Internet Protocol Stack:

Application Layer: HTTPS, SMTP, & FTP protocols here → send sequence data over multiple endsystems (browsers, chat/email clients) (SQL injection, defense: Intrusion detection)

Transport Layer: TCP/UDP → transfer content between 2 endpoints (port numbers) (port scan attack, defense: intrusion detection)

Network Layer: Move Packets between any 2 hosts in network, IP protocol (IP address) (DoS, syn flooding, defense: Syn cookies, firewall)

DataLink Layer: Point-to-Point Protocol. Move packets from one node to the next. (data packets = frames, MAC addresses) (MAC spoofing, defense: detection-&containment)

Physical Layer: Transfer individual bits from one node to the next node w/in the frame (transmits electrically, optically, or w/ radio waves) (no attacks, keep people away (physically) from computer)

HTTPS is compilation of HTTP with SSL. HTTPS (application layer protocol) secures HTTP master communication (http = https + SSL)

Firewall: | application layer: IP spoofing → don't authenticate w/ IP, anti spoof packet filter, IDS
transport: Syn flooding, reset, session hijacking → random port/seqNum; syn cookies, shut syn, Delete; IDS, firewall, encrypt payload

Netfilter:

Provide hooks @ critical points on packet traversal inside Linux Kernel

hooks are rich packet processing & filtering framework

- Each protocol stack defines a series of hooks along packet's traversal path in the stack, developers can use loadable kernel modules (LKM) to register callback functions to hooks
- When a pkt arrives at each of these hooks, the protocol stack calls the netfilter framework w/ the packet and hook #. Netfilter checks if any kernel module has a registered a callback function @ this hook
- each registered module will be called. they're free to analyze/manipulate the pkt → return verdict on pkt

Policy	Firewall Setting
No outside web access	drop all outgoing packets to any IP port 80
No incoming TCP connections, except those for institutions public web server only	drop all incoming TCP syn pkts to any IP except 130.207.244.203 port 80
Prevent web videos from eating up available bandwidth	drop all incoming UDP pkts - except DNS & router broadcasts
Prevent your network from being used for Smurf DoS attack	drop all ICMP pkts going to a "broadcast" address (eg 130.207.255.255)
Prevent your network from being tracerouted	drop all outgoing ICMP TTL expired traffic

Kernel space: Location where the code of the kernel is stored/executes under. when kernel is executing on behalf of the user program (i.e. system call), address space for all kernel threads. Executing code has unlimited access to any mem address space

User space: set of locations where normal user processes run (everything other than kernel). Kernel manages everything running in this space.

SSL based better for bypassing firewalls
IPSEC faster/more universal
coexist

IPSEC: AH and ESP, TLS = reliable transport (TCP), IPSEC = unreliable (can drop/reorder packets)

TLS → MAC then ENCRYPT, IPSEC → ENCRYPT → MAC   IPSEC = OS Level, TLS = App level

AH is only for authentication: incompatible w/ NAT, authenticates data + finds changes during transmission. Provides integrity, data origin authentication, optional replay protection. authentication header, ESP (encapsulating security payload): performs authentication for sender but also encrypts data being sent. Provides confidentiality (encryption), authentication (data integrity, data origin authentication, replay protection), can be used with either/both. ESP authenticates only the IP datagram of IP packet but AH authenticates entire packet.

TAP vs TUN: TAP = Ethernet Level (layer 2) like a switch. TUN = Level 3 network layer to route packets on VPN. Tap bridges where TUN routes

Tap benefits: behaves like real network adapter (except it's virtual), can transport any network protocol. works in layer 2 so ethernet frames are passed over VPN tunnel, can be used in bridges

Tap disadvantages: causes much more broadcast overhead on the VPN tunnel, adds overhead of ethernet headers on all packets transported over the VPN tunnel, scales poorly

TUN benefits: Lower overhead, only transports traffic destined for VPN client, transports only layer 3 IP packets

TAP - virtual ethernet adapter
TUN - virtual "raw" adapter

TUN drawbacks: broadcast traffic is not normally transported, can only transport IPv4, can't be used in bridges

TAP: used for providing virtual network adapters for multiple guest machines connecting to a physical device of the host machine

TUN: sending any packet to TUN will result in the packet being delivered to the user space program.

IPSEC Tunneling: utilizes the Internet Protocol Security Protocol. Has Tunneling mode: original IP packet encapsulated & places into a new IP packet

TLS/SSL Tunneling: Tunneling done outside of the kernel, at app layer. Idea is to put each VPN-bound IP packet inside a TCP or UDP packet, other end of the tunnel will extract the IP packet from the TCP/UDP payload. To secure packets, both ends will use TLS/SSL protocol on top of TCP & UDP

IP prefix: A.B.C.D/x → "/x" is IP prefix: identifies the number of significant bits used to identify network. E.x. 192.9.205.22/18 ⇒ first 18 bits are used to represent the network & the remaining 14 bits are used to identify hosts. IP address is 32-bit identifier for host, router interface

TCP ATTACKS: TCP SYN Flooding Attack: fill the TCB queue storing the half-open connections so that there will be no space to store TCB for any new half-open connection, basically the server cannot accept any new SYN Packets. Defense: SYN cookies: server receives a SYN packet, it calculates a keyed hash (H) from the info in the packet using secret key (KS) that is only known to the server (MAC). The hash (H) is sent to the client as the initial seqNum. from the server (H = SYN COOKIE). Server won't store half-open connection in its queue if the client is an attacker, H will not reach the attacker. If the client is not an attacker, it sends (H+1) to the ACK field. Server checks if the number in the ACK field is valid by recalculating the cookie.

TCP Reset Attack: Goal is to break up connection from A → B. Telnet, SSH, video-streaming connections. Spoof RST pkt by setting correctly: src IP, src Port, dest. IP, dest Port, seq num (SSH encrypts but not TCP header so unprotected)

Defense: encrypt connections using VPN. Includes TCP header. An attacker would have to do a TCP reset attack on all encrypted connections causing collateral damage

TCP Session Hijacking: inject data into an established connection: spoof TCP Pkt if set correctly: src port/IP, dest IP/Port, seqNum

Session Hijacking: remove a secret, removed by the spoofed command from the attacker, Hijacked TCP connection freezes

Defense: random src port / initial seq. Num. although not effective against local attacks. Encrypt payload w/ TLS/SSL, Intrusion detection System

After Hijacking, run a reverse shell: shell process running on a remote machine connecting back to the machine, uses one end of TCP connection for input/output and the other is used/controlled by attacker machine

TCP 3-Way Handshake: First, client establish connection w/ server. sends segment w/ SYN and informs server the client should start communication and w/ what should be its SeqNum. Next, server responds to client request with syn/ACK signal set, ACK helps signify response of segment that is received & SYN signifies what seqNum it should be able to start w/ the segments. Third/Finally, client acknowledges response of server & they both create stable connection beginning w/ actual data transfer process. Fin used to terminate.

Private IP addresses 10.0.1.12, 10.0.1.17, 10.0.1.19 are in a local network behind a NAT'd router that sits between these 3 hosts & the larger internet. hosts IP datagrams being sent from, or are destined to, these 3 hosts must pass through NAT router. Routers interface on the Lan side has IP 10.0.1.28, internet side = 135.122.204.208. Host w/ IP 10.0.1.19 sends IP datagram to 128.119.167.188. Src Port is 3411, dest. port is 80.

src address = local host's IP (10.0.1.19) after sent by host but before reaches router

Dest. address = remote machine IP (128.119.167.188) at step 1, what is dest IP address

datagram @ step 2, after it's been transmitted by the router. Src IP = 135.122.204.208

step 2, dest address: 128.119.167.188 (remote machine)

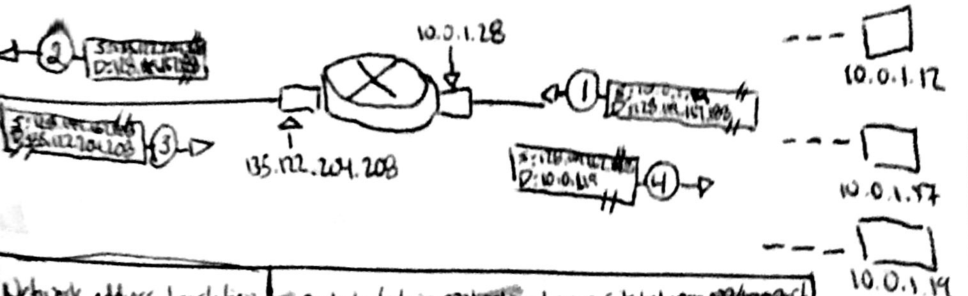
Will the src port change? Yes, NAT will change it

step 3, just before received by router, src IP = 128.119.167.188 (remote machine)

step 4, transmitted by router but hasn't been received by host. src IP = 128.119.167.188 (remote machine)

step 4 dest address = 10.0.1.19 (local host)

Has a new entry been made in the routers NAT TABLE? No, entry is made on outbound requests, only happens in step 1 & 2



NAT Translation Table

WAN side	LAN side
135.122.204.208	10.0.1.19

TLS = protocol that provides secure channel between 2 communicating applications

Confidentiality: No one but the two ends can see the actual content of data transmitted.

Integrity: Channel can detect data changes during transmission

Authentication: at least one end of channel needs to be authenticated

SSL sits between transport & application layer

Unprotected data given to app layer. TLS handles enc, dec, & integrity. Protected data given to transport layer.

Before secure communication, encryption algorithm/keys(KS) & MAC algorithm (Mx), and key exchange algorithm need to be set up.

given IP 1.2.3.4 & netmask 255.255.255.0, what is network address? 1.2.3.0

Network address translation | SSL protects data in application layer. Sits between app/transport

SSL client/server agree on cipher suite, client offers choices & server picks one. | Smurf attack: attacker generates ping stream to target network broadcast address w/ spoofed source IP set to victim host. Every host on ping target network will generate ping reply (ICMP echo reply). Amplified ping stream can easily overwhelm victim's network connection.

Phase 2: key derivation
- = enc. key for data sent from client to server
- = MAC key for data sent from client to server
- = enc. key for data sent from server to client
- = MAC key for data sent from server to client

→ Keys derived from key derivation function which takes master secret & possibly some additional random data & creates keys

Each record carries MAC. Never use same cryptographic functions. (keys for many)

Real SSL handshake steps:
1) client sends list of algorithms it supports, along w/ client nonce
2) Server chooses algorithms from list, sends back choice & cert & server nonce
3) Client verifies cert, extracts server's public key, generates pre_master_secret, enc. w/ server pub. key, sends to server
4) client/server independently compute enc/MAC keys from pre_master_secret & nonces
5) client sends MAC of all handshake messages. 6) server sends MAC of all handshake messages. (See next line) this and last 2 messages = encrypted

Last 2 steps protect handshake from tampering. Client sends range of strong → weak algorithms, man-in-middle could delete strong algorithms, last steps prevent

Attacker can capture & replay or reorder records of SSL, what prevents this? use of random nonce and adding sequence number into MAC. For integrity Mx, SeqNum, data type & app data included in MAC

SSL encrypts data in constant stream & perform message integrity until all data processed. SSL uses pub key for key exchange only, then switches to symmetric key for data communication

SSH subjects to RST ATTACKS BC it doesn't encrypt TCP Header. Telnet = port 23

TCP/UDP header: Port # = 16 bits. # of TCP && UDP ports (each) 2^{16}

Syn flooding attack on port 23 won't effect listening on port 8080

A & B Machines on 2 TCP programs [illegible] on A send their data to TCP server listening to port 8023 on machine B. No they will not mix data on server side. TCP allocates different buffers for different communication. They have individual src ports even from same machine so server treats them as different connections. Data coming from each connection will have their own buffers.

Checksum:
0101-0011 | 0111-0100
0110-0110 | 0011-1001
1011 1001 | 1010-1101
wrap around → 1
flip → 0010-1110
1101-0001

Using TCP/UDP checksum: 5-bit error may go undetected. 1-bit errors detected. 2 bit undetected, 4 bit undetected. Syn flooding won't cause victim's machine to freeze.

TCP 3-way handshaking: Syn request Seq.Num = 5432. Syn-Ack reply = 6789. All packets carry 10 bytes payload in both directions. ACK response: [illegible] Seq.Num = 5442, acknowledgment # = 6799

TCP: reliable, in-order delivery. congestion/flow control & connection setup.
UDP: unreliable, unordered delivery. connectionless, no frills extension of best-effort IP
→ simple, small overhead, no connection establishment, blast away as fast as desired. No congestion control.

Alice wants to send Bob a dig sig w/ message M.
Alice: Equation: 1) H(m) 2) KA_private[H(m)] 3) M + KA_private[H(m)]
RSA
Expression: 5 + ((5%13)^3 mod 40)

Bob: 1) separate M & KA_private[H(m)]
2) Decrypt dig sig (KA_public(KA_private[H(m)]) = H(m)
3) Calculate H(m)
4) compare H(m) with H(m) → if equal, accept, else: drop

1) 5 + ((5%13)^13) mod 40 = 5, ((5%13)^3 mod 40)
2) ((5%13)^3) mod 40)^27 mod 40 = 5%13
3) 5%13 → compare result from steps 2&3 → accept / drop

Pub Key Crypto: Alice wants to send secret message to Bob. Bob wants to be sure that Alice sent the message.

Alice hashes message/message digest, encrypts w/ private key → send to Bob. Bob splits hash/msg, hashes msg, decrypt w/ Alice pub-key, compare received & generated hashes.

Alice: Equation: KB_public[KA_private[M]]; Expression: 29^(5^3 mod 40) mod 35; send cipher = 29^(5^3 mod 40) mod 35

Bob: Equation: KA_public(KB_private(KB_public[KA_private(M)]))

Get cipher: 29^(5^3 mod 40) mod 35

Apply KB_private: (29^(5^3 mod 40) mod 35)^5 mod 35

Apply KA_public: ((29^(5^3 mod 40) mod 35)^5 mod 35)^27 mod 40

RSA

Alice: KA pub: (40, 27), KA priv: (40, 3), Bob: pub (35, 5), Priv (35, 29)
Hash: H(x) = X % 13
Alice: Symm. Key = KS = 5, Msg. = M = 15 (Not known from Bob)

Alice wants to send secret msg using shift cipher

Alice: Equation: KS(M) Expression: 15+5

Bob: Equation: KS(KS(M)) Expression: (15+5) − 5

CBC Mapping

Input	Output	Input	Output
000	110	100	001
001	111	101	011
010	101	110	000
011	100	111	010

Sender generates plaintext 100-101-110 w/ CBC IV = 101

Ciphertext = 101-111-101-100

Mi	010	111	000	101
101	101	111	101	100
010	111	000	101	001

Receiver gets ciphertext 010-100-100-101 w/ CBC

Plaintext: 001-111-110

c0	c1		
010	100	110	010
010 → 010	100	101	
	110	010	111

RSA

e = 5, d = 29
5 < 91
e < n
(p-1)
Z = (p−1)(q−1) = 72

P = 7, q = 13, assume 3 < e < 6 & d < 50
Pub Key = (91, 5)
$c^d = m^{ed}$
Priv. Key = (91, 29)

IP Spoofing: changing/disguising the source IP of a system.
Helps attackers hide themselves. Undermine systems that use just IP for authentication/filtering.
{ simply spoofing: change IP address / or spoofing w/ source routing

Simply spoofing advantages: hide src of packet flood or DoS attack/effective & easy to use. Disadvantages: Not interactive/Not receiving response

Source routing: option in IP that allows src machine sending a pkt to specify routing path

Loose src routing: src specify just some of the hops that packet shave to take | strict src routing: entire route included in pkt header

Subject to IP spoofing. Defenses: avoid IP for authentication (use passwords, crypto tech, etc. IPSEC). Implement "anti-spoof" packet filter at border router and firewalls. Connecting your organization to the internet & business partners. Firewall && IDS

Network Mapper: automated ping sweep & traceroute tool. Support OS fingerprinting using TCP stack fingerprinting, selecting appropriate icons for hosts based on OS detection

Defenses: use firewalls & packet filtering capabilities (at router) → at gateway: block incoming ICMP messages, except for hosts meant to be public web servers

Can filter to allow specific IPs like your ISP to be able to send Pings. Can filter ICMP time exceeded messages. Limits attacker info but also network management

Port scanning: [illegible]. Secure all ports to system. Attacker only needs one path in.

Signature: Several packets to different dest ports from same src in a short time. TCP SYN to non-listening port. Many fast connection ACK attempts.

TCP Port Scans:

TCP Connect: Polite scan by completing 3-way handshake w/ each target port on system

TCP SYN Scans: Half-open scan, stealthier than ↑ Target host won't log connection but firewall may. Faster, may crash host. Violates Protocol.

TCP Fin, Xmas Tree, & Null scans: send unexpected packets to target host to get feedback (eg reset packet). Does not work against MS → no response sent.

TCP ACK Scans: Violates Protocol. Can get through packet filtering devices like router/firewall.

Scanning Programs: Network Mapper (NMAP)

Active Port scanning: allows to scan/discover open ports/specific network/hosts & where it's running

Host discovery: lets you ID potential hosts that are responding to network requests.

OS Detection: discover OS name/version/network details

Application Version Detection: nmap can also be used to determine what kinds of apps are running/version #

Defenses: Close all unnecessary ports/harden system → Find open ports, check if necessary, disable permanently if not necessary. Stateful packet filtering, Intrusion prevention/detection systems.