CSci 4271W Development of Secure Software Systems Day 17: Web Security 1

Stephen McCamant University of Minnesota, Computer Science & Engineering

Outline

More choices for isolation (cont'd) The web from a security perspective Announcements intermission SQL injection Cross-site scripting

Separate users

Reuse OS facilities for access control

- Unit of trust: program or application
- 🖲 Older example: qmail
- 🖲 Newer example: Android
- Limitation: lots of things available to any user

chroot

- Unix system call to change root directory
- Restrict/virtualize file system access
- Only available to root
- Does not isolate other namespaces

OS-enabled containers

One kernel, but virtualizes all namespaces
 FreeBSD jails, Linux LXC, Solaris zones, etc.
 Quite robust, but the full, fixed, kernel is in the TCB

(System) virtual machines

- Presents hardware-like interface to an untrusted kernel
- Strong isolation, full administrative complexity
- I/O interface looks like a network, etc.

Virtual machine designs

- (Type 1) hypervisor: 'superkernel' underneath VMs
 Hosted: regular OS underneath VMs
- Paravirtualization: modify kernels in VMs for ease of virtualization

Virtual machine technologies

- Hardware based: fastest, now common
- Partial translation: e.g., original VMware
- 🖲 Full emulation: e.g. QEMU proper
 - Slowest, but can be a different CPU architecture

Modern example: Chrom(ium)

- Separates "browser kernel" from less-trusted "rendering engine"
 - Pragmatic, keeps high-risk components together
- Experimented with various Windows and Linux sandboxing techniques
- Blocked 70% of historic vulnerabilities, not all new ones
- http://seclab.stanford.edu/websec/chromium/

Outline

More choices for isolation (cont'd)

- The web from a security perspective
- Announcements intermission
- SQL injection
- Cross-site scripting

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, Ruby, Perl, 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: last important use was 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

More choices for isolation (cont'd)

The web from a security perspective

Announcements intermission

SQL injection

Cross-site scripting

Project 1 now available

 Two submissions, November 10th and December 1st
 We're implementing extensions as "late" Canvas submissions

Next reading and related quiz

- Next reading is an OWASP web page about the top 10 web risks
- Another 5-question reading quiz is due a week from today.

Instructor working from home

- Prof. McCamant won't be in the lab tomorrow afternoon
- 🖲 Thursday's lecture will also be via Zoom.

Outline

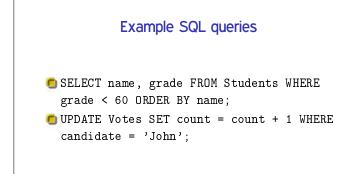
More choices for isolation (cont'd) The web from a security perspective Announcements intermission

SQL injection

Cross-site scripting

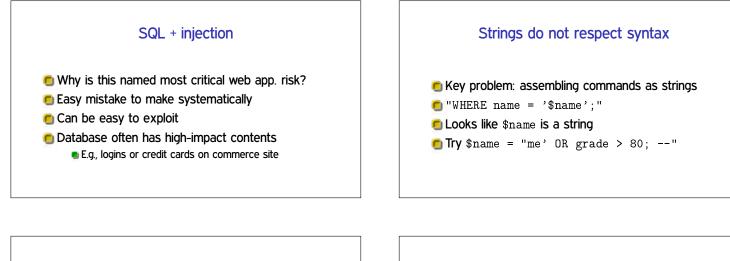
Relational model and SQL

- 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



Template: injection attacks

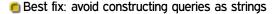
- Your program interacts with an interpreted language
- Untrusted data can be passed to the interpreter
- Attack data can break parsing assumptions and execute arbitrary commands



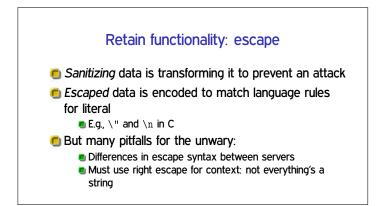
Tautology: formula that's always true
 Often convenient for attacker to see a whole table
 Classic: OR 1=1

Using tautologies

Non-string interfaces



- SQL mechanism: prepared statement
 Original motivation was performance
- Web languages/frameworks often provide other syntax



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

Earlier: shell commands, format strings
 XPath/XQuery: queries on XML data
 LDAP: queries used for authentication
 Next up: XSS

Outline

More choices for isolation (cont'd)

The web from a security perspective

Announcements intermission

SQL injection

Cross-site scripting

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 <code>bank.com</code> 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

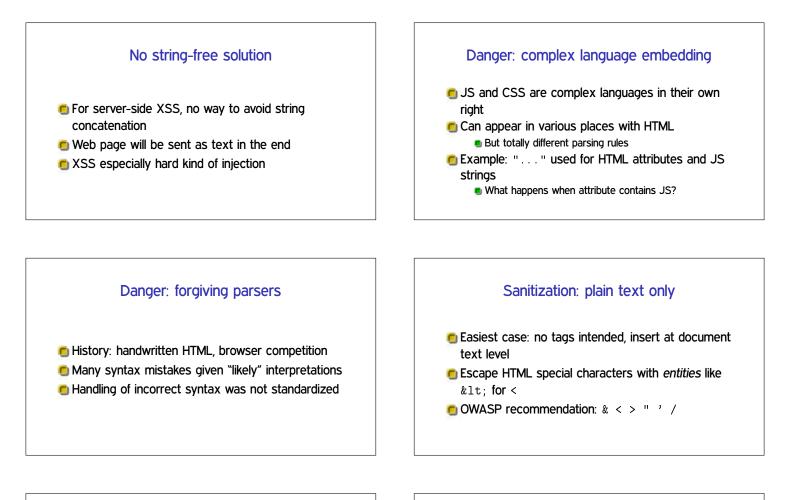
Persistent XSS

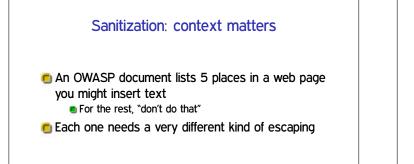
Injected data used to produce page later

- 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

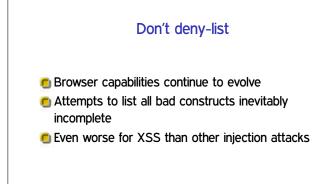
- Injection occurs in client-side page construction
- Flaw at least partially in code running on client
- Many attacks involve mashups and inter-site communication







- 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



Filter failure: one-pass delete

Simple idea: remove all occurrences of <script>
 Mhat 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

- 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

