

CSci 5271
Introduction to Computer Security
Day 19: Web security, part 1

Stephen McCamant
University of Minnesota, Computer Science & Engineering

Outline

SSL/TLS

DNSSEC

Announcements intermission

The web from a security perspective

SQL injection

Web authentication failures

But wait, there's more!

- Two recent problems, with cute names:
- Heartbleed
 - Information-disclosure implementation bug in OpenSSL
 - Buffer over-read
- POODLE: "Padding Oracle On Downgraded Legacy Encryption"
 - Padding oracle in SSL 3.0 returns when a MITM forces downgrade

CA vs. leaf checking bug

- Certs have a bit that says if they're a CA
- All but last entry in chain should have it set
- Browser authors repeatedly fail to check this bit
- Allows any cert to sign any other cert

MD5 certificate collisions

- MD5 collisions allow forging CA certs
- Create innocuous cert and CA cert with same hash
 - Requires some guessing what CA will do, like sequential serial numbers
 - Also 200 PS3s
- Oh, should we stop using that hash function?

CA validation standards

- CA's job to check if the buyer really is `foo.com`
- Race to the bottom problem:
 - CA has minimal liability for bad certs
 - Many people want cheap certs
 - Cost of validation cuts out of profit
- "Extended validation" (green bar) certs attempt to fix

HTTPS and usability

- Many HTTPS security challenges tied with user decisions
- Is this really my bank?
- Seems to be a quite tricky problem
 - Security warnings often ignored, etc.
 - We'll return to this as a major example later

Outline

SSL/TLS

DNSSEC

Announcements intermission

The web from a security perspective

SQL injection

Web authentication failures

DNS: trusted but vulnerable

- Almost every higher-level service interacts with DNS
- UDP protocol with no authentication or crypto
 - Lots of attacks possible
- Problems known for a long time, but challenge to fix compatibly

DNSSEC goals and non-goals

- + Authenticity of positive replies
- + Authenticity of negative replies
- + Integrity
- Confidentiality
- Availability

First cut: signatures and certificates

- Each resource record gets an RRSIG signature
 - E.g., A record for one name→address mapping
 - Observe: signature often larger than data
- Signature validation keys in DNSKEY RRs
- Recursive chain up to the root (or other "anchor")

Add more indirection

- DNS needs to scale to very large flat domains like .com
- Facilitated by having single DS RR in parent indicating delegation
- Chain to root now includes DSes as well

Negative answers

- Also don't want attackers to spoof non-existence
 - Gratuitous denial of service, force fallback, etc.
- But don't want to sign "x does not exist" for all x
- Solution 1, NSEC: "there is no name between acacia and baobab"

Preventing zone enumeration

- Many domains would not like people enumerating all their entries
- DNS is public, but "not that public"
- Unfortunately NSEC makes this trivial
- Compromise: NSEC3 uses password-like salt and repeated hash, allows opt-out

DANE: linking TLS to DNSSEC

- "DNS-based Authentication of Named Entities"
- DNS contains hash of TLS cert, don't need CAs
- How is DNSSEC's tree of certs better than TLS's?

Signing the root

- Political problem: many already distrust US-centered nature of DNS infrastructure
- Practical problem: must be very secure with no single point of failure
- Finally accomplished in 2010
 - Solution involves 'key ceremonies', international committees, smart cards, safe deposit boxes, etc.

Deployment

- Standard deployment problem: all cost and no benefit to being first mover
- Servers working on it, mostly top-down
- Clients: still less than 10%
- Will be probably common: insecure connection to secure resolver

Outline

SSL/TLS

DNSSEC

Announcements intermission

The web from a security perspective

SQL injection

Web authentication failures

Upcoming assignments

- Exercise set 3 due tonight
- HA2 Q1-2 readable now
- HA2 materials coming probably Friday or over weekend
- Exercise set 4 posted soon, due 11/20

Outline

SSL/TLS

DNSSEC

Announcements intermission

The web from a security perspective

SQL injection

Web authentication failures

Once upon a time: the static web

- HTTP: stateless file download protocol
 - TCP, usually using port 80
- HTML: markup language for text with formatting and links
- All pages public, so no need for authentication or encryption

Web applications

- The modern web depends heavily on active software
- Static pages have ads, paywalls, or "Edit" buttons
- Many web sites are primarily forms or storefronts
- Web hosted versions of desktop apps like word processing

Server programs

- Could be anything that outputs HTML
- In practice, heavy use of databases and frameworks
- Wide variety of commercial, open-source, and custom-written
- Flexible scripting languages for ease of development
 - PHP, Perl, Ruby, etc.

Client-side programming

- Java: nice language, mostly moved to other uses
- ActiveX: Windows-only binaries, no sandboxing
 - Glad to see it on the way out
- Flash and Silverlight: most important use is DRM-ed video
- Core language: JavaScript

JavaScript and the DOM

- JavaScript (JS) is a dynamically-typed prototype-OO language
 - No real similarity with Java
- Document Object Model (DOM): lets JS interact with pages and the browser
- Extensive security checks for untrusted-code model

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

SSL/TLS

DNSSEC

Announcements intermission

The web from a security perspective

SQL injection

Web authentication failures

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: whitelisting

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

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

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

- XPath/XQuery: queries on XML data
- LDAP: queries used for authentication
- Shell commands: example from Ex. 1
- More web examples to come

Outline

SSL/TLS

DNSSEC

Announcements intermission

The web from a security perspective

SQL injection

Web authentication failures

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

Session ID: what

- Must not be predictable
 - Not a sequential counter
- Should ensure freshness
 - E.g., limited validity window
- If encoding data in ID, must be unforgeable
 - E.g., data with properly used MAC
 - Negative example: `crypt(username || server secret)`

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

Account management

- ▣ 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

Next time

- ▣ Cross-site scripting and related risks
- ▣ Confidentiality and privacy risks