CSci 4271W
Development of Secure Software Systems
Day 15: Web Application Security, part 3

Stephen McCamant
University of Minnesota, Computer Science & Engineering

Outline

SQL injection
Injection attack demo
Confidentiality and privacy
Even more web risks
Crypto basics

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;`
- `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

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

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; --"`

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

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

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: queries used for authentication

Outline

SQL injection

Injection attack demo

Confidentiality and privacy

Even more web risks

Crypto basics

Injection attack template

- Injection attacks often have a three-part structure:
  1. Break out of enclosing structures
  2. Malicious commands
  3. Re-enter structures, or otherwise clean up

Injection attack demo

- To illustrate, let's see how the CSS XSS attack from the lab works in this way
- (Demo in text editor)

Outline

SQL injection

Injection attack demo

Confidentiality and privacy

Even more web risks

Crypto basics

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

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

SQL injection

Injection attack demo

Confidentiality and privacy

Even more web risks

Crypto basics

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

Outline

SQL injection

Injection attack demo

Confidentiality and privacy

Even more web risks

Crypto basics

-ography, -ology, -analysis

- Cryptography (narrow sense): designing encryption
- Cryptanalysis: breaking encryption
- Cryptology: both of the above
- Code (narrow sense): word-for-concept substitution
- Cipher: the "codes" we actually care about

Caesar cipher

- Advance three letters in alphabet: $A \to D, B \to E, \ldots$
- Decrypt by going back three letters
- Internet-era variant: rot-13
- Easy to break if you know the principle

Keys and Kerckhoffs's principle

- The only secret part of the cipher is a *key*
- Security does not depend on anything else being secret
- Modern (esp. civilian, academic) crypto embraces openness quite strongly

Symmetric vs. public key

- Symmetric key (up first): one key used by all participants
- Public key: one key kept secret, another published
  - Techniques invented in 1970s
  - Makes key distribution easier
  - Depends on fancier math

Goal: secure channel

- Leaks no content information
  - Not protected: size, timing
- Messages delivered intact and in order
  - Or not at all
- Even if an adversary can read, insert, and delete traffic

One-time pad

- Secret key is truly random data as long as message
- Encrypt by XOR (more generally addition mod alphabet size)
- Provides perfect, "information-theoretic" secrecy
- No way to get around key size requirement

Computational security

- More realistic: assume adversary has a limit on computing power
- Secure if breaking encryption is computationally infeasible
  - E.g., exponential-time brute-force search
- Ties cryptography to complexity theory

Key sizes and security levels

- Difficulty measured in powers of two, ignore small constant factors
- Power of attack measured by number of steps, aim for better than brute force
- 2^{32} definitely too easy, probably 2^{64} too
- Modern symmetric key size: at least 2^{128}

Crypto primitives

- Base complicated systems on a minimal number of simple operations
- Designed to be fast, secure in wide variety of uses
- Study those primitives very intensely

Attacks on encryption

- Known ciphertext
  - Weakest attack
- Known plaintext (and corresponding ciphertext)
- Chosen plaintext
- Chosen ciphertext (and plaintext)
  - Strongest version: adaptive

Certificational attacks

- Good primitive claims no attack more effective than brute force
- Any break is news, even if it's not yet practical
  - Canary in the coal mine
- E.g., $2^{126.1}$ attack against AES-128
- Also watched: attacks against simplified variants

Fundamental ignorance

- We don't really know that any computational cryptosystem is secure
- Security proof would be tantamount to proving $P \neq NP$
- Crypto is fundamentally more uncertain than other parts of security

Relative proofs

- Prove security under an unproved assumption
- In symmetric crypto, prove a construction is secure if the primitive is
  - Often the proof looks like: if the construction is insecure, so is the primitive
- Can also prove immunity against a particular kind of attack

Random oracle paradigm

- Assume ideal model of primitives: functions selected uniformly from a large space
  - Anderson: elves in boxes
- Not theoretically sound; assumption cannot be satisfied
- But seems to be safe in practice

Pseudorandomness and distinguishers

- Claim: primitive cannot be distinguished from a truly random counterpart
  - In polynomial time with non-negligible probability
- We can build a distinguisher algorithm to exploit any weakness
- Slightly too strong for most practical primitives, but a good goal

Open standards

- How can we get good primitives?
- Open-world best practice: run competition, invite experts to propose then attack
- Run by neutral experts, e.g. US NIST
- Recent good examples: AES, SHA-3

A certain three-letter agency

- National Security Agency (NSA): has primary responsibility for “signals intelligence”
- Dual-mission tension:
  - Break the encryption of everyone in the world
  - Help US encryption not be broken by foreign powers