CS161 Cheat sheet — Katrina Sharonin

Security principles

TCB: Trusted Comp. Base
Must work, unbypass, for sec.
Time of check to Time of use (TOCTTOU)
Race conditions btw check & use

1. Know your threat model
2. Human factors
3. Security is economics
4. Detect if you can't prevent
5. Defense in depth: layer mul. types
6. Least privilege: grant for correct functioning
7. Separation of resp.: require mul. ppl to get access
8. Ensure complete mediation: unbypassable, monitor accs
9. Shannon's maxim: the enemy knows the system
10. Fail-safe defaults: fail in safe state, balance security usability
11. Design in security from start: avoid patching

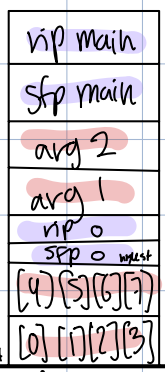
x86

stack: function frames
Heap: malloc/dynamic
Static: const strings, buffers in static mem
Code: x86 instr, when rip & eip point to!

x86 cont

Calling Convention

0x44332211 → 0x11 0x22 0x33 0x44 (0x1 0x2 0x3 0x4)



1. Push arguments onto stack (in reverse order). — main
2. Push old EIP onto stack (RIP). — main
3. Move EIP. — main
4. Push old EBP onto stack (SFP). — orbit
5. Move EBP down to ESP. — orbit
6. Move ESP down for new frame. — orbit
7. Execute the function. — orbit
8. Move ESP up to EBP. — orbit
9. Restore old EBP by popping SFP. — orbit
10. Restore old EIP by popping RIP. — orbit
11. Remove arguments from stack. — main

- Prologue: push args in rev order, push eip & mov eip, push old ebp (sfp), move ebp down to esp, mov esp down to end of new frame
- Epilogue: mov esp to ebp (remove local vars) pop ebp (+4 esp), pop eip (+4 esp), rm args

Mem Safety Vulnerabilities

- Buf overflow: gets() read() vs. fread() fgets: write to region above buf
- Stack smash: overflow to overwrite rip → pt to shellcode
- int conversion: size_t (unsigned, 0 → 255), int (signed, -128 → 127)

ex 0XFFFF = -1 vs. 2^16 (2^8 = 256)

☆ Two's comp: invert → +1 → get value + apply sign

- off-by-one: manipulate 1 byte of sfp ☆ USE NULL BYTE!
- format string: %c print as char, %<k>u print arg as unsigned int + white space → k chars ex "%7u\n", 123 → ____123 → 7 chars

%s treat as ptr → access & print as str (☆ till it sees '\0') %n writes num of bytes currently printed (4 byte) to address arg %hn 2 byte %x print as hex

- ret2ret: make rip point to ret instr; bumps & uses next ptr above rip as the eip ☆ write into wher above ptr targets

ex 000%n → write 3 since 3 chars %9$c → consumes arg 1

Defenses

1. Stack canaries (leak / guess / brute force)
2. Non-exec pages: W^X, DEP, No-execute bit (writable or exec) ~use existing code → ROP

③ PAC: func() auth'd by CPU ④ ASLR: shuffle segments (exploit w/ %x print)

Cryptography

Confid: cannot read msgs Integrity: not changed Auth: claimed auth

★ Integrity before authenticity — Enc-to-MAC

★ Kerckoff's: secure when details except key known

	Symm	Assym
Conf	• one time pads eg: • Block ciphers CBC • stream ciphers	• RSA enc • El Gamal enc
Int/ Auth	• MACs e.g. HMACs AES-CBC-MAC	• Digital signs. (e.g. RSA)

Threats: intercept C, known plaintext, resend enc (replay), chosen P (E forces A to encrypt), chosen C (decrypt), or trick into enc & decrypt

Symm-Key enc

IND-CPA: No & M_1 → send back C? → which one it is

XOR: $X \oplus Y = Y \oplus X$ $X \oplus (Y \oplus Z) = (X \oplus Y) \oplus Z$ $X \oplus X = 0$ $0 \oplus Y = 0$

One-time pad (OTP): key: shared n bit secret key K { Key cannot be reused! + expensive

Encryption: $C = M \oplus K$ Decryption: $M = C \oplus K$

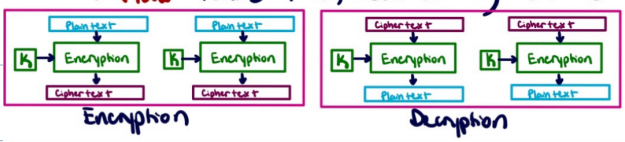
Block ciphers: key: agreed & key ★deterministic ★correct/efficient/"random"

Encrypt: $E_{nk}(M) = C$ (fixed size) $E: \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$ Dec: $D_k(E_k(M)) = M$

Block Modes of Op

★ AES: n = 128 block len, key => k = 128 bits

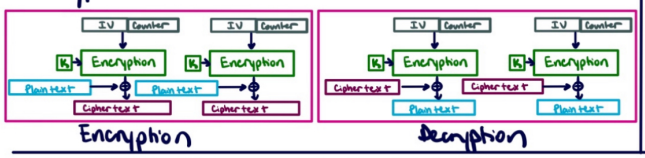
1) ECB Mode (Electronic Code Book):
break into n-bit blocks $M_1 \dots M_i$
Encryption: $C_i = E_K(M_i)$, concate blocks
Decryption: $M_i = D_K(C_i)$
↳ flaw: leaks info, redundancy shows



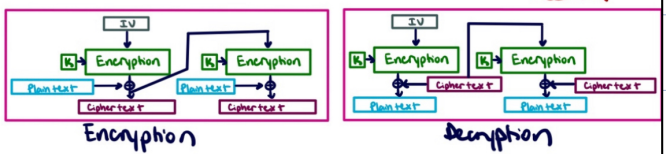
2) CBC Mode (Cipher Block Chaining): need to send IV
Choose random initial vector (IV), use prev
Encryption: $C_0 = IV$, $C_i = E_K(C_{i-1} \oplus M_i)$ Not Parallel
Decryption: $P_i = D_K(C_i) \oplus C_{i-1}$ Parallelizable

ECB not CPA
CBC secure
CFB no padding
CTR secure

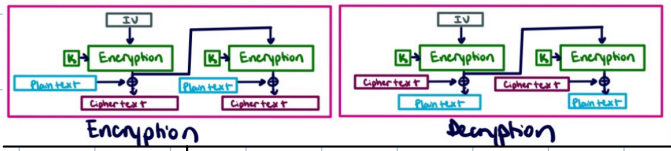
3) CTR Mode (Counter): counter initialized to IV, MALLEABLE
secure w/ predictable nonce, no reuse; incremented & encrypted, nonce = IV
Encryption: $C_i = E_K(IV+i) \oplus M_i$ Parallelizable
Decryption: $M_i = E_K(IV+i) \oplus C_i$ Parallelizable



4) CFB Mode (Ciphertext Feedback)
Encryption: $C_0 = IV$, $C_i = E_K(C_{i-1}) \oplus P_i$
Decryption: $P_i = E_K(C_{i-1}) \oplus C_i$ ★no padding needed



5) OFB Mode (Output Feedback)
Encryption: $Z_0 = IV$, $Z_i = E_K(Z_{i-1})$, $C_i = M_i \oplus Z_i$
Decryption: $P_i = C_i \oplus Z_i$



Padding: Need to make sure message is multiple of block cipher, CTR, CFB doesn't need
PKCS #7: pad message by num of padding bytes
- Don't reuse IV, makes it IND-CPA insecure
- CTR fails catastrophically, CBC is contained

★ Avalanche effect: $H(M) \neq H(M')$ very wild maps

Hashes

① One-way (infeasible to find x s.t. $H(x) = y$) ③ Collision resist x, x' s.t. $H(x) = H(x')$
② Pre-image resist ($x' \neq x$, $H(x) = H(x')$ hard to find)

MACs MAC(K, M) → Tag ☆deterministic/not confidential, but for integ/auth

① Enc-then-MAC ☆Best option $(MAC_{k_2}(Enc_{k_1}(M))$ ② MAC-then-enc $(Enc_{k_1}(M \| MAC_{k_2}(M))$

PRNGs seed (rand bits) → deterministic (but looks random!) ☆efficient ☆roll back resistance (not required for a secure PRNG) ☆reseeding adds more entropy can't learn abt past state

CTR-DRBG (use block CTR to gen pseudo rand bits) **HMAC-DRBG**: repeat HMAC(use) to gen pseudo rand bits

Stream Cipher symm enc, use pseudo rand bits (PRNG output) as one time pad (seed(K) gen)

Diffie-Hellman public g^a secret mod p ⟷ public g^b secret mod p ⇒ g^{ab} mod p } MITM tamper g^{am} mod p

Public-Key Enc ☆ given g^a & g^b mod p ⇒ hard to recover g^{ab} mod p

El Gamal: $E_B(M) = (g^r \bmod p, M \times B^r \bmod p)$ g: $1 < g < p-1$ large prime

B: $g^b \bmod p$ $M \times g^{br} \times g^{-br} = M \bmod p$ $D_b(R, C_2) = R^{-b} \times C_2 \bmod p$

☆ Not IND-CPA secure ($M_0 = 0, M_1 \neq 0$) ☆ Malleable/can be tampered e.g. 2M

RSA Enc: N = pq (2 large primes) e prime to (p-1)(q-1), $2 < e < (p-1)(q-1)$

$d = e^{-1} \bmod (p-1)(q-1)$ ⇒ **Public: N & e** **Private: d** $D() = (M^e)^d$

☆ given N & $C = M^e \bmod N$, hard to find M (can't factor N out)

☆ Not INDCPA secure since its deterministic ☆ Same m diff PK leaks info

OAEP: RSA but w/ randomness **Hybrid:** gen K (random) using symm enc, E(K) Assym ☆Almost all sys use hyb.

Signatures Keygen() → PK, SK Sign(SK, M) → sig Verify(PK, M, Sig) → 0,1

Verify(PK, M, Sign(SK, M)) = 1 for all PK, SK EU-CPA secure / integrity/auth

RSA Sign Keygen() → N & e, d ⇒ Sign(d, M) ⇒ $H(M)^d \bmod N$

Verify(e, N, M, sig) → $H(M) \equiv sig^e \bmod N$ ☆Simplified Subject to existential forg.

Certificates Idea: sign public key w/ an authority

☆Trust-first-use: first time → trust public key + warn if changes

PK: {M}$_{PK}$ SK: {M}$_{SK^{-1}}$ → {"Bob's PK is PK_B"}$_{SK_E^{-1}}$ (trust anchor)

☆Trusted directory (TD): fetch anyone's public key ⇒ Signed by SK_{TD} (Not scalable)

☆Hierarchial trust: root authority (root CA) (sign for) V.S. Intermediate CAs

☆Revocation: contain expiration date vs. CAs sign list of revoked certs

Passwords ☆ Need to verify pass w/o storing info you could recover with

Rainbow tables: algo for computing hashes that makes brute easier

Salt: random public value to make attacks harder: H(pass || salt)
↳ should be long + random, not secret (try pass & user) O(MN) to guess vs. O(M+N) (hash & lookup)

Slower hashes: add large constant factor, make it inconvenient

Offline: computation themselves (defend: salt / slow)

Online: interacts w/ service (defend: timeout / rate limit)

Traffic Analyzing who is talking to whom & when

Side channels: info about Plain revled bc of implementation

☆ good sys should desc how public param generated: **Nothing-Sleeve**

Extra tips/materials

0xffff1244: 0x00ab0000 0x00000000 0x00000033 0xff0
(46 45 44)

Q2.3 True or False: Byte 0xab appears at address 0xffff1246

Writing attacks ☆ +4 to addy if SHELL above

☆ "jmp *esp + SHELLCODE" : if above mp, it jumps to hi take fine

☆ x86 registers are stored in the processor

```
0x08988154: xor %ebx, %ebx
0x0898815c: mov %ebp, %esp
0x08988160: pop %edx   ← esp
0x08988164: pop %ecx
0x08988168: pop %ebx
0x0898816c: pop %eax
0x08988170: pop %ebp
0x0898816c: ret   ← pop eip
```

bump pointer

at spot in stack, deref & go

```
printf("%x", peter);
```

prints actual ptr address value, not string! vs. &peter

$C_i \oplus C_{i-1} = M_{i-1} \oplus E_K(M_i)$

C_i known by attacker; $M_{i-1} \oplus E_K(M_i)$ deterministic → vulnerable

gets(char *str) read from stdin, append null byte

fgets(char *str, int n, FILE *stream) n includes final null char

strcpy(char *dest, (const) char *source) reads till null byte

strncpy(char *dest, const char * src, size_t n) stops at size

memcpy(void * dest, const (un) *src, size_t n) n chars to dest

read(int fd, void * buf, size_t cnt) cnt == num bytes

fread(void * ptr, size_t size, size_t nmemb, FILE *stream) size * num elems

scanf(const char * format, ...)

fgets: append a null byte

fget(&buf) → write at buf aka addrit

fget(buf) → write at arr, dereference value in buf

Intro to Web

URL: uniquely ID data [path] [query]

http [protocol] : // toon.cs161.org [domain + info = location] / index.html ? int = 1

HTTP: req - response (to server) GET (doesn't change server) POST (send info to server, can modify)

HTML (structured doc) <a href="link"> <img src=" " > <iframe src="">

CSS (modify HTML appearance just as powerful as JS e.g. load mal CSS)

JS (client side, code sent by server, run in browser, DOM + JS in tire)

↳ secure via sandbox (privilege separation), server-side security, same org.

Same origin policy: port, domain (page name) protocol (http v. https)

↳ Exceptions: ① JS runs with the origin of the page that loads it ★ DN block requests "src = 'google.com'"...

② Images have the origin of the page it comes from

③ Frames have origin of URL where frame is retrieved from

Cookie anatomy: Domain (google.com) path (/...) Secure (if T, only send cookie over HTTPS)

HTTPOnly (if T, JS cannot access / modify cookie) Expires (when to stop store (set to -1 to delete)) Same Site = Strict (only rec sent in 1st party)

Cookie policy: document.cookie URL domain must end in cookie domain (suffix), path must begin w/ cookie path (prefix)

CSRF (cross site request Forgery): force user to make request, browser auto attach session token, server accepts

ex GET request ⇒ auto attach ssh

CSRF defense: CSRF token (new CSRF ⟷ session token, mapping needs to exist)

Referer Validation: ex form on bank.com vs. evil.com req; referer header

Same-Site Flag = strict → only send to 1st party ★ Same origin → leak value of CSRF

Cross-Site Scripting (XSS): executed on client side, injected JS

Stored XSS: store JS on server ex Post: <script>alert("!")</script>

Reflect XSS: server recieves user input → display input in response

ex force users to make req. ex request by embedded into iframe

UI attacks: trick user to think they are taking intended action

Clickjacking: trick into clicking thing from attacker (invisible frames, legit frame under, temporal attack (quick swap), cursor jack

↳ Defenses: visual integrity / separation, temporal delay, require confirm, frame busting

Phishing: trick into sending to attacker personal info

ex Homograph (same looking bad URL), browser-in-browser simulate

↳ Defenses: 2FA (subvert: relay attack (MITM) vs. security key

CAPTCHAs: distorted text/audio, ML train vs. arm race vs. pay

SQL Injection: ★ -- & ' important to work, UNION SELECT

↳ Defense: escape ex alice\' OR 1=1 --\- vs. \" input → \\"

↳ Defense: Param/prepare: compile query first, then plug input

XSS defense: Sanitize input ex

remove endings <, >, &lt, > can still be tricked

Content Security policy (CSP): ban any inline scripts, permit only certain ★ must load external scripts

↳ defined by web-server, enforced by browser

no direct command in HTML allowed

Network Security | Layer Attack + defense table

Layer	Name	Types
7	Application	DNS, DNSSEC
6.5	Secure transport	TLS (4-7)
4	Transport	TCP, UDP, ports
3	(Inter) Network	IP, DHCP, BGP
2	Link (inside LAN)	MAC, ARP, DHCP, WPA
1	Physical (not bits)	Wire

LAN: 1+ computers
Router: 2+ LANs

Type	Attack	defense
ARP (MAC of IP?)	on-path can race w/ a false mapping	switches → VLANs → arp watch
WPA	on-path: has pass → can get PSK; Eve: learn handshake off: brute password	WPA2-enterprise, auth server, sent over Enc channels AP relays
DHCP	on-path: see req & spoof G become MITM w/ own IP & gateway (domain → own IP)	higher layers
BGP	Malicious AS, forward to wrong spot / rogue AP	higher TCP resends, TLS (conf. + integrity)
TCP	on: see seq num + ports G inject + in path: full power off: guess 4 vals + seq.	Random seq num, higher layers
UDP	on: see ports → inj off: guess ports / len, no seq num	higher layers

Type	Attack	defense
TLS	No anon / availability; replay → RSA, RSA pms; on/MITM can spoof	On / MITM insecure, plaintext handshake
DNS	on: insecure, race w/ correct ID; on/off: TTL wait to replace Kaminsky	UDP port randomization G easy for on path; Glue record control
DNSSEC	Trust root + integrity + authenticity all path vdce	

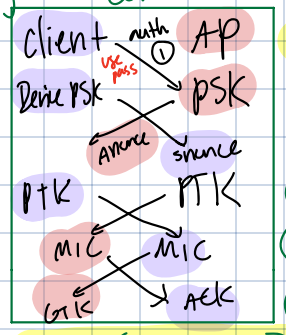
Network Adversaries: ① On-path: read ② Off: no R/W ③ In: R/W
☆ MAC addy switches b/c recipient on a diff network

MAC (2) ID (e.g. .to.old machine on LAN)
IP (global ID) ↔ port (app)

Wired Local Networks: ARP → translate layer 3 IP → MAC (2)
① "What is MAC of IP X" to LAN ② Bob → A "My IP x & MAC y" ③ cache
☆ IF Bob outside LAN, router replies w/ its MAC in ②
☆ Any received ARP replies are cached even w/o ①

ARP Attacks: ARP spoofing: ☆ cannot verify ARP
Fake map: Bob IP ↔ Malley's MAC, race
↳ Defense: Switches (MAC cache; min broadcast), VLAN isolation ☆ Must be in the same LAN + arp watch

Wireless Local Net: WPA2 encrypted msgs w/ keys, recv password
join: connect to network AP (Access pt) & announce SSID + password



Pre-shared key (PSK): for all users, PBKDF2-SHA1 (SSID, pass) ← name of network
① Auth request; ANonce → SNonce (no enc)
② Indep. derive PTK = (nonces, PSK, MAC addys)
③ Exchange MICs (MAC integrity + nonces using PTK etc)
④ AP Enc (GTK) → client
⑤ Client sends ACK for getting GTK

GTK (group temp): broadcast to entire network **PTK:** USER client ↔ AP
4-way ① Auth ② ANonce from AP ③ SNonce + MIC from client ④ MIC + GTK ⑤ client ACK

WPA Attacks: ☆ only GTK encrypted
☆ on-path: learn nonce + MAC + PSK (if on network w/ password) ⇒ derive PTK ⇒ dec (msgs) & eve on comms & enc/inject msgs
☆ Not on network → brute pass → PSK → PTK, check MICs
☆ Rogue AP (offer nonce → 4 msg)
☆ offline-brute Record nonces → gets pass exposed
No forward secrecy ☹

Defense: WPA2-Enterprise: use auth server to get key; Accept cert of server → user/pass
→ one time key (PMK) instead of PSK ☆ rogue AP has to auth to server ☆ hard to brute ☆ forward secrecy since discarded

Dynamic Host Config Protocol (DHCP): layer 2, 3, set up config 1st join
Need: to get IP address, IP of DNS, IP of router / gateway
① Client discover (request config) ② Server offer (DHCP lease) ③ Client request (which config chosen) ④ Server ack (confirms its chosen)

NAT (Net addy trans): multi comps. on LAN share IP, send w/ one
☆ Needs same LAN ☆ No way of verifying DHCP response → claim self DNS server as
☆ Issues: seen by all on LAN, No trust anchor

DHCP Attacks: forged connection
Attacker offers own IP as gateway → MITM
OR offer DNS server addy
↳ Defense: Higher layers

Border Gateway (BGP): layer 3, send msg globally, connect LANs

Subnets: group of addresses w/ common prefix (ex) 128.32/16 (32-16 subnet)

① In LAN: dest in same subnet → ARP to get MAC ② Not in LAN: send to gateway interface forwarded

AS (Autonomous System) + **ASN** (unique ID): 1+ LANs managed

AS receives packet → forward if dest inside → otherwise forward to other AS

BGP: AS advertise which networks its responsible for, path built

BGP Attacks:

Malicious ASs / BGP hijack: claim responsibility

IP spoofing: spoof src IP (false/wrong origin)

Transport (TCP): reliable*, in order, no confidentiality/integrity

*Byte stream

★ 32 bit sequence ★ 32 bit ACK ★ Seq num: detect err

16 bits	16 bits
Src port	Dest prt
Seq num	
Ack num	
Flags (+handshake)	Checksum

Seq	ACK
② A →	SYN
B ←	A+1 SYN ACK
A+1 →	B+1 ACK

Seq	ACK
X+1 →	Y+1 →
Y+1 ←	X+1+A ←
X+1+A →	Y+1+A →
Y+1+B ←	X+1+A+C ←

① Agree on 2 ISNs A & B (different & random per connection)

② client send SYN (A) → Sever responds SYN-ACK → client ACK

Data send: Byte i ⇒ seq num = X+i, A = len of data, B = len of data

Retransmission: ① Dropped packet → no ACK returned → send till ACK ② ACK not received → resend → ignore duplicate data

TCP flags: FIN (no longer send, but receive) RST (end everything)

TCP Attacks: Data injection: need to know server seq. num, race + if off 2^{32} RST inject spoof RST packet to end

(ex) client: X+1, Y+1 Attack Y+1, evil data, ignore Y+1 after...

TCP spoofing: appear to come from dif src IP + spoof seq num race

(ex) Sq = X Seq Y ACK X+1 Seq X+1 ACK Y+1, evil data RST

best effort, no guaranteed, add ports

★ faster, no 3-way handshake

Defenses: TLS, random Seq numbers

★ raw IP = no reliability

UDP (user datagram)

16 bits	16 bits
Src port	dest port
length	checksum

TLS (security): end-to-end enc, integrity, authenticity, conf.

★ No anonymity/availability ★ Headers not encrypted

① K_B ② server cert + pk_{CA}^{-1} ④ sym keys

K_B → ← K_S

← cert $(ID, PK)_{CA^{-1}}$

RSA → $\{PS\}_{K_{server}}$

DHE ↙ $\{g, p, g^a \bmod p\}_{K^{-1}_{server}}$

DHE ↘ $g^b \bmod p$

DHE $PS = g^{ab} \bmod p$

Both do: $K_B + K_S + PS = C_B, C_S, I_B, I_S$

→ MAC (I_B, dialog)

← MAC (I_S, dialog)

↘ $\{M, MAC(I_B, M)\}_{C_B}$

↙ $\{M, MAC(I_S, M)\}_{C_S}$

③ ★ MAC then encrypt

Security: legit server (cert + priv key w/ RSA/DHE)

Enc messages (PS hidden, sym key derivation)

Defend replay: (both RSA & DHE) diff K_B/K_S → sym key deriv.

★ Add record numbers to prevent old msg replay

Forward secrecy: NO for RSA, YES DHE

★ RSA uses K_B, K_S, PS → Rec w/ priv key → exposed PS

★ DHE: PS deleted after session

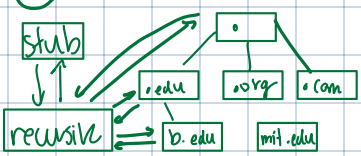
In practice: cheap, CA manages pub key + certs

★ In-secure against on path & MITM, spoof

Domain Name system (DNS): human readable ⟺ IP addy

Name server: reply to DNS req., responsible for zone of domains

(ex) name: "a.edu-servers.net" at 192.5.6.30, zone is ".edu" — authority in sect.



For eecs.berlc.edu

16 bit MSG format	16 bit
ID	Flags
# Questions	# Answers RRs
# Auth RRs	# Additional RRs
Questions	
Answers	
Authority	
Additional info	

★ Use UDP

A: Name ⟺ IPv4

NS: zone & domain of child

TTL: how long rec and can be cached (secnds) ★caching

Glue records: acq additional, non-auth records for domain

Attacks: **Cache poison:** A record: mal IP to legit domain

↳ Defense: **Bailiwick checking:** NS only provide record in its zone

(ex) berkeley.edu NS only provide for domains under berlc.edu

↳ **On-path:** insecure, plaintext; mal records + correct ID, race

↳ **Off-path:** guess ID $1/2^{16}$, Kaminsky, ↳ cache → wait TTL

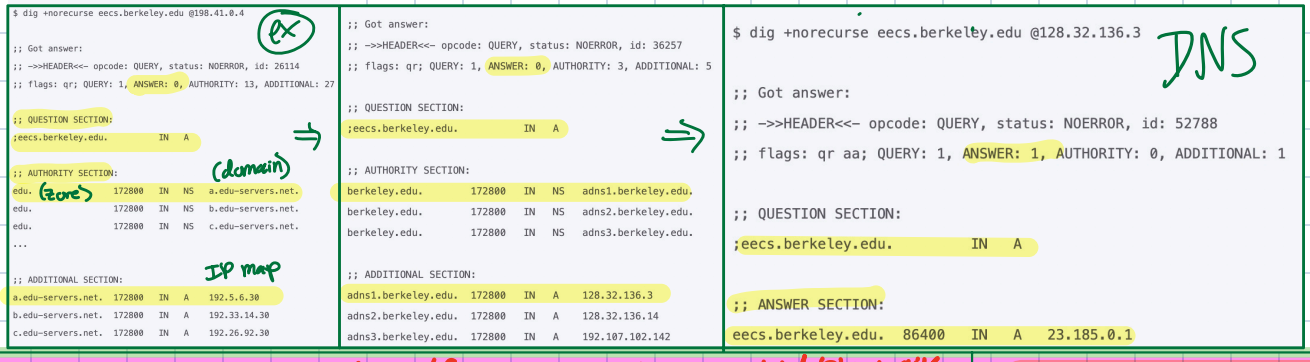
Kaminsky attack: ★need mult rd. query non-exist domains (NXDOMAIN)

(ex) Question: fake161.berlc.edu Add: berlc.edu 6.6.6.6 } race, cache

Force queries to NX until mal answers first (ex) <img src = "http://fake001.berlc.edu..">

↳ Defense: **UDP src port randomization:** 1/16, off path hard + src port $1/2^{32}$

↳ Defense: **glue validation:** don't cache glue records as part of DNS lookups



valid A: RRSIG ↓ PK ↓ sig before

DNSSEC: ★server: provide self PK & sign hash of child PK, + glue hash(3) integrity + authenticity on DNS; trust (anchor) root servers

RRSET (group of DNS records w/ same name & type)

RRSIG (encode sign on records) **DNSKEY** (encode PK) **DS** (encode child key) (delegated signer)

Delegate trust (parent sign child key): DS on child, RRSIG sign on DS

(ex) IP of eecs.berlc.edu to .edu → Reply: NS, A, DS (hash PK), RRSIG DS (sign DS), DNSKEY (.edu PK)

(ex) Prove .com ⇒ Send DS, RRSIG, DNSKEY root + DNSKEY .com

NSEC (sign Non-existent)

2 adjacent → Domain enumeration ("retrieve all")

NSEC3 (hashed)

(ex) btw c6125 ⟷ d8lolc enumeration still possible

Defense: online sign gen w/ priv key

★Attack: get server priv key

☆ Algo complexity

DoS: overload bottleneck (area w/ least resources)

Amplified DoS ex ① send spoof IP udp to DNS recursor ② Large req → send to spoof ("src" 1 → 2)
SYN Flood: send large # of SYN w/ no ACK, waste mem
↳ works on DNS over TCP! **Defense:** SYN cookies (client stores state) filter, over prov.
DDoS: botnet (collection computers controlled by 1 attacker): Def: analyze & drop packets

Application DoS: target app
↳ **Defense:** Identification, Isolation, Quotas, Proof of work, over-prov.
Network-Level: target bandwidth
↳ Defense: packet filter, over p.

Firewalls: block access to network services from running on internal machines
Outbound (traffic exits) **Inbound** (traffic enters)
Default-allow (deny list) vs. **Default-deny** (allow list) ← Best choice / fail safe
Stateless packet filter: no history, make decision using info of packet itself
Stateful: keep state, tracks all in/out connections, forward/drop connections [client ⇄TCP Fire; TCP server]
Subvert packet filters: ex split "root" word ex ping w/ incr. TTLs
Proxy firewall (MITM w/ TCP ⇐⇒) **Application proxy:** application layer
VPN: set of protocols that allows direct access to internal net w/ external connection (enc tunnel)
☆ Attacks: DoS to overwhelm wall, XSS to JS on external server, internal server loads
☆ pro: mitigate on end hosts
Evasion: special inputs to avoid detection

Intrusion Detection Internet > Border Router ← server computer ☆ **NIDS on Border**

NIDS (network) — Pro: cheap, easy to scale, simple; Con: inconsistent, evasion, no-enc det.	**HIDS** (Host) — Pro: less inconsist, encrypted, performance issues; Con: expensive, evasion attacks	**Logging** analyzes logs — Pro: access to end host data, cheap; Con: evasion, not real time

False Negative: attack happened, not det **False positive:** no attack, attack reported (FPR)
☆ **Parallel composition:** increase FPosR, decr other ☆ **Series comp:** incr FNegR ☆ **Path attack** / none / 1 / can → / round

Signature-Based	Anomaly-Based	Specification-based	Behavioral
matches known attack, black list	model of normal activity, flag deviation	Specify normal activity, flag dev. (white list)	Look for evidence of compromise
pro: known signatures	Pro: new attacks caught	pro: catch new, FP lower	Pro: catch new, low FP, cheap
con: can't catch new attacks/variants	con: train model, large FN, FP	con: time consuming to write specifications	con: detected after started, use diff behavior to execute

Vulnerability scanning: probe code for wide range con: lots of work/dangerous
Honey pot: sacrificial bait system con: hard to trick attackers/needs convince
Intrusion prevention (IPS): block attacks while detecting (IDS) con: hard to modify, FP impact users
↳ Attacks: limited resources (DoS), exhaust IDS mem (SYN flood), algo complex, code inj.

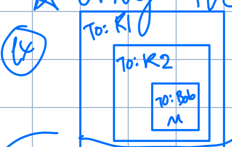
Malware: attacker code runs on victim computers
Virus: needs user action to propagate
↳ Detection: Signature based ☆ **polymorphic** (insert enc code, obfuscation)
Metamorphic semantically diff, use behavioral det.
Worm: dn need user action, alters code already running, spread exponentially

Anonymity: conceal identity **Proxy:** 3rd party relay traffic
TOR: network that uses multiple relays (proxies) for anon comms ☆ n vs. n-1 malicious

Tor Browser → relay → relay → relay → relay → website
(circuit (3+ relays); TLS connection)

☆ Security ☆ performance ☆ Anonymity against adversaries

☆ only the exit node (MITM) dec final mssg for recipient **relay ≠ server**



Timing attack: global network; link when send vs. recieve
☆ Tor only defends against local adversaries w/ partial view (easy to form w/ own)
Collusion: mul. nodes collab; more we use, more confident they don't collude
↳ **Defense:** **Guard nodes** (high reputation, exist a while)
↳ Defense: **Tor Bridges:** entry nodes not available on public lists **Pluggable transports:** change of appearance of entry

TOR onion Services: web only available via TOR (ex) Darknet
TOR in practice: ☆ Free ☆ exit nodes MITM ☆ performance ☆ tradeoffs
☆ Arms race btw Tor & censors

Bitcoin: digital cryptocurrency
☆ Hash algorithm makes it difficult to append
☆ signatures prevent users from spending other user's coin

Certificates ☆ Sign public key w/ authority $\{M\}_{PK}$ $\{M\}_{SK^{-1}}$
Trust on First Use: 1st time you comm., trust PK & warn if changes
$\{$ Bobs Pk is PK_B $\}_{SK^{-1}}$
☆ **Trusted directory (TD):** fetch anyone's public key ⇒ signed by SK_{TD} (Not Scalable)
☆ **Hierarchial trust:** root/authority (root CA) V.S. intermediate CAs (sign for)
☆ **Revocation:** contain expiration date vs. CAs sign list of revoked certs

Passwords ☆ Need to verify pass w/o storing info you could recover with
Rainbow tables: algo for computing hashes that makes brute easier
Salt: random public value to make attacks harder: $H(pass \| salt)$
↳ should be long + random, not secret (try pass & user) $O(MN)$ to guess vs. $O(M+N)$ (hash & lookup)
Slower hashes: add large constant factor, make it inconvenient
Offline: computation themselves (defend: salt/slow)
Online: interacts w/ service (defend: timeout/rate limit)

Traffic Analyzing who is talking to whom & when
Side channels: info about Plain revealed bc of implement of scheme
☆ good sys should desc how public param generated: Nothing-Sleeve

gets(char *str) read from stdin, appened null byte
fgets(char *str, int n, FILE *stream) n includes final null char
strcpy(char *dest, (const) char *source) reads till null byte
strncpy(char *dest, const char * src, size_t n) stops at size
memcpy(void * dest, const void *src, size_t n) n chars to dest
read(int fd, void * buf, size_t cnt) cnt == num bytes
fread(void * ptr, size_t size, size_t nmemb, FILE *stream) (size * num elems)
scanf(const char * format, ...)
(ex) fgets(&buf) → write at buf aka the address in it
fgets(buf) → write at arr, deref value in buff

ret2ret return existing ptr that points into shellcode (stack shift, nop, nop)
(ex) overwrite rip w/ ret instr, above put ptr to buf / \x00
aim ↳ shift, nop

ret2esp jmp esp (overwrite rip w/ code already) put shellcode above rip, esp goes to right after eip/rip

ret2pop gadget to chain returns

Security Principles

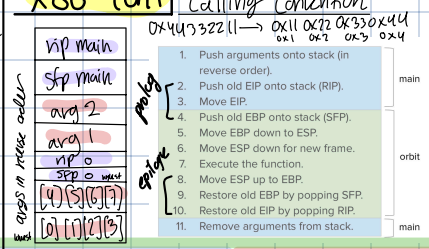
① Know your threat model
② Human factors
③ Security is economics
④ Detect if you can't prevent
⑤ Defense in depth: layer mul. types
⑥ Least privilege: grant for correct functioning
⑦ Separation of resp: require mul. ppl to get access
⑧ Ensure complete mediation: unbypassable, monitor access
⑨ Shannon's maxim: the every knows the system
⑩ Fail-safe defaults: fail in safe state, balance security usability
⑪ Design in security from start, avoid patching

x86

Stack: function frames
Heap: malloc/dynamic
Static: const strings, buffers in static mem
Code: x86 instr, where rip & eip point to!

x86 cont

Calling Convention



Prologue

```
push %ebp  //save sfp
mov %esp %ebp  //new frame
sub $X %esp  //make esp down
```

args in reverse, push eip, save to sfp, bump ebp down to esp & bump esp down

Epilogue

```
add $X %esp  //make esp up by X
pop %ebp  //pop sfp, store in ebp
ret  //pop rip, go up
```

Move esp to ebp, pop ebp, pop eip & make local args

```
int main(void) {
  foo(1,2);
}
void foo(int a, int b) {
  int bar[4];
}
```

```
main:
push $2
push $1
call foo
(finishes, remove args from stack)
add $8, %esp
```

```
foo:
push %ebp  - push old ebp (sfp on stack)
mov %esp, ebp  - move ebp down to esp
sub $16, %esp  - move esp down
move %ebp, %esp  - move esp
pop %ebp  - restore old ebp (sfp), +4 esp
pop %eip  - restore old eip (rip) +4 esp
```

☆ pop always bumps esp up, stuff value from esp into register
ex: eax
execute func...
☆ pop %edx = pop %esp

```
native process 2075 In: orbit                    L6    PC: 0xbffc4ab
(gdb) info frame
Stack level 0, frame at 0xbffff860:
 eip = 0xb7ffc4ab in orbit (orbit.c:6); saved eip = 0xb7ffc4d3
 called by frame at 0xbffff880
 source language c.
 Arglist at 0xbffff858, args:
 Locals at 0xbffff858, Previous frame's sp is 0xbffff860
 Saved registers: sfp & rip addresses
  ebp at 0xbffff858, eip at 0xbffff85c
```

next instr to be executed
value of rip of current frame, jump back when done

p [VAR]	// print the value of a variable
p &[VAR]	// print the address of a variable
x/nwx [VAR]	// print "n" words of memory starting at VAR in hex.
x/nwx [VAR]	// print "n" words of memory starting at VAR in hex.
info registers	// display current ESP/EBP/EIP
info frame	// display location of SFP/RIP
refresh	// re-render the screen

"x/16ux buf" 16 words of mem starting at buf
0xffff1430: 0x.... (16 bytes)
0xffff1440: 0x.... (higher addr)

```
0xb7ffc4a5 <orbit>    push   %ebp
0xb7ffc4a6 <orbit+1>  mov    %esp,%ebp
0xb7ffc4a8 <orbit+3>  sub    $0x18,%esp
```

address it resides at ☆ ret like pop %eip

Mem Safety Vulnerabilities

Name	Desc
Buf overflow	gets(), read() vs. fread() fgets; write to buf & above
Stack Smash	overflow to overwrite rip & point to shellcode
Int conversion	size_t (unsigned), uint8_t (0→255) vs. int (signed, -128, →127) ex: -1 → 1111 1111 -128 1000 0000 (128 0111 1111)
off-by-one	manipulate 1 byte of sfp & null byte, addy is fake sfp, +4 above is fake rip, fake buf
format string	%s treat arg as addy, print str at addy till null %n treat arg as addy, write # chars (4 bytes) printed to addy %c treat arg as val & print as char %x print variable as hex %[b]u print [b] bytes unsigned int starting at arg %hn same but write 2 bytes "x is %d, y is %d" / "%d, %d is %d" (x, y / y, x)
ret2libc (see later for more)	'A'*24 + [address of system] + 'B'*4 + [address of "rm -rf /"] (overwrite rip, next rip → + "rm -rf /" expected)
ROP: chain of ret addys starting at rip	ex want: mov $1, %eax ; xor %eax, %ebx vs. <bar+25> → <foo+10> → get addr sfp main / rip vulnerable → addr foo+10 / addr bar+25

Defense

Defense	Desc	Subversion
Non-exec (WX, DEP, NX)	writeable or executable not both, stop stack smash	ret2libc: overwrite rip to jmp to funcs in standard lib ROP: construct custom shellcode using pieces of code already in mem
Stack canary	same for all funcs within a run	leak, bypass, brute-force
PAC	64 bits ~22 bit PAC, each addy has its own	leak, brute, pointer reuse
ASLR	segment in diff spot	leak addr of ptr / RIP

ex: printf("%x") vuln vulnerable(char *dest){ ... buf[20] ... vulnerable(buf); } print(dest);

ex output: bfff0408 → grab sfp & print value (pointer → val is addy) which is the main sfp address

Cryptography

	Symm	Assym
Conf	• One time pads eg. • Block ciphers CBC • Stream ciphers	• RSA enc • El Gamal enc
Int/ Auth	• MACs e.g. HMACs AES-CBC-MAC	• Digital signs. (e.g. RSA)

	E tricks A into E() M of choosing	E tricks B into DC) M of E choosing
C-only	N	N
chosen P	Y	N
chosen C	N	Y
chosen PC	Y	Y

SYM-Key

KeyGen() → K, Enc(K,M), Dec(K,C)

IND-CPA: chosen P: M0 & M1 → send C? back → E can send M & get C back → ½ + guess?

One-time pads (OTP) $X \oplus 0 = X$ $X \oplus X = 0$ $X \oplus y = y \oplus X$ $(X \oplus y) \oplus z = X \oplus (y \oplus z)$ $(X \oplus y) \oplus X = y$

is KeyGen() → n-bit key (n = len) Enc(K,M) = K ⊕ M Dec(K,C) = K ⊕ C = M

★ Needs key gen + distribution ★ cannot reuse key ★ expensive

Block Ciphers: encrypts fixed size block bits $\{0,1\}^{keyx} \{0,1\}^n \to \{0,1\}^n$

★ correct ★ efficient ★ secure ★ Bijective ★ deterministic → not IND-CPA secure

3) CTR Mode (Counter): counter initialized to IV, incremented & encrypted, nonce = IV (malleable; secure w/ predictable, unique IV, no reuse)
Encryption: $C_i = E_K(IV+i) \oplus M_i$ (can parallelize)
Decryption: $M_i = E_K(IV+i) \oplus C_i$ (parallelizable)

4) CFB Mode (Ciphertext Feedback):
Encryption: $C_0 = IV$, $C_i = E_K(C_{i-1}) \oplus P_i$
Decryption: $P_i = E_K(C_{i-1}) \oplus C_i$ (no padding needed)

1) ECB Mode (Electronic Code Book): break into n-bit blocks $M_1 \dots M_i$
Encryption: $C_i = E_K(M_i)$, concate blocks
Decryption: $M_i = D_K(C_i)$ not secure
↳ flaw: leaks info, redundancy shows

2) CBC Mode (Cipher Block Chaining): need to reveal IV (secure)
choose random initial vector (IV), use prev
Encryption: $C_0 = IV$, $C_i = E_K(C_{i-1} \oplus M_i)$ Not Braille
Decryption: $P_i = D_K(C_i) \oplus C_{i-1}$ parallelizable

5) OFB Mode (Output Feedback):
Encryption: $Z_0 = IV$, $Z_i = E_K(Z_{i-1})$, $C_i = M_i \oplus Z_i$
Decryption: $P_i = C_i \oplus Z_i$

Padding: Need to make sure message is multiple of block cipher, CTR, CFB doesn't need
PKCS #7: pad message by num of padding bytes
- Don't reuse IV, makes it IND-CPA insecure
- CTR fails catastrophically, CBC is contained

Secure: CBC, CTR Not: CBC No pad: CFB

★ No integrity / authenticity

Hashes $\{0,1\}^* \to \{0,1\}^n$ ★ correct/deterministic ★ efficient ★ secure $x' \neq x$ hard to

One-way (infeasible to find x s.t. H(x) = y) Pre-image/collision resist find H(x) = H(x')

Avalanche effect (H(M) ≠ H(M 1 bit change)) - random oracle assumption

Birthday attack (on coll resist): find collision on n bit output takes $2^{(n/2)}$ tries on avg

Standard: SHA-3, SHA2 (len ext), SHA1 & MD5 (broken), SHA256 preferable

Len-extension attack: given H(x) & len of x → make H(x || m) for any m of attackers choosing

MACs (K, M) → Tag ★ deterministic / not confidential ★ Integrity / authority

Enc-then-MAC (MAC k2 (Enc k1 (M)) MAC-then-enc Enc k1 (M || MAC k2 (M))

HMAC (K, M): pad K if needed w/ 0s, or hash to shorten = H((K' ⊕ opad) || H((K' ⊕ ipad) || M))

Authenticated enc w/ Additional data (AEAD): conf. + integrity w/ extra data e.g. rem addy

PRNGs seed(randomness) → reseed(random) vs. generate(n) → n pseudo rand bits

★ correct/deterministic ★ efficient ★ security (indist. random) ★ rollback resist (cannot learn prev. states)

Stream Ciphers (secure) symmetric enc, pseudo-rand bits as key to one-time pad seed(K||V) and G(generate(n)) ⊕ P = C

Diffie Hellman g, p, g^a mod p → hard to find a g^{ba} mod p ★ ephemeral / discard a, b

★ Not secure w/ MITM ★ NO authentication ★ Need to be online active

Pub-key Enc ★ much slower (priv) than symm-key (rand) KeyGen() → PK, SK, Enc(PK, M) → C Dec(SK, C) → M

ElGamal: KeyGen() → $B = g^b$ mod p, Enc(B, M): $R = g^r$ mod p → $M \times B^r$ mod p = C_2, $C_1 = R$

Dec(b, C_1, C_2): $C_2 \times C_1^{-b} = M \times B^r \times R^{-b} = M \times g^{br} \times g^{-br} = M$ mod p ★ Not secure (is malleable)

RSA: KeyGen() p & q (2 large primes), N = pq ⇒ choose e rel. prime 2 < e < (p-1)(q-1) → $d = e^{-1} \bmod (p-1)(q-1)$

★ Public: N & e ★ private key: d ★ correct $M^{ed} = M \bmod pq$ ★ Not IND secure

↳ OAEP: variation of RSA introduces randomness → ★ avoid determinism

Hybrid encryption: Enc( ) symmetric enc, then enc(k) assym encryption ★ most use this

Digital Signatures KeyGen() → PK, SK Sign(SK, M) → sig Verify(PK, M, sig) → {0,1} (G on sigs don't need inputs)

★ correctness ★ efficiency ★ security (EU-CPA)

RSA Sign KeyGen() → N & e, d → sign(d, m) = $H(M)^d \bmod N$ Verify(e, N, M, sig) $H(M) = sig^e \bmod N$

unforgeable w/ secure hash