MIT 6.858 - Computer Systems Security

INTRODUCTION, THREAT MODELS

- Security
  - Policy - what you want to achieve (confidentiality, integrity)
  - Threat Model - assumptions about bad guy
  - Mechanism - SW/HW implementation
- Negative goal - e.g. nobody but X can access Y
- Adding security questions to passwords weakens security. more vulnerabilities
- Threat Model weaknesses
  - Humans
  - Time - computers constantly improving
- SSL Certificates
  - Written in C, null terminated strings
  - can request google.com\0x.foo.com\0, stops at first \0
- Buffer overflow
  - stack grow down - overflow w/ "outer" method (caller of this)
  - stack grow up - overflow w/ "inner" method (called inside method

2. CONTROL HIJACKING ATTACKS

- Buffer overflows
  - Take advantage of
    - System software is written in C
    - C exposes raw memory addresses
    - No bounds checking
    - Knowledge of x86 (stack, registers, calling conv.)
  - Can run with priority of the function it overtakes
  - Hardware is watching you all the time, not OS
- Avoiding Buffer Overflow
  - Avoid bugs in the first place
    - Don't use gets(), gotos, etc. → although these are not always used
  - Build tools to fix bugs
    - Static analysis
    - Fuzzing
  - Use memory safe languages
    - Usually not possible because
      - Legacy
      - Need low level access
      - Performance (less of an issue now)

- Buffer overflows exploit
  - Gaining control instruction pointer
  - Make it point to malicious code

- Stack canaries
  - Put a value before return address (canary), check it before jumping to return address
  - Problem - attacker can guess/find canary
    - One solution is to have canary be /0, carriage return, line feed,
    - Another solution is a randomized value
  - When they fail
    - Overwriting pointer functions

```
int *ptr = m;   ← overwrite here
...
*ptr = 5;
```

    - If you can guess a random canary
    - malloc and free attack
      - overflow overwrites size data of malloc block

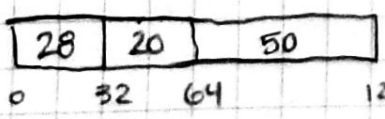
- Bounds checking
  - Hard to tell what you want, esp with structs + unions
  - For a pointer p' that's derived from p, then p' should only be used to deref memory that belongs to p

- Electric fences
  - set guard page on heap, if pointer hits guard page it faul
    - very space intensive because it takes a whole page

- Fat Pointers
  - Add bounds info to pointer
  - 3x size: start, end, current
  - Can cause concurrency bugs

- Shadow data structure
  - For each object, store size too
  - Can't check mathematical bounds by itself because for loops, etc must be broken once and become invalid to stop

- Buddy Allocation
  - Treats unallocated memory as one big block
    - Splits by powers of two until memory is just big enough
    - Splits until actual size ≥ 1/2 size of block



→ when freed, combines 2 of same size

WASTEFUL!

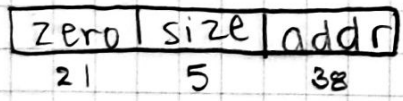
- Baggy Bounds
  - Round up each allocation to power of 2, align request to power
  - Express each bound as $\log_2$ of allocation size
  - Store limit info in linear array with 1 byte/entry
  - Allocate mem. at slot granularity

```
WITH SLOT SIZE 16 FOR EXAMPLE
p' = p+i;
size = 1<< +[p>> log of slot size];
base = p & ~(size -1)  ← clear bits that are offset of base
```

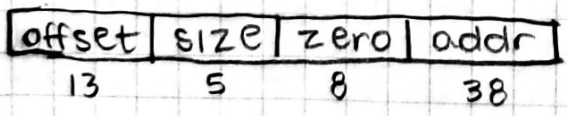
  - Uses virtual memory system to preven OOB dereference

3. BUFFER OVERFLOW EXPLOITS AND DEFENSES

- Baggy bounds throws error if you get beyond half the slot
  - If out of bounds but < half, set high order bit so if it's dereferenced it throws a hard error
- When we access the table the entry # means pointer size is 2 raised to that number
  - Max is automatically set to 31
  - Instrumented code
- Can't detect a bad pointer from instrumented to uninstrumented
- Regular pointer



- OOB



still 64bit addresses

- Still problems
  - Can't look at uninstrumented code
  - Could still be within baggy bounds if pointer
- Costs of baggy bounds
  - Space - store bounds table
  - CPU overhead to check bounds
  - False alarm - make OOB ptr, but doesn't dereference
  - Need compiler support

- Non-executable memory
  - Paging hardware writes 3 bits for each page (R, W, X)
    - W^X (can't do both)
  - Makes it hard to dynamically write code
    - Just in time compiler - turn common func into x86

- Randomized address spaces
  - A lot of attacks use hard code addresses
  - Stack randomization
    - they can just guess seed

- Heap Attack
  - Stuffs a bunch of shell code into memory
  - Jump to random
  - No-op sleds
    - Tons of no-ops, actual code at end so you just fall through to there

- Change system call numbers to be random (e.g. stop is 4, etc)
- some people suggest a hardware change, adding a XOR key that's XOR'd with each instruction

- GCC, Vscode use canaries
- Linux, Windows use non exec memory

- Return oriented programming
  - you can fake a calling frame, put it in the buffer
  - Gadgets
    - Small set of assembly instructions
  - Works with non-executable stack
  - Uses stack pointer as instruction pointer

- Guessing canaries
  - Assume server has BOF vulnerability
  - Server will crash and restart on bad canary
  - After restart, canary + addresses are not reset
    - Fork inherits canary, etc
  - Probe each next byte

- Defeat by catching segfault and staying open
  - Attacker thinks guess is correct

- 32 bit machine, arguments are in stack
  - 64 bit - registers

- Blind Return Oriented Programming
  - Find stop gadget - pause program but not crash
  - Find gadgets that pop stack entries
    - Example

STOP : 0x5
TRAP : 0x0
PROBE: 0x4..8 ← say this is pop rax, ret

If probe works, pops trap, returns to stop, all good. If not, returns to trap, crashes

To find more pops, just add more traps in between

- Know a pop happens, but not which reg its popped to
  - Take advantage of 'pause' call, put it in between gadgets, get it to hang and inspect registers
  - Try to find a gadget that pops pause # to rax, then call syscall(), if it's right # it will pause, else crash
- Invoke write()
  - Need pop rdi, rsi, rdx, rax, invoke syscall()

socket buf buf len syscall #

- How to defeat ROP
  - Use exec() instead of fork() so values are re-randomized
  - Use windows - no fork() equivalent
  - When crash happens, don't close connection
  - NOT A GOOD IDEA - hashing time of day
    - Bits of entropy are reduced if input is guessable (eg. time)

4. PRIVILEGE SEPARATION

- If app that attacker gains access to has privileged status, it can affect much more besides itself
  - Privilege separation aims to break app into smaller pieces, only give them access to what they need
  - VMs are used for this
  - Unix provides you with tools

- UNIX (32 bit int → [userID, groupID])
  - Principal: What privileges the subject has
  - Subject: process - single UID, list of GID
  - Object: What to protect (files, dirs, sockets, other proc)
    - Files - R, W, X, chmod
    - Directory - unlink, link, rename, create, lookup } inode
      - owned by uid/gid - can change perm if uid is same
      - permission bits for uid, gid, other

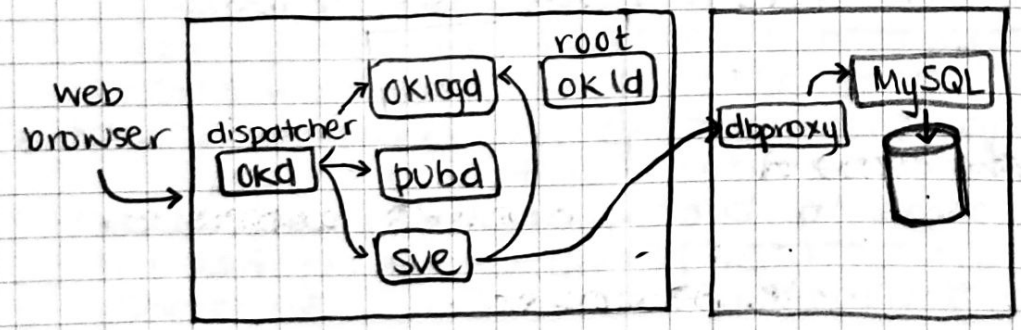
open("/etc/passwd") - need x /, x /etc, r /etc/passwd

- can combine permissions in layers

  e.g. ∃ 6.858 gid, TA gid, ∄ 6.858 TA gid
  Make /foo/bar/file, give foo 6.858 perm, bar TA perm

- File descriptors - allows you to open file
  - can be passed to you
- Process - create, kill, debug (ptrace)
  - Needs same uid
- Networking - listen, connect, read/write, send/recieve
  - Anyone can connect
  - Need to be su to listen to port <1024 (uid = 0)
- Creating UIDs
  - Can set_uid if root (uid=0), also set_gid and set_groups
  - Login process runs as root, checks passwd from /etc/passwd and /etc/shadow
- Set uid binaries
  - When you exec, switches uid of process to owner of binary
  - Have to be careful with what environment variables you pass
- How to avoid setuid issues
  - Call chroot("/foo")
    - / = /foo (restrict namespace)
    - Prevents /../ bc its now /foo/../
    - Need to be uid 0

- OKWS: Case Study



- Need token to call dbproxy
- Each process has a specific uid

- okd and sve share same chroot
- okld needs root bc port 80
- Why don't they have true userids instead of service ids?
  - Performance overheads
  - Depends on having access to all data at once

6. CAPABILITIES

- Capabilities let you control privileges better than Unix
- 2 examples: confused deputy, sandboxing

- Confused deputy
  - Had FORTRAN compiler in /sysx/fort
    - Wanted it to write usage statistics into /sysx/stat
  - Home files license - gives you extra privileges, can access files in /sysx/*
  - Someone told a program to output to /sysx/bill, for example, since the compiler had Home files License even though user didn't
  - 2 sources of privilege
    - User invoking
    - Home files license
    } their design used a union of this
  - What goes wrong?
    - Ambient authority - making an operation, but decision of whether it succeeds comes from outside parameters
    - Access control checks are complicated
      - If you try to replicate check system, maybe you miss some, or kernel updates and adds new checks and your app doesn't keep up
  - Capability has no ambient authority
    - Capability points to object, if you have capability you can access object

- Capability
  - Need to be unforgable
  - File descriptor - just an integer
    - Kernel interprets based on descriptor table
    - To forge a file descriptor, you have to pick a number that refers to a non-null entry in table, but those are already the capabilities you have

- Solving Confused Deputy
  - Don't pass file names around
  - Frontend turns input files into file descriptors, passes them
    - If a user can't open it, it fails
  - Interpeter then deals with file descriptors

- Capabilities - can't talk about object without conveying rights to that object
- Privileges don't stack

- Sandboxing
  - Network stuff - how much could a hacker control
  - Best if dev knows it will be run in sandbox
  - Why not use VM?
    - Memory overhead
    - Network issues
    - Hard to share data

- How to sandbox
  - Unix
    - Change permissions, make sure there aren't world readable files
  - capsicum
    - Once you enter sandbox, everything depends on capabilities
    - Global namespace (e.g. file system)
    - Can't make it just a library, needs to be a kernel thing
      - Could still make syscall
    - When you enter capsicum, stops all syscalls to global namespaces
      - can't open sockets, need to set them up beforehand
    - How do you kill processes?
      - Using a file descriptor
    - What about using ..?
      - Disallow, just to be safe
    - Once you enter capability mode, does UID matter?
      - Yes to be compatible w/ general Unix permissions
    - Why do they need a library?
      - To use function lch_start (vs. cap_enter)
        - Erases all inheritance stuff, start a process fresh
      - Fd lists
        - Generalization of how Linux handles file descriptors
    - Don't allow setuid binaries
    - In general, can't gain any privileges

7: SANDBOXING NATIVE CODE

- Native client
  - Software fault isolation

- Why do you need to run native code? (In the case of browser)
  - Performance
  - Existing "Legacy" code
  - Don't need to use Javascript
  - Other languages

- How to use Native Client
  - Need some JS to connect webpage to Na. Cl. module
  - Can contain crashes, module goes down but page doesn't

- Alternatives
  - Trust the developer/ask user
    - Insecure, can't tell if binary is safe (as user)
    - Native client ensures it's safe, and protects you
  - OS/Hardware isolation
    - Worried about OS bugs

- OSs could be incompatible, need diff isolation mechanism
- Hardware bugs

- Software Fault Isolation
  - Don't rely on OS/HW to see if it's safe at runtime, look ahead to see if it's safe
    - Safe - allow (e.g. ALU, math, movs)
    - Unsafe - (e.g. mem access, syscall, privileged instruction)
      - Necessary - (e.g. mem access) instrument, add in checks, make safe
      - Unnecessary - prohibit, kill application
  - Since it can't access memory, network, etc., need to add a Trusted service runtime
    - Not checked
    - Things like alloc, threads

- Safety
  - No disallowed instructions can execute
    - syscall, privileged instructions
  - All code/data accesses are in bounds
    - [0... 256MB] in module is safe
    - Without, you could do disallowed instructions by jumping wherever
    - This means one module per process since you need low addresses

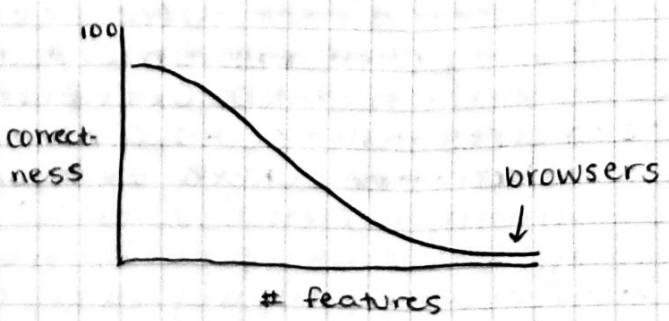
- How to Implement
  - Scanning instructions
    - Problem: variable length instructions (need to look at the first couple of bytes to see how long it will be)
    - Could also jump to middle of instruction
    - Look at every offset?
      - No bc false positives
  - How they actually do it
    - Need to deal with jumps
      - So they check where the jumps go
      - If it jumps somewhere we haven't seen (e.g. we saw 0x20 but not 0x22 bc instruction is 4 bytes long not 2) flag it
      - Issue - indirect jumps (e.g. jmp %eax)
        - Solve with instrumentation
        - Add an 'AND 0xffffffe0' to clear lowest 5 bits, forces value to be a multiple of 32 (other threads can't break)
        - Then make sure everything at addresses that are multiples of 32 are safe
        - Why 32? Power of 2, fits longest instruction, but not too big
        - Prevents jumping over and by treating the and+jmp as one instruction

- Rules
  - C1 - binary is not writeable after loading
  - C2 - linked @ 0, starts @ 64K
  - C3 - indirect jumps use and + jmp
  - C4 - pad out to page boundary w/ halt (safety buffer)
  - C5 - no instructions span over 32 byte boundary
  - C6 - all instr. reachable by disassembly
  - C7 - all direct jumps are ok
- Bounds checking
  - Zero out upper bytes
- Segmentation in x86
  - Whenever a process is running, hardware makes segment descriptor table
    - Segments numbered
    - Each entry has base and length
  - Whenever x86 talks about memory, it uses this table, talking about a specific segment
    - Segment selectors (16 bit) are cs, ds, es, fs, gs, ss
      - cs - code selector
      - ds/es - for memory
      - ss - stack
  - Manipulate table with system calls
  - Tables are per process

- How to jump out of Native Client
  - Change the selectors
  - Trampolines and springboards
    - Trampoline code comes from trusted runtime, not NaCl
  - Trampoline
    - move values into selectors
    - jump into normal runtime
  - Springboard - jump back into NaCl
    - restore selectors
    - jmp into NaCl module
    - put halt at front so NaCl cant jump into the springboard itself

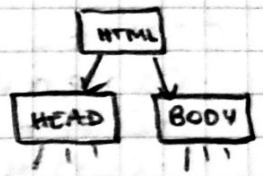
8: WEB SECURITY MODEL

- Browser is complicated today
  - JavaScript
  - DOM Model
  - XMLHTTP request (AJAX)
  - Web sockets
  - Multimedia support
  - Geolocation
  - NativeClient



- Parsing contexts
  - <script> var x = "UNTRUSTED"; </script>
  - Could put in JS
  - Could put in </script> and add in HTML

- Same-origin policy
  - 2 websites should not be able to tamper with each other unless they want to
  - strategy
    - Each resource is assigned an origin
    - JS can only access resources from its own origin
    - Origin: network protocol scheme + hostname + port
      - http://foo.com/index.html = HTTP, foo.com, 80
      - https://foo.com/index.html = HTTPS, foo.com, 443
      - Like a UID
  - 4 ideas
    - Each origin has client-side resources
      - Cookies - tiny file associated with origin
      - DOM Storage - key/value storage
      - JS namespace
      - DOM tree - tree of HTML structure
      - Visual display area
    - Each frame gets origin of its URL
    - Scripts execute with the authority of its frame's origin
    - Passive content gets ∅ authority
      - Non executable things like images/CSS
  - postMessage() allows 2 frames to communicate



DOM tree

- MIME Sniffing Attack
  - MIME type - text/html, jpg, etc.
  - Web servers could misattribute MIME types
  - IE used to look at first 256 bytes to guess what type of object it is
  - Could coerce an image into HTML/JS content + execute it

- Frames and Window Objects - Same Origin Policy
  - Frame: instance of a DOM node
  - Window: alias for global namespace
  - Contain pointers to one another
  - Get origin of frame's URL OR get suffix of original domain
  - Two frames can access each other if
    - Both frames set document.domain to same val
    - Neither have changed document.domain + vals match
      - Prevents domains from being attacked by subdomains

- Cookies
  - Have domain + path

- Can be set by server or on client side
  - Client can change cookie w/ JS document.cookie
- Frames accessing other cookies
  - If they have same doc.domain and both changed it

- XMLHTTPRequest
  - Can only generate one if its going to its origin server
  - Same origin unless Cross Origin Request (CORS) enabled
    - Adds HTTP header called access-control-allow-origin: <URL>

- Images
  - Frames can load images from wrerever
    - Can't directly inspect bits
    - You can still get, say, size info based on what it affects

- CSS
  - Can embed from any origin
    - Can't inspect file
    - can infer based on what nodes are created

- JavaScript
  - Can cross origin fetch
    - Can't inspect source code
    - Since functions are objects though, you can call f.toString
  - People try to hide by obfuscating and min-ifying

- Plugins
  - Almost out of use

- Cross Site Request Forgery (CSRF)
  - Example: http://bank.com/transfer?amount=5000&to=hacker
  - Has all the users cookies, so it'll run
  - Fix by introducing randomness the attacker can't guess
    - Add a random token as a query

- Network Addresses
  - DNS rebinding attack
    - Run malicious JS with authority of the website
    - Steps
      - Register a domain name (e.g. attacker.com)
      - User visits attacker.com
      - Browser generates DNS request to attacker.com, attacker responds with something that will expire soon (needs to be re-checked soon)
      - Attacker binds attacker.com to user's IP address
      - Website fetches new object, but this goes to victim.com
    - Can continually rebind

- Fixes
  - Make it impossible to resolve external domains to internal IPs
  - could also ignore TTL and keep for, say, 30 min

- Click Jacking Attack
  - Pixels are bounded by frames
  - parent frames can write over child frames
  - Hide malicious frames behind normal ones
  - solution
    - Frame busting code
      - if (self != top) ← JS
    - X-Frame-Options HTTP Header
      - Don't allow anyone to put my content inside a frame

- Typo squatting Attacks
  - Attacker can buy Cats.com with Cyrillic C instead of Latin

- Plugins
  - Java assumes same IP, diff host = same origin

- Screensharing
  - can screenshot outside of the frame

9: SECURING WEB APPLICATIONS

- Shellshock
  - You can specify custom HTTP headers
  - () { :; };

- XSS
  - Malformed URL can bypass XSS protection
  - Defense
    - XSS filters in the browser
      - Checks if <script> is in URL
      - can't prevent persistent XSS
    - HTTP Only Cookies
    - Privilege separation
      - Use a separate domain for untrusted data
    - Content sanitization
      - Don't trust any user input
      - < → &lt
      - " → &quot
    - Content Security Policy
      - Tell browser what content can be loaded and from where
      - Sent in HTTP Header

- SQL
  - Escape SQL queries
  - Solution - rigorously encode your data
    - Character escaping
- Cookies
  - Session management
  - Stateless cookie - avoid attacks?
    - Need to then authenticate every request
      - MAC - Message Authentication Codes
        - Hash key with message
    - Use DOM client side storage
      - Less susceptible to SOP attacks
    - Client side certificates
      - Revocation is hard
- Protocol Vulnerabilities in Web Stack
  - URL Parsing
    - http://example.com:80@foo.com
      - Which is the origin?
        - Flash says example
        - Browser says
  - JAR - GIFAR attack
    - Can combine .gif and .zip, bc .gif reads top down, .zip reads bottom up
    - JAR files are derivatives of .zip
    - Embed JS in GIF
      - Can be used in conjunction with XSS attack
    - Also works with PDF
- Covert Channel Attack
  - CSS-based Sniffing Attack
    - Attacker wants to know what websites user visited
    - Get them to visit a site the attacker controls
    - Generate a list of URLs the user may have visited, then use JS to see what color they are
    - Solution - Browser lies to JS about link colors
  - Cache-Based Attack
    - Time response time for various websites
    - Shorter time = cached = visited
    - Can use this on something like Google Map tiles
      - Visits = maybe you live there
  - DNS based attack
    - Check how long DNS requests take
    - Shorter time = you visited
    - caching doesn't matter since DNS is OS stuff

- Rendering attacks
  - Faster to render a URL you have visited
  - When an iframe is loading, attacker owns it
  - When they lose it, they know it rendered

10: SYMBOLIC EXECUTION

$t_0 = \begin{cases} x > y \Rightarrow x \\ x \leq y \Rightarrow y \end{cases}$ when is $t_0 < x$

can also add preconditions, like $x \leq y$

SMT - will either return satisfying conditions or unsat.
↳ Satisfiability Modulo Theories

11: UR/WEB

- Full stack web development language

12: NETWORK SECURITY

- Threat Model
  - Intercept, modify packets
  - Inject packets
  - Attacker has computer
  - Distributed
- TCP
  - 3 way Handshake (32 BIT)
    - Client: C→S SYN(SEQUENCE NUM$_c$)
    - Server: S→C SYN(SN$_s$), ACK(SN$_c$)
    - Client: C→S ACK(SN$_c$)
    - SN is incremented per byte sent
  - The SN was easy to guess
    - The starting SN is bumped up by some 25, 64, 128K
- Attack: Forge another machines' IP
  - IP Based Authorization
    - SMTP still uses this
    - rlogin used this
  - RST
    - If you know a sequence number, you can send a RESET request
    - Multiple routers connecting to each other
      - If a TCP connection breaks, reset routing tables
      - DOS Attack
      - Routers ensure it's from the immediately next router by only accepting packets with TTL of 255 (max)
        - So you can see who's sending reset packets

- Data Injection
  - Inject bits into existing connection
- Prevention
  - General: 32 bit offset to add to initial sequence num

- DNS
  - Uses UDP - stateless
  - Just need SRC + DST IP and port (53)
  - Inject packets before you get your real response
    - It might even cache this as the correct IP
  - Solution
    - Add 16 bit query ID
    - Add 16 bit random +53
  - Non Existent Records
    - Have to be signed by server
    - NSEC - organizes domains alphabetically, if theres nothing in between, you know it doesn't exist
      - You can query for all the "gaps" to get the things on either side
      - e.g. [aaa - bbb] query for aab, doesn't exist, tells you what's on left and right (aaa, bbb). Then query for bbc, tells you bbb and ccc, continue ad nauseum

- SYN Flooding
  - Data structure with connection, SN_c and SN_s
  - DOS by sending a bunch of SYN packets, because it keeps all the connection/SN data it gets
  - Solution - only add to stack when you've authenticated the client
    - Do this by choosing SN a different way
    - SN = F(SRCIP, DSTIP, SRCPORT, DSTPORT, TIMESTAMP, KEY) + timestamp
    - No need to store the info until after handshake