Ideal: least privilege

- Programs and users should have the most limited set of powers needed to do their job
- Presupposes that privileges are suitably divisible
  - Contrast: Unix root

"Trusted", TCB

- In security, "trusted" is a bad word
- X is trusted: X can break your security
- "Untrusted" = okay if it's evil
- Trusted Computing Base (TCB): minimize

Restricted languages

- Main application: code provided by untrusted parties
- Packet filters in the kernel
- JavaScript in web browsers
  - Also Java, Flash ActionScript, etc.

SFI

- Software-based Fault Isolation
- Instruction-level rewriting
  - Analogous to but predates control-flow integrity
- Limit memory stores and sometimes loads
- Can't jump out except to designated points
- E.g., Google Native Client

Separate processes

- OS (and hardware) isolate one process from another
- Pay overhead for creation and communication
- System call interface allows many possibilities for mischief

System-call interposition

- Trusted process examines syscalls made by untrusted
- Implement via ptrace (like strace, gdb) or via kernel change
- Easy policy: deny
Interposition challenges
- Argument values can change in memory (TOCTTOU)
- OS objects can change (TOCTTOU)
- How to get canonical object identifiers?
- Interposer must accurately model kernel behavior
- Details: Garfinkel (NDSS’03)

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/
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: 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

More choices for isolation
The web from a security perspective
Cross-site scripting

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

Why XSS is bad (and named that)

- attacker.com can send you evil JS directly
- But XSS allows access to bank.com 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

No string-free solution

- For server-side XSS, no way to avoid string concatenation
- Web page will be sent as text in the end
  - Research topic: ways to change this?
- XSS especially hard kind of injection
### Danger: complex language embedding
- JS and CSS are complex languages in their own right.
- Can appear in various places with HTML.
  - But totally different parsing rules.
- Example: "..." used for HTML attributes and JS strings.
  - What happens when attribute contains JS?

### Danger: forgiving parsers
- History: handwritten HTML, browser competition.
- Many syntax mistakes given "likely" interpretations.
- Handling of incorrect syntax was not standardized.

### Sanitization: plain text only
- Easiest case: no tags intended, insert at document text level.
- Escape HTML special characters with entities like &lt; for <.
- 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 <b>.
- 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
- Browser capabilities continue to evolve.
- Attempts to list all bad constructs inevitably incomplete.
- Even worse for XSS than other injection attacks.

### Filter failure: one-pass delete
- Simple idea: remove all occurrences of <script>.
- What 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

Content Security Policy

- Added HTTP header, W3C recommendation
- Lets site opt-in to stricter treatment of embedded content, such as:
  - No inline JS, only loaded from separate URLs
  - Disable JS `eval` et al.
- Has an interesting violation-reporting mode