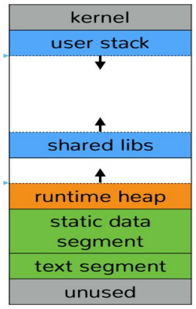
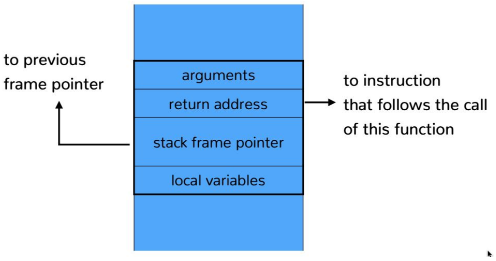
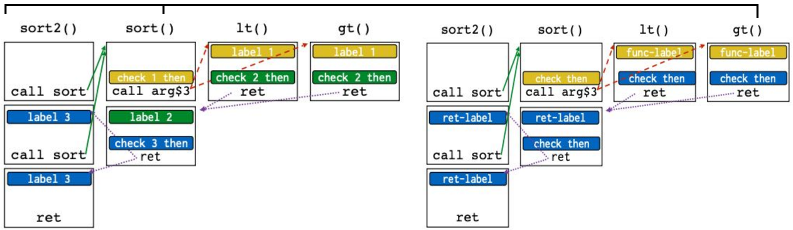



to previous frame pointer
arguments
return address
stack frame pointer
local variables
to instruction that follows the call of this function

Avoid unsafe funct	Good idea in general	Requires manual code rewrite Non-lib funct vulnerable No guarantee everything found Alternatives also error
Stack canary	No code changes Only recompile	Performance penalty per return Only protects against stack smashing Fails if can read memory
ASLR	No code changes or recompile	32-bit arch get limited protection Fails if can read memory Load-time overhead
W^X	No code changes or recompile	Requires hardware support Defeat by ROP; Not protect JIT code
CFI	No code changes or Hardware support Protect many vulns	Performance overhead Requires smarter compiler Need all code available (see)

Stack canary	Use targeted write gadget (format strings); pointer subterfuge; overwrite funct ptr elsewhere; memcpy with fixed canary
Separa te stack	Find a funct ptr and overwrite it to point to shellcode; Put buffers, &var, and function pointers on the **user stack** such that overwrite function pointers when c programs compiled to WebAssembly
W^X	Write to stack &jmp to existing code; find system call, replace args
ASLR	Derandomize (brute force for 32; heap spray for 64); find mapped region and call system() with replaced args
CFI	Jmp to funct that has the same label, then return to more sites
Int Ove -flow	Truncation (assign 64 to 32); arithmetic overflow (0xffffffff + 2); Signedness bugs (0xffffffff = -1 > some num)

ROP gadgets: overwrite saved %eip to pinter to the first gadget, then 2nd …
Make shellcode out of existing code, ending in ret inst (ending in 0xc3 in mem)
UAF: overwrite vtable so entry points to attacker gadget (tmp mem violation)



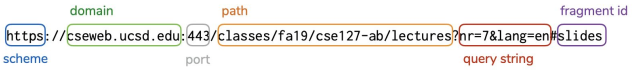
RUID	Inherit from parent; denote who starts the process
EUID	From setuid bit on file executed; determines permission
SUID (save)	Save and restore EUID
SetUID	Setuid: set EUID of procee; Setgid: set EGID; sticky: owner/user has the permission

Sticky bit examples: Andriod apps has its own process ID, commu limited using UNIX domain sockets + ref monitor checks permission; OKWS each server runs with unique UID, commu limited to structured RPC; modern browsers process; Qubes OS, trusted domain.

Seccomp-bpf: browser side syscall filtering on args
MEM isolation: each process has its own VM
Use page table to translate VM to PM (multi-level page table walking - tree)
ACL - determines page's access control information (R, W, X)
TLB - small cache of recently translated (faster); flush when context switch
Iso in VM: extended nested page table entries (VPID)
Defeat: find a bug in the kernel or hypervisor; find a hardware bug; exploit side-channels (cache based - use cache to learn about other process, VM)
Side-channels: evict & time; prime & probe (time, slower means evicted); flush & reload (flush the cache, faster means evicted)

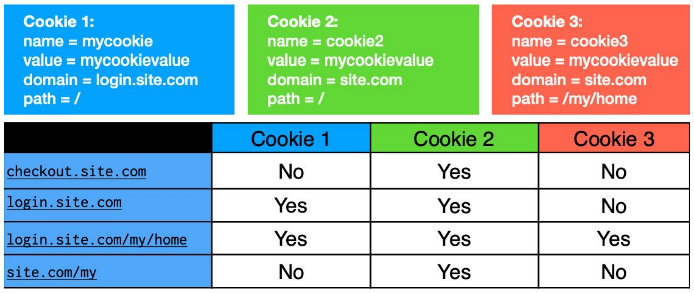
Malware
Virus: eventually be executed ←→ Worm: immediately be executed

HTTP:


domain
path
fragment id
https://cseweb.ucsd.edu:443/classes/fa19/cse127-ab/lectures?hr=7&lang=en#slides
scheme
port
query string

HTTP/2: allows **pipelining** requests for multi objects; multiplexing multiple requests over one TCP connection; header compression; server push
Cookies: small piece of data server sends to browser, browser updates and sends it back with subsequent requests.

SOP: origin: isolation unit/trust boundary (scheme, domain, port)
Isolate content of different origins;
SOP for DOM: each frame has its own origin; can only access data with the same origin; commu using **postmessage API**
SOP for HTTP responses: prevents code from directly inspecting HTTP responses; documents: can load cross origin but not inspect or modify frame content; scripts: can load cross origin, exe with same privilege of the page; images, fonts, css: can render cross origin but not inspecting each pixel
SOP for cookies: origin (scheme, domain, path) browser makes cookie available to **given domain + sub-domains**


Cookie 1:
name = mycookie
value = mycookievalue
domain = login.site.com
path = /

Cookie 2:
name = cookie2
value = mycookievalue
domain = site.com
path = /

Cookie 3:
name = cookie3
value = mycookievalue
domain = site.com
path = /my/home

	Cookie 1	Cookie 2	Cookie 3
checkout.site.com	No	Yes	No
login.site.com	Yes	Yes	No
login.site.com/my/home	Yes	Yes	Yes
site.com/my	No	Yes	No

CSRF: use attacker's domain to interact with banks url with user's cookie; submit transfer form from attacker's site with user's cookie
Defense: secret token validation (session-dependent identifier or token so attacker cannot retrive due to SOP - attacker's site has different origin); referer or origin validation: includes url of the previous web page; Samesite cookies: **strict: never** send cookie in any cross site browsing context; Lax: allowed when following a navigation link but blocks it in CSRF-prone request method; None: send cookies from any context

Injection:
Command injection: execute command on system b ypassing unsafe data into shell (./head10 "myfile.txt; rm -rf /home");
Code injection: eval function
SQL injection: take user input and add it to SQL string; prevention: never build SQL commands by urself. Use parameterized (AKA prepared) SQL; ORM (object relational mappers) (provide interface between obj & DBs)

Cross-Site Scripting (XSS)
App takes untrusted data and sends it to a web browser w/o proper validate

Command/SQL Injection

attacker's malicious code is executed on victim's server

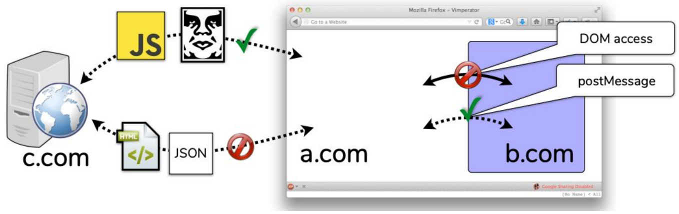
Cross Site Scripting

attacker's malicious code is executed on victim's browser

Reflected XSS: script reflected back to the user as part of a page from the victim site
Stored XSS: stores the malicious code in a resource managed by the web app (DB)
Defense: Content security policy (CSP): specify domains that browser should consider to be valid sources of exe scripts (set up in HTTP header)

Recall: SOP

Isolate content from different origins



Not strict enough: Third-party libs run with the **same privilege of the page;** Code within page can arbitrarily leak data; Iframe isolation is limited

Not flexible enough: Cannot read cross-origin responses

Modern Web Defense Mechanism

Iframe sandbox: Restrict actions iframe can perform;

Whitelisting privileges

allow-scripts: allows JS + triggers (autofocus, autoplay, etc.)

allow-forms: allow form submission

allow-pointer-lock: allow fine-grained mouse moves

allow-popups: allow iframe to create popups

allow-top-navigation: allow breaking out of frame

allow-same-origin: retain original origin

Iframe sandbox can: Run content in iframe with least privilege; Privilege separate page into multiple iframes

CSP: Consider running library in sandboxed iframes (**desired guarantee: checker cannot leak password**); **Problem:** sandbox does not restrict exfiltration; Restrict resource loading to a whitelist

HTTP strict transport security (HSTS): Attackers can force you to go to HTTP vs. HTTPS; HSTS: never visit site over HTTP again

Subresource integrity (SRI): CSP + HSTS can be used to limit damages but cannot really defend against malicious code; Idea: page author specifies hash of (sub)resource they are loading; browser checks integrity; When check fails: 1. Browser reports violation and does not render or execute resource; 2. CSP directive with integrity-policy directive set to report (report but may render or execute)

Cross-origin resource sharing (CORS)

Recall: SOP is not flexible: Problem: cannot fetch cross-origin data Solution: cross-origin resource sharing (CORS);Data provider explicitly whitelists origins that can inspect responsesBrowser allows page to inspect response if its origin is listed in the header

How it works: Browser send origin header with XHR request; Server can inspect origin header and respond with access-control-allow-origin header; CORS XHR may send cookies + custom headers

Example: amazon (multiple domains, Problem: amazon.com cannot read cross origin aws.com data; with CORS amazon.com can whitelist aws.com)

COWL

Provide means for associating security label with data

Ensure code is confined to obey labels by associating labels with browsing contexts

Confining the checker with COWL: Express sensitivity of data (checker only receive pw if its context label is as sensitive as the pw); Use postMessage to send labeled pw (at time of sending source, specify the sensitivity of the data)

Other:

A temporary err: violation by using a pointer whose ref has been deallocated; A spatial err: violation by dereferencing a pointer that refers to an address outside the bounds of its referent.

Heap corruption:

Bypass secuity checks (isAytgebtucated, buffer size, isAdmin, etc); overwrite function pointers (vtables); each object contains a pointer to vtable; vtable is an array of function pointers; call looks up entry in vtable

OS defense level (coarse → fine): physical machine → VM → OS process → Library → function