### CSci 4271W Development of Secure Software Systems Day 13: Isolation, Web Security part 1

Stephen McCamant University of Minnesota, Computer Science & Engineering

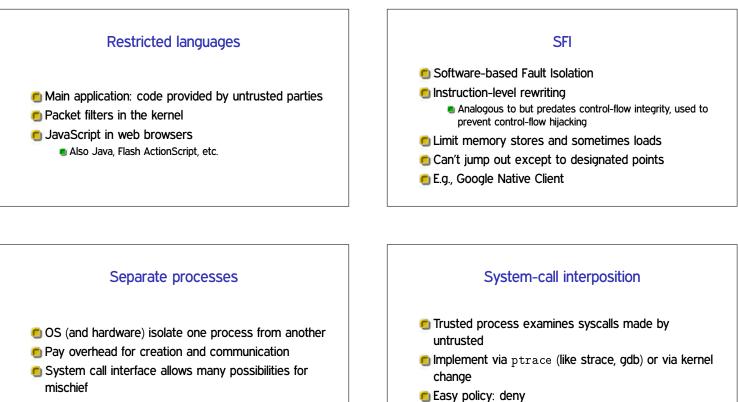
### Outline

Isolation mechanisms

The web from a security perspective

Cross-site scripting

# Ideal: least privilege "Trusted", TCB Programs and users should have the most limited set of powers needed to do their job In security, "trusted" is a bad word Presupposes that privileges are suitably divisible X is trusted: X can break your security Contrast: Unix root "Untrusted" = okay if it's evil Trusted Computing Base (TCB): minimize



### 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
   Free RCD is the line way of the line o
- 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

Isolation mechanisms

The web from a security perspective

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: 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 request loads a URL, may have parameters delimited with ?, &, = Standard: should not have side-effects

GET, POST, and cookies

- 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

**Isolation mechanisms** 

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)

end tacker.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

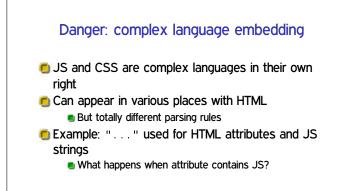
### Persistent XSS

- 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



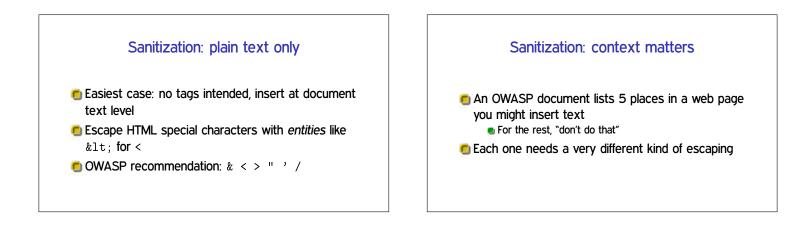
- For server-side XSS, no way to avoid string concatenation
- Web page will be sent as text in the end Research topic: ways to change this?
- XSS especially hard kind of injection



### Danger: forgiving parsers

History: handwritten HTML, browser competition

- Many syntax mistakes given "likely" interpretations
- Handling of incorrect syntax was not standardized

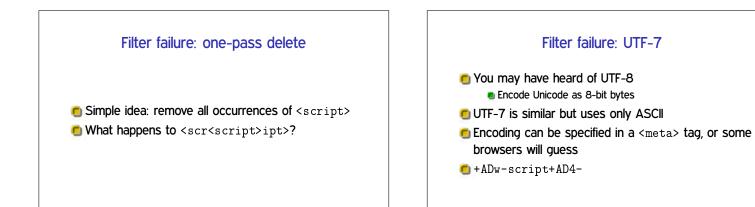


### Sanitization: tag allow-listing

- 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: event handlers

<IMG onmouseover="alert('xss')">

- 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