CSci 5271 Introduction to Computer Security Day 20: Web security, part 1

Stephen McCamant University of Minnesota, Computer Science & Engineering

Outline

Key distribution and PKI, cont'd SSH TLS (SSL) The web from a security perspective SQL injection

Certificates

() A name and a public key, signed by someone else • $C_A = \text{Sign}_S(A, K_A)$

Basic unit of transitive trust

Commonly use a complex standard "X.509"

Certificate authorities

"CA" for short: entities who sign certificates

- 🖲 Simplest model: one central CA
- Works for a single organization, not the whole world

CA hierarchies

🖲 Organize CAs in a tree

Distributed, but centralized (like DNS)

- Check by follow a path to the root
- Best practice: sub CAs are limited in what they certify

PKI for authorization

- Enterprise PKI can link up with permissions
- One approach: PKI maps key to name, ACL maps name to permissions
- Often better: link key with permissions directly, name is a comment
 - More like capabilities

The revocation problem

How can we make certs "go away" when needed?

- Impossible without being online somehow
- 1. Short expiration times
- 2. Certificate revocation lists
- 3. Certificate status checking

Outline

Key distribution and PKI, cont'd

SSH

TLS (SSL)

The web from a security perspective

SQL injection

Short history of SSH

🖲 Started out as freeware by Tatu Ylönen in 1995

- Original version commercialized
- Fully open-source OpenSSH from OpenBSD
- Protocol redesigned and standardized for "SSH 2"



SSH host keys

Every SSH server has a public/private keypair
 Ideally, never changes once SSH is installed
 Early generation a classic entropy problem
 Especially embedded systems, VMs

Authentication methods

- Password, encrypted over channel
- 🍯 . shosts: like . rhosts, but using client host key
- 🖲 User-specific keypair
 - Public half on server, private on client
- Plugins for Kerberos, PAM modules, etc.

Old crypto vulnerabilities

I.x had only CRC for integrity

 Worst case: when used with RC4
 Injection attacks still possible with CBC

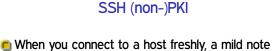
 CRC compensation attack
 For least-insecure 1.x-compatibility, attack detector
 Alas, detector had integer overflow worse than original attack

Newer crypto vulnerabilities

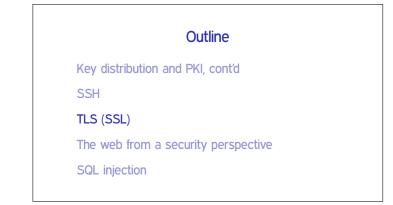
- IV chaining: IV based on last message ciphertext
 Allows chosen plaintext attacks
 - Better proposal: separate, random IVs
- Some tricky attacks still left
 - Send byte-by-byte, watch for errors
 - Of arguable exploitability due to abort
- Now migrating to CTR mode

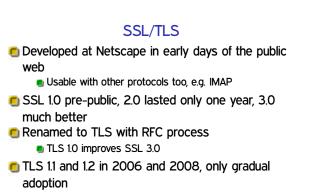
SSH over SSH

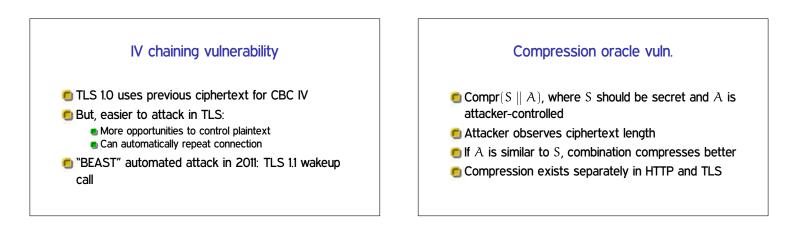
- Common in these days of NATs
- Better: have machine 1 forward an encrypted connection
- 1. No need to trust 1 for secrecy
- 2. Timing attacks against password typing



When the host key has changed, a large warning







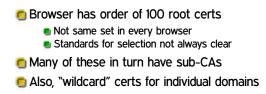


Too many vulnerabilities to mention them all in lecture

Kaloper-Meršinjak et al. have longer list "Lessons learned" are variable, though

Meta-message: don't try this at home

HTTPS hierarchical PKI

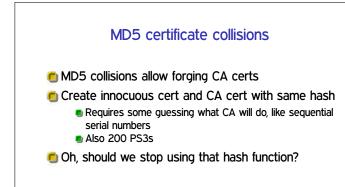


Hierarchical trust?

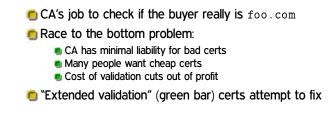
- No. Any CA can sign a cert for any domain
- A couple of CA compromises recently
- Most major governments, and many companies you've never heard of, could probably make a google.com cert
- Still working on: make browser more picky, compare notes



- 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







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

Key distribution and PKI, cont'd SSH TLS (SSL)

The web from a security perspective

SQL injection

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

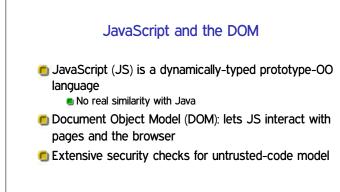
- The modern web depends heavily on active software
- Static pages have ads, paywalls, or "Edit" buttons
- Many web sites are primarily forms or storefronts
- Web hosted versions of desktop apps like word processing

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

Flexible scripting languages for ease of development PHP, Ruby, Perl, etc.



- 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

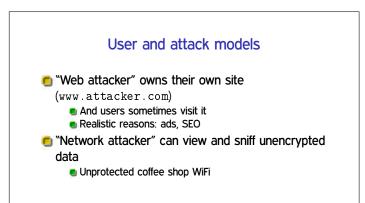


Same-origin policy

- 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



Outline

Key distribution and PKI, cont'd

SSH

TLS (SSL)

The web from a security perspective

SQL injection

Relational model and SQL

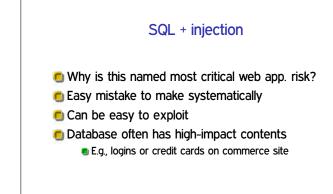
- Relational databases have tables with rows and single-typed columns
- Used in web sites (and elsewhere) to provide scalable persistent storage
- Allow complex queries in a declarative language SQL

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

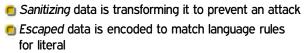


Strings do not respect syntax

Key problem: assembling commands as strings 🖲 "WHERE name = '\$name';" Looks like \$name is a string E Try \$name = "me' OR grade > 80; --"

Using tautologies Non-string interfaces Best fix: avoid constructing queries as strings Tautology: formula that's always true SQL mechanism: prepared statement Often convenient for attacker to see a whole table Original motivation was performance Classic: OR 1=1 Web languages/frameworks often provide other syntax Retain functionality: escape



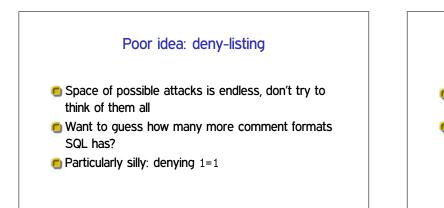


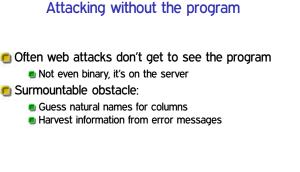
```
■ E.g., \ mathbb{n} and n in C
```

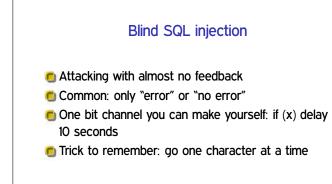
- But many pitfalls for the unwary:
 - Differences in escape syntax between servers
 - Must use right escape for context: not everything's a string

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







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