User privacy goals can be opposed to site goals
- Such as in tracking for advertisements
- Browser makers can find themselves in the middle Of course, differ in institutional pressures



Browser fingerprinting

- Combine various server or JS-visible attributes passively
 - User agent string (10 bits)
 - Window/screen size (4.83 bits)
 - Available fonts (13.9 bits)
 - Plugin verions (15.4 bits)

(Data from panopticlick.eff.org, far from exhaustive)

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

History stealing

- History of what sites you've visited is not supposed to be JS-visible
- But, many side-channel attacks have been possible
 - Query link color
 - CSS style with external image for visited links
 - Slow-rendering timing channel
 - Harvesting bitmaps
 - User perception (e.g. fake CAPTCHA)