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

<u>Trusted Computing Base</u>: 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.

<u>Access Control</u>: a set of rules that limit access to a system.

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

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

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

<u>Privilege Separation</u>: 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).

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

<u>Address Space Layout Randomization</u> (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 ⊂ {0, 1, ... , p-2} and computes $A = g^a \bmod p$.
4. Bob picks a secret value b ⊂ {0, 1, ..., 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 ⊂ {0, 1, ..., 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 ⊂ {1, ..., p-1} to Bob, she picks a random value r ⊂ {0, ... , 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' ≠ x but H(x) = H(x').
3. Collision Resistant: Infeasible to find any x, x' such that x' ≠ 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 $p = 2 \pmod 3$ and $q = 2 \pmod 3$ then ∃ 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)) = ∀ 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) = $H(M)^d$ mod n
- Verify(M, S, n) = Boolean(H(M) == S^3 mod 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.