Calling convention

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

* strings in C terminated with \00

SECURITY PRINCIPLES

1. Detect if you can't prevent
2. Defense in depth (multiple types of defense layered tgt) — or key in diff loc
3. Least privilege (give only access it needs to do its job)
4. Separation of responsibility (not 1 party has complete power)
5. Ensure complete mediation (check access to every object)
6. Shannon's maxim: attacker knows system they're attacking (don't rely on obscurity)
7. Kerckhoffs's principle: secure even when attacker knows all internal details
8. Fail-safe defaults: if mechanisms fail → lead to secure behavior
9. Design security from the start

increase →

0xFFFFFFFF
...
0x00000000

increase ↑

* TCB = trusted computing base: must operate correctly for security
  ↳ unbypassable, tamper-resistant, verifiable

* TOCTTOU = time of check to time of use: race condns between check & use

Mem safety vulnerabilities

1. **Buffer overflow**: no bounds checking → leads to out of bounds memory access
   - ↳ unsafe fn like $\boxed{\text{gets() or read()}}$
   - ↳ user can provide arbitrary # of bytes

EIP = instr pointer (curr instr addy)

EBP = addy @ top of curr stack frame

ESP = addy at bottom of curr stack frame

2. **Stack smashing**: use long input → overwrite SFP → overwrite RIP to point @shellcode
3. **Integer conversion attack**: signed vs unsigned, C will implicitly cast
   - ↳ $\boxed{\text{memcpy}}$ takes in size_t → unsigned but pass in int8_t
   - ↳ 0xFFFFFFFF = −1 w/ 2's complement

$\boxed{\text{push}}$: decrement esp
* store value in newly allocated space

$\boxed{\text{pop}}$: increment esp
- copies popped val into a register

4. **Off by one attack**: overflow 1 byte after buffer → redirect SFP and run shellcode
   - ↳ using <= instead of <
   - ↳ start loop at i=0 instead of i=1
   - ↳ needs TWO function returns: one to move SFP into buffer, one to actually execute instr @ arbitrary location

   ** put it as \xff NOT −1

   ** want to make fake RIP contain shellcode

fread(*void ptr, size_t size, size_t count, FILE * stream) — unsigned

gets() adds null byte after last byte read

printf reads string until null byte (\x0)

5. **Format string attack**: improper usage of $\boxed{\text{printf}}$ to read/write arbitrary locations
   - ↳ looks for '%' format string modifier 4 bytes above RIP of printf
   - ↳ args start 8 bytes above RIP of printf

printf("x has the value %d, y has the value %d, z has the value %d \n", x, y, z)

z
y
x
&("x has value %d, y has value %d, z has value %d \n")
RIP printf
SFP printf

* pass in 3 %d modifiers, only have 2 args
  ↳ will print out whatever is 16 bytes above RIP

Format string modifiers

- %x reads hexadecimal
- %d reads decimal
- %c reads char (reads 4, prints 1 byte)
- %s deref pointer → print string
- %n write # of chars printed to that addy
- %hn write 2 bytes to address
- %<>u where <> is a # print <># starting at arg → unsigned int
  * can combine w/ %n to do %123u%n — write this to address

6. **ret2ret attack**: utilizes special instr ret & presence of pointer to redirect execution to shellcode → rather than fixed shellcode address

* useful for when ASLR is enabled → jump to stack location dynamically

DEFENSES

1. **Stack canaries**: insert random 4-byte value below SFP and above local vars
   - ↳ limitations: no protection against heap overflows, local var can be overwritten
   - ↳ counter: guess the canary (32 bit arch w/ 24 bits randomness)
   - ↳ counter: leak the canary: use vulnerability to read canary (format str vulnerability)
2. **Address space layout randomization (ASLR)**: randomize start of each segment of memory st absolute addresses of each item = unique
   - ↳ relative positioning still stays the same
   - ↳ can guess or leak the address (32 bit system: 16 bits of entropy)
3. **Non-executable pages**: mark stack as non-executable to prevent stack smashing
   - ↳ counter: return into libc
   - ↳ counter: return oriented programming (chain of return addy: point to gadget → end in ret)
   - ↳ local data manipulation still possible

Tips

1) Cryptographic hash not 1 to 1 has collisions
2) Stack canary, ASLR, non-executable pages doesn't prevent all buffer overflows
3) Deterministic chosen IVs make block cipher not secure.
4) CTR secure w/ predictable nonce, no reuse counter
5) Pad CBC
6) Digital Signature: verifying key public, signing key private

THREAT MODELS

① ciphertext only attack : intercept encrypted msg → want to recover plaintext

② known plaintext attack : intercept encrypted msg WITH partial knowledge of plaintext
↳ find plaintext

③ replay attack : capture encrypted msg and re-send to recipient repeatedly

④ chosen plaintext attack : trick A to encrypt msgs of choice → observe ciphertexts
↳ use info for msg recovery for new msgs

⑤ chosen ciphertext attack : trick B into decrypting certain ciphertexts
↳ use this output to decrypt other ciphertext

⑥ chosen plaintext/ciphertext attack : encrypt / decrypt msgs of choice

XOR: $0 \oplus 0 = 0$, $0 \oplus 1 = 1$, $1 \oplus 0 = 1$, $1 \oplus 1 = 0$

$(x \oplus y) \oplus x = y$ — cancels out x

PASSWORDS

① online guessing (dict w/ common passwords)
↳ rate limit, CAPTCHA

② eavesdropping
↳ SSL

③ client side malware (key strokes)
↳ 2 factor auth

④ server comp
↳ salting for randomness
↳ slow hash

IND-CPA game

① Adversary chooses M_0 & M_1 and sends to Alice
② Alice encrypts either M_0 or M_1 & sends back to Eve
③ Adversary does chosen plaintext attacks
④ Guess whether encrypted msg is M_0 or M_1

* $len(M_0) = len(M_1)$
* practical # of encryption requests
* win w/ non negligible advantage

One time pad :

KeyGen: pick random key K
Encryption: $C = M \oplus K$
Decryption: $M = C \oplus K$

* key cannot be reused

AES = advanced encryption standard

block length n = 128 bits
key k = 128 bits

encrypt : $Enc_K(M) \to C$
$G : \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$ * deterministic
decrypt : $D_K(E_K(M)) = M$
invertible encryption

→ computationally indistinguishable from random permutation

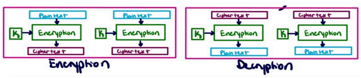
BLOCK CIPHERS

Modes of operation

ECB mode : electronic codebook

encrypt: $C_i = E_K(M_i)$
decrypt: $M_i = D_K(C_i)$

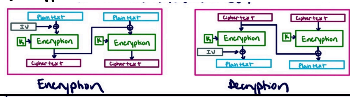
NOT parallelizable
* leaks info, redundancy



Encryption Decryption

SECURE!! CBC mode : cipher block chaining

encrypt: $C_0 = IV$, $C_i = E_K(P_i \oplus C_{i-1})$ NOT parallel
decrypt: $P_i = D_K(C_i) \oplus C_{i-1}$ parallel

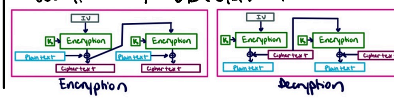


Encryption Decryption

SECURE!! CFB mode : cipher feedback

* no padding needed

encrypt: $C_0 = IV$, $C_i = E_K(C_{i-1}) \oplus P_i$
decrypt: $P_i = E_K(C_{i-1}) \oplus C_i$

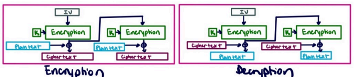


Encryption Decryption

OFB mode output feedback

encrypt: $Z_0 = IV$, $Z_i = E_K(Z_{i-1})$, $C_i = M_i \oplus Z_i$
decrypt: $P_i = C_i \oplus Z_i$

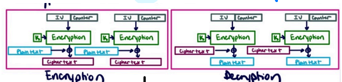
* easy to tamper



Encryption Decryption

CTR mode : counter

encrypt: $C_i = E_K(IV + i) \oplus M_i$ parallel
decrypt: $M_i = E_K(IV + i) \oplus C_i$ parallel



Encryption Decryption

Padding : add padding to input until multiple of 128 bits

$\boxed{\text{PKCS\#7 Padding}}$ = pad msg by # of padding bytes used
↳ only needed for CBC

HASHES : generates deterministic fingerprint of document

** same input = same hash value

PROPERTIES:
① one-way : (given output, hard to find input)
② Given input, infeasible to find any input to hash to same value
③ collision resistant : infeasible to find x' st $x' \neq x$ but $H(x) = H(x')$
* 2 inputs that hash to same value

SHA-1 → not secure bc of collisions
SHA-2 → length extension attack (create $H(K \| M \| X)$ given $H(K \| M)$)
SHA-3 → secure — SHA3-256, SHA3-384, SHA3-512
** relate hash function output w/ symmetric encryption algorithm key length

MACs = message authentication codes → gurantee integrity/authenticity

$T = F(K, M)$ = 128 bit tag on msg M
↓
128 bit key K

A sends <M,T>
→ B checks if $F(K,M)$ matches T

** deterministic for correctness, not confidential

$NMAC(K_1, K_2, M) = H(K_1 \| H(K_2, M))$ * $K_1 \neq K_2$
$HMAC(M, K) = H((K' \oplus opad) \| H(K' \oplus ipad \| M))$
* if you change a single bit → unpredictable

Pseudorandom Number Generators

- entropy = measure of uncertainty → uniform distribution - highest entropy
↳ TRUE randomness = expensive
- PRNG : take small amt of truly random bits → output long sequence of pseudorandom bits
** deterministic, computationally indistinguishable
① seed (entropy) → take truly random entropy & initialize internal state
② reseed (entropy) → take in additional truly random entropy → update PRNG
** rollback resistance : attacker cannot deduce anything abt prev generated bit

stream-cipher : encrypt + decrypt msgs as they come 1 bit @ a time
$Enc(K, M) = \langle IV, PRNG(K, IV) \oplus M \rangle$, $Dec(K, IV, C_2) = PRNG(K, IV) \oplus C_2$ ← ciphertext
generates same pseudorandom bits

Authenticated Encryption = confidentiality + integrity

① encrypt then MAC = send $\langle Enc_{K_1}(M), MAC_{K_2}(Enc_{K_1}(M)) \rangle$
↳ may leak ciphertext through MAC → ok!

② MAC then encrypt = send $Enc_{K_1}(M \| MAC_{K_2}(M))$
↳ susceptible to side-channel attacks

(!) use diff keys

Public key Encryption

Trapdoor one-way : one-way w/ special backdoor to allow special compute

RSA : B encrypts w/ A's publickey, A decrypts w/o wn priv key
gen random symmetric key

* add randomness through padding (OAEP)

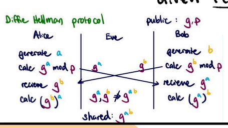
El Gamal : modified Diffie Hellman

Encryption: $E_B(m) = (g^r \bmod p, m \times B^r \bmod p)$
Decryption: $D_b(R, S) = R^{-b} \times S \bmod p$

$g: 1 < g < p-1$
$B: g^b \bmod p$
b: Bob's private key
p: large prime
↳ drawback: does not preserve integrity

Diffie-Hellman

* DISCRETE LOG PROB: Given $f(x) = g^x \bmod p$ → no efficient algorithm to solve for x



** Difficult to find $S = g^{ab} \bmod p$ from A, B, g, P

Man in the middle attack ** tamper with messages
① Alice sends $g^a \bmod p$ → replaced with $g^m \bmod p$
② Bob receives $g^m \bmod p$ → calculates $g^{mb} \bmod p$
③ Bob sends $g^b \bmod p$ → replaced with $g^m \bmod p$
④ Alice receives $g^m \bmod p$ → computes $g^{ma} \bmod p$

DIGITAL SIGNATURES : only B can sign, everyone verify

KeyGen() → (PK, SK)
public ↑ ↖ priv

signing → sign $S = Sign(SK, M)$ = signature w/ priv key SK
verify → verify(PK, M, S) using public key PK

Ex) RSA
primes p, q $n = pq$
$Sign(M) = H(M)^d \bmod n$
$Verify(M, S) = T$ if $H(M) = S^3 \bmod n$

CERTIFICATES

$\{M^3\}_{PK}$
① encrypt w/ public key
② sign w/ priv key $\{M\}_{SK_J^{-1}}$ ↙ sign by Jerry

Alice must trust Jerry

** forming hierarchical tree

* PUBLIC CHANNEL

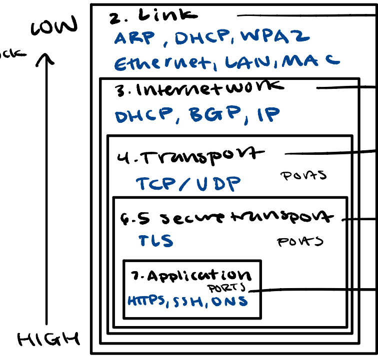
REVOCATION = expiration periods
Revocation lists

Leap of faith = trust public key the first time you communicate

Adversaries

off-path: cannot read/modify
on-path: can read but NOT modify
in-path/MITM: can read/modify/block
ALL: can send own packets
↳ spoof packets

Headers/OSI model



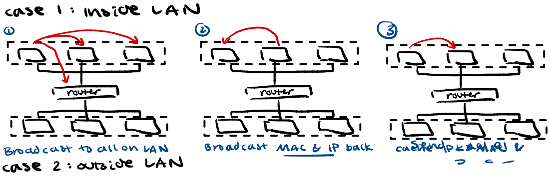
Link: connect machines in LAN
Internetwork: connect LANs. Best effort delivery
Transport: ensure reliable data delivery
Secure transport: provide end-to-end secure communication
Application: connect network to end-user apps

* high relys on low for transmission
* low relys on high for cryptography

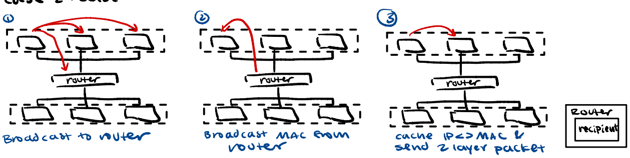
ARP: layer 2 (link) *wired local network*

Address resolution protocol
- Purpose: translate IP to MAC addresses
- vulnerability: on-path can see requests + send malicious response
- defenses: switches, arpwatch to keep track of IP<=>MAC pairs

case 1: inside LAN



case 2: outside LAN



DHCP: layer 2 (link)/layer 3 (internetwork)

Dynamic Host config protocol
- purpose: get configs when first connecting
- vulnerability: on path can see requests & send malicious responses
- defense: rely on higher layers

4 steps:
1. client discover: client broadcasts request for config
2. server offer: server offers some config settings
3. client request: client broadcasts the one it chooses
4. server acknowledge: chosen server confirms its config has been chosen

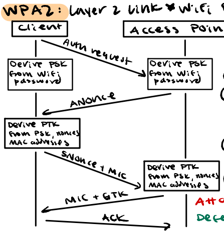
Need:
IP address for yourself
IP address of DNS server
IP address of router (gateway)
mapping of internal → remote IP addresses

NAT: network address translation: allows multiple computers on local network to share IP

Attack: attacker sends forged config MITM

Defense: rely on higher layers

WPA2: layer 2 link *wifi Protected Access - wireless local network*



PSK = preshared key = 1 password for all users = apply PBKDF2-SHA1 on SSID, password

1. AP sends ANonce
2. Client receives ANonce & derives PTK (pairwise transport keys), sends SNonce, MIC to AP
3. AP receives SNonce & can derive PTK
   ↳ sends GTK (group temporal key) encrypted, MIC
4. sends ACK for successful GTK

Attacks: on path attackers can learn encryption keys → brute force password → decrypt messages

Defenses: WPA2-enterprise: all users have unique UN, PW, random pairwise master key, encrypted channels

Malware: attacker code that runs on victim computers

Virus: requires user action to propagate ex) infect startup code of application
↳ Detection: Signature based → Polymorphic & Metamorphic Code: encrypted code of malcode or code rewriter

Worm: does not require user action, alters code already running, spreads exponentially

➤ HTTPS encrypts URL parameters

BGP (layer 3) = Border Gateway Patrol

Purpose: send msgs globally by connecting lots of local networks
vulnerability: read msgs in intermediate transit and forward to wrong place → lying AS to act responsible for network it's not
operates on trust lying AS = MITM
defense: rely on higher layers (TCP, TLS)

1. if same subnet, ARP to translate IP to MAC address
2. if different subnet, send packet to gateway → Autonomous system
   ↳ each AS advertises which network it's responsible for
   ↳ select shortest path

UDP: layer 4 (user datagram protocol) = best effort transport layer protocol

* not guaranteed order
* faster so better when speed is a concern
* checksum: NON cryptographic to detect corrupted packets

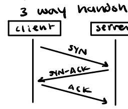
←16→	←16→
source port	Destination port
Length	checksum

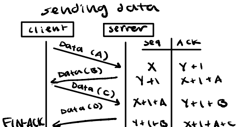
TCP: Layer 4 (transmission control protocol) reliable, in order, connected stream based

1. HANDSHAKE to establish connection to server
2. resends dropped packets & breaks up long msgs into indiv packets
- uses seq # and ACK # (SYN, ACK, FIN)
- good for client-server relationship

←16→	←16→
source port	destination port
sequence #	
Acknowledgement #	
Flags	checksum

3 way handshake / sending data





* guarantees correctness
* SLOWER

* To end: send FIN, reply w/ FIN-ACK
  To abort: send RST, no ACK needed

SEQ set to index of first byte in packet (fill in header like TB)

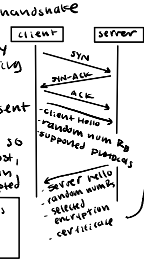
Attack: Packet injection (spoof)
1. off-path: know/guess client IP/port, server IP/port, seq num
2. on-path: can inject into TCP → RACE
3. in-path: can block msg from party & send their own

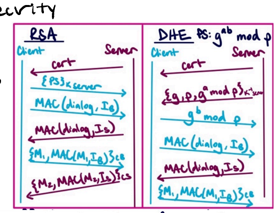
Defense: rely on higher layer TLS

TLS: layer 6.5: transport layer security

starts w/ 3 way handshake

* server needs public/private key pair (lead to spoofing)
* replay attack = recording old msgs sent
* DHE provides FORWARD SECRECY so if priv key lost, old conns can NOT be decrypted



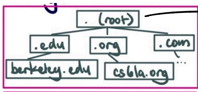


1) Starts w/ ClientHello random num R_B, supported encryption protocols
2) ServerHello, random num R_S, selected encryption, Server's certificate, server's pub key signed by certificate authority (CA)
3) Generate Premaster Secret (PS) w/ RSA or DHE
4) use PS to derive 4 shared symmetric keys, encryption C_B, C_S, integrity I_B for client, integrity I_S

* R_B, R_S defend against replay attacks
* Three security guarantees
  1. client talking w/ legit server
  2. No one has tampered w/ handshake
  3. client & server share set of symmetric keys UNIQUE to connection
* trust can be DELEGATED w/ certificates

DNS: layer 7: domain name system

Translate between IP & human-readable



→ root server which will direct to one of children (provides 1: child's zone 2: human readable domain 3: IP address)

DNS Recursive resolver: DNS lookup from internet service provider, internal cache

DNS stub resolver: sends query to recursive resolver then receives response

↳ uses UDP

Identification	Flags
#Questions	#Answers
#Authority RR	#Additional RR
Questions	
Answers	
Authority	
Additional Info	

① A Type record: domain → IP
② NS Type: zone → domain
③ RRSIG record: DNSSEC only - digital signature on records
④ DS Type: DNSSEC only: verifies authenticity of child zone

① Question section: domain queried for
② answer section: direct answer to query
③ authority section: zone & domain of next name server
④ additional section: IP address of next server

Attack: malicious name server → send to malicious IP
↳ Defense: Bailiwick checking → name server only provides records in its zone

Attack: on path attacker can read/respond w/ malicious (completely insecure)

Attack: Kaminsky attack (off path) querying for non existent domains
↳ NXDOMAIN causes nothing to be cached
↳ race cond to cache malicious response
↳ defense: UDP source port randomization → random 16 bit source port: guess ID + source port $= \frac{1}{2^{32}}$

DNSSEC = extension to DNS to provide integrity & auth to DNS msgs sent

* using trust anchor to sign public keys in DNS tree → delegate trust to others
  ↳ have signature on next name server's public key
  ↳ signing done offline to save time (more memory)
* sign next server's public key & give your own public key
* two public/priv key pairs
  ① key signing key → sign the zone signing key (KSK)
  ② zone signing key → sign everything else (ZSK)
* Give records have IP address of domain's name server
* DS and RRSIG records used to endorse trust to another name server
* A type and RRSIG record used to deliver answer + signature on answer

Denial of service (DoS)

① Application level (layer 7) → targets resources an app uses
examples: exhausting filesystem w/ write
exhausting RAM w/ malloc
exhausting processing threads

Defense → THREE PRONGS
① Identify
② Isolation
③ Quotas

* program flaws
* resource exhaustion
* attack bottlenecks

② SYN flood attacks (layer 3) (w/ spoofed address): ① send a lot of SYN ② ignore SYN/ACK ③ never send ACK

* wastes & blocks memory for legit TCP requests

Defenses: ① overprovisioning
② filter packets w/ SYN cookies
↳ don't create state until end handshake

③ DDoS = distributed DoS
leverage multiple machines & botnet

Defense: analyze incoming traffic and drop packets

Firewalls

* Blacklist (default allow) = allow everything except those listed (fails open = security flaw)
* Whitelist (default deny) = deny everything except those listed (fail closed = loss of functionality)
  ↓ better for security — most firewalls use
* stateful packet filter = router that checks each packet against provided policy
  ↳ keeps track of all open established conns (looks inside @ data)
  careful to not run out of memory
* stateless packet filters = only look at TCP, UDP, IP headers
  ↳ firewall has no memory
* application layer: restrict traffic according to content of data fields
* PROXY: connect to proxy instead of servers (more complex)
* Firewall principles: ① unbypassable ② tamper resistant ③ verifiable

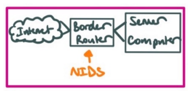
False Negatives: attack happened, no attack reported
False Positive: no attack, attack reported

1) Signature-Based Detection matches known attack structure, blacklisting Pro: good at know signatures, Con: won't catch new attacks, variants	2) Anomaly-Based Detection model of normal activity, flag any deviations Pro: catch new attacks Con: difficult to train model, can have FN, FP
3) Specification-based detection Specify normal activity, flag deviation (whitelist) Pro: catch new attacks, FP can be very low Con: time consuming to write specifications	4) Behavioral Detection Look for evidence of compromise Pro: catch new attacks, can have low FP, cheap Con: detect after started, use different behavior to execute

Intrusion Detection

detect when attacks happen

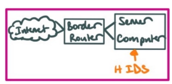
① NIDS = network intrusion detection
↳ between router & internal network



PROS
- single NIDS for entire network, cheap
  easy to scale, simple compute

CONS
- packets out of order / separated
- can't analyze encrypted traffic
- reconstructed TCP ≠ TCP connection w/ end host

② HIDS = host based intrusion dectection
↳ installed directly on end posts



PROS
- check directly what data is received & how it's parsed
- can decrypt data

CONS
- expensive
- path traversal attacks

③ logging = generate logs w/ info on end host

PROS
cheap
similar to HIDs

CONS
not REAL-TIME

Anonymity and Tor → conceal source

onion routing: use multiple chained proxy servers & hope at least ONE is trusted

* security guarantee even if n-1 intermediaries were malicious

Alice $\xrightarrow{(((M,Bob)_{K_{charlie}},Dan)_{K_{Dan}},Frank)_{K_{Frank}}}$ Frank $\xrightarrow{((M,Bob)_{K_{charlie}},Charlie)_{K_{Dan}}}$ Dan $\xrightarrow{(M,Bob)_{K_{charlie}}}$ Charlie $\xrightarrow{M}$ Bob

decrypt w/ priv key

* performance decreases linearly w/ proxies added
* 3 nodes so entry node can't directly collude w/ exit
* correlation attack to see packets enter/exit certain times
* RST injection to detect & block

CSRF (cross site request forgery): force victim to make request

browser will attach session token
server accepts

example:

Defense 1 = CSRF token that's embedded in valid page HTML → request has to have this token

Defense 2 = Referer validation = check which URL the request made from

Defense 3 = SameSite cookie flag = when to send cookie

* subverts same origin

XSS = cross site scripting: inject malicious JS onto page, runs JS when loaded

① Stored XSS: persistently store malicious JS on server (load = trigger script)
FB post "<script>alert("attack")<script/>

② Reflected XSS: create malicious URL: server displays user input
malicious URL = "https://google.com/search?&q=<script>alert("attack")<script/>

DEFENSE: - sanitize input / use HTML encodings
- content security policy = specify list of allowed domains where scripts can be loaded from

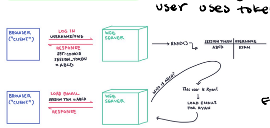
http://www.example.com/index.html?k1=v1&k2=v2

protocol: tells browser how to retrieve resource (http/https)
location: which web server to contact to retrieve
path: which resource on the server
Parameters: (optional) sends info to server

cookies = save info within browser

Domain, Path = which URL
Secure: HTTPS
HTTPOnly: no Javascript access
expires: when to stop storing

session tokens = after login, generate session token → send info to user
user uses token in future way



same origin policy: each site isolated from all others
multiple pages from same origin NOT isolated

EXCEPTIONS:
javascript
images
frames

http://example.com:81/tools/info.html
PROTOCOL, HOSTNAME, PORT, PATH

ORIGIN: PROTOCOL + HOSTNAME + PORT

SQL injection: insert SQL to force it to return sth

* use quotes to end opening quote
* use -- to add comments to parts of query we don't want to execute

Ex) garbage'; SELECT password FROM passwords WHERE username = 'admin' --

'; UPDATE keys set key=Kmallory where username='bob'; --

* can add "or 1=1" to force SQL to return something

Clickjacking / User Interface (UI) Attacks: Fool victim into clicking on attacker input

Steal a click by deceiving the user, ex) download buttons, fake cursor

↳ Defenses: Confirmation Pop-ups, UI Randomization, Delay the click, Direct attention to click

CAPTCHAs: test to make sure user is human, not a bot

- typically machine vision problems, but as algorithm improves, defense gets worse
- reCAPTCHAS used to train AI, human or bot spending $0.10?