CSci 4271W Development of Secure Software Systems Day 16: Web Application Security, part 2

Stephen McCamant University of Minnesota, Computer Science & Engineering

Outline

Cross-site scripting, cont'd

- More cross-site risks
- SQL injection
- Confidentiality and privacy
- Even more web risks

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

Sanitization: plain text only

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

Sanitization: context matters

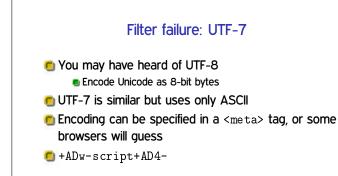
An OWASP document lists 5 places in a web page you might insert text For the rest, "don't do that"

Each one needs a very different kind of escaping

Sanitization: tag allow-listing

- In some applications, want to allow benign markup like
- 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 Filter failure: one-pass delete Image: Browser capabilities continue to evolve Image: Simple idea: remove all occurrences of <script>incomplete Image: Attempts to list all bad constructs inevitably incomplete Image: Simple idea: remove all occurrences of <script>incomplete Image: Even worse for XSS than other injection attacks Image: What happens to <script>ipt>?



Outline

Cross-site scripting, cont'd

Confidentiality and privacy

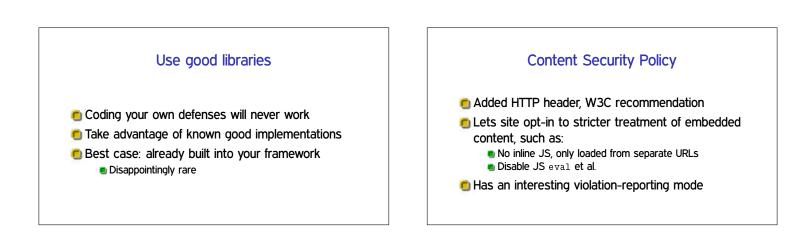
More cross-site risks

Even more web risks

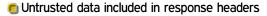
SQL injection

Filter failure: event handlers

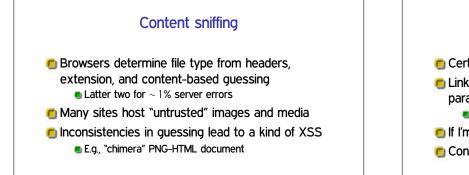
- Put this on something the user will be tempted to click on
- There are more than 100 handlers like this recognized by various browsers



HTTP header injection



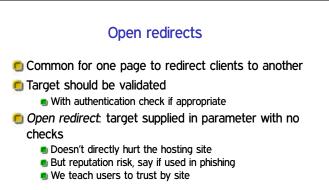
- Can include CRLF and new headers, or premature end to headers
- AKA "response splitting"





- 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
- Confused deputy, cookies are ambient authority





Outline

Cross-site scripting, cont'd

More cross-site risks

SQL injection

Confidentiality and privacy

Even more web risks

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

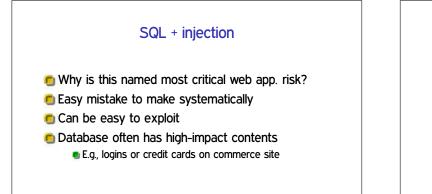
Example SQL queries

SELECT name, grade FROM Students WHERE grade < 60 ORDER BY name;</p>

UPDATE Votes SET count = count + 1 WHERE candidate = 'John';

Template: injection attacks

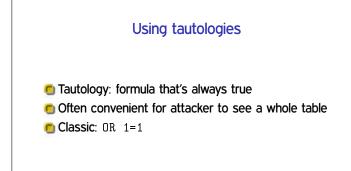
- Your program interacts with an interpreted language
- Untrusted data can be passed to the interpreter
- Attack data can break parsing assumptions and execute arbitrary commands



Strings do not respect syntax

- Key problem: assembling commands as strings
- 🖲 "WHERE name = '\$name';"
- 🖲 Looks like \$name is a string

```
🖲 Try $name = "me' OR grade > 80; --"
```



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



Escaped data is encoded to match language rules for literal

• E.g., $\ \$ and $\ \$ 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

Lazy sanitization: allow-listing

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

Poor idea: deny-listing

- 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: deny 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

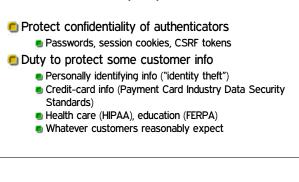
Blind SQL injection

- 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

- Shell commands, format strings, XSS
- XPath/XQuery: queries on XML data
- LDAP: gueries used for authentication





You need to use SSL Server-side encryption Finally coming around to view that more sites need Also consider encrypting data "at rest" to support HTTPS (Or, avoid storing it at all) Special thanks to WiFi, NSA Provides defense in depth If you take credit cards (of course) Reduce damage after another attack 🖲 If you ask users to log in May be hard to truly separate keys Must be protecting something, right? **•** OWASP example: public key for website \rightarrow backend Also important for users of Tor et al. 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



Adjusting client behavior

HTTPS and password fields are basic hints

Usability tradeoff, save users from themselves

Consider disabling autocomplete

Consider disabling caching

Performance tradeoff

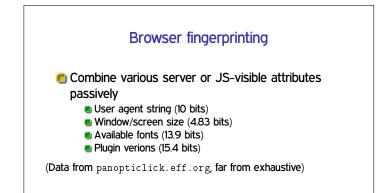
Or proxy? You need SSL

Finally standardized in HTML5

Better not to have this on user's disk

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



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



Cross-site scripting, cont'd

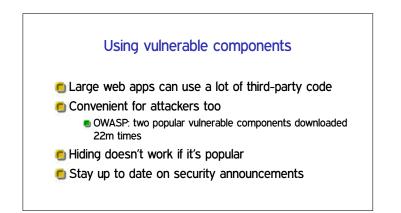
More cross-site risks

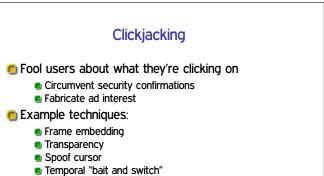
- SQL injection
- Confidentiality and privacy

Even more web risks

Misconfiguration problems Openness tradeoffs Error reporting Default accounts Unneeded features Directory listings Hallmark of the old days Framework behaviors Readable source code of scripts Don't automatically create variables from query fields

- Few benign users want to see a stack backtrace
 - Doesn't have your DB password in it, does it?





Crawling and scraping

- A lot of web content is free-of-charge, but proprietary
 - Source of the second se
- Sites don't want it downloaded automatically (web crawling)
- Or parsed and user for another purpose (screen scraping)
- High-rate or honest access detectable