CSci 4271W
Development of Secure Software Systems
Day 18: Web Security 2

Stephen McCamant

University of Minnesota, Computer Science & Engineering

Outline

SQL injection, cont'd

Cross-site scripting

Announcements intermission

More cross-site risks

Confidentiality and privacy

Even more web 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

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

Outline

SQL injection, cont'd

Cross-site scripting

Announcements intermission

More cross-site risks

Confidentiality and privacy

Even more web risks

XSS: HTML/JS injection

- Note: CSS is "Cascading Style Sheets"
- Another instance 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

- 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
- 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: plain text only

- Easiest case: no tags intended, insert at document text level
- Escape HTML special characters with *entities* like `<` for <
- OWASP recommendation: & < > " ' /

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

Sanitization: tag allow-listing

- 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

Don't deny-list

- Browser capabilities continue to evolve
- Attempts to list all bad constructs inevitably incomplete
- Even worse for XSS than other injection attacks

Filter failure: one-pass delete

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

Filter failure: UTF-7

- 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
- `+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

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

- Added HTTP header, W3C 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

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

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

Cross-site request forgery

- Certain 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

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

- 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

Site perspective

- 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.

Server-side encryption

- 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
  - OWASP example: public key for website → backend credit card info

Adjusting client behavior

- HTTPS and `password` fields are basic hints
- Consider disabling autocomplete
  - Usability tradeoff, save users from themselves
  - Finally standardized in HTML5
- Consider disabling caching
  - Performance tradeoff
  - Better not to have this on user's disk
  - Or proxy? You need SSL

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

Third party content / web bugs

- Much tracking involves sites other than the one in the URL bar
  - For fun, check where your cookies are coming from
- Various levels of cooperation
- *Web bugs* are typically 1x1 images used only for tracking

[f Like] ⟨0⟩

Cookies arms race

- Privacy-sensitive users like to block and/or delete cookies
- Sites have various reasons to retain identification
- Various workarounds:
  - Similar features in Flash and HTML5
  - Various channels related to the cache
  - *Evercookie*: store in n places, regenerate if subset are deleted

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)

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)

Browser and extension choices

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

Outline

Misconfiguration problems

- 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

- 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

Clickjacking

- Fool users about what they're clicking on
  - Circumvent security confirmations
  - Fabricate ad interest
- Example techniques:
  - Frame embedding
  - Transparency
  - Spoof cursor
  - Temporal "bait and switch"

Crawling and scraping

- A lot of web content is free-of-charge, but proprietary
  - Yours in a certain context, if you view ads, etc.
- Sites don't want it downloaded automatically (*web crawling*)
- Or parsed and user for another purpose (*screen scraping*)
- High-rate or honest access detectable