Notes from Network Security Introduction Slides

Network Security Requirements

- **Computer Network Security** is the protection given to an automated information system to achieve the goals of preserving the integrity, availability, and confidentiality of information system resources, including hardware, software, firmware, information/data, and telecommunications.
- **Confidentiality:** Ensures that private or confidential information is not made available or disclosed to unauthorized individuals.
    - **Data confidentiality:** Protects information from unauthorized access.
    - **Privacy:** Gives individuals control over what information related to them is collected and stored, and who can access and disclose that information. For example, student grade information.
- **Integrity:** Ensures that information and systems remain unaltered and trustworthy.
    - **Data Integrity:** Ensures that data (both stored and transmitted) and programs are changed only in a specified and authorized manner.
    - **System Integrity:** Ensures that a system performs its intended functions without being compromised by unauthorized manipulation. For example, a hospital patient's allergy information.
- **Availability:** Ensures that systems and information are accessible to authorized users when needed. This means systems work promptly and service is not denied to authorized users. For example, protection against denial of service attacks.
- **Other Security Requirements:**
    - **Authenticity:** Verification of the genuineness of a user, process, or device.
    - **Accountability:** The ability to trace actions to a specific individual or entity. This includes:
        - **Tracible Data Source:** Knowing the origin of data.
        - **Fault Isolation:** Identifying the source of a problem.
        - **Intrusion Detection and Prevention:** Identifying and stopping unauthorized access attempts.
        - **Recovery and Legal Action:** Restoring systems after an attack and pursuing legal action against attackers.
        - **System Activity Records:** Maintaining logs of activities to enable forensic analysis and investigation.

OSI Security Architecture

- Recommended by the International Telecommunication Union – Telecommunication (ITU-T) as the X.800 standard.
- Provides a systematic approach to defining and implementing security requirements in the Open Systems Interconnection (OSI) model.
- Used by IT managers and vendors to build secure products.

- Components of OSI Security Architecture:
    - **Security Attacks:** Actions that compromise the security of information.
        - **Passive Attacks:** Aim to gain information without altering data. Difficult to detect but easy to prevent. Examples include eavesdropping and traffic analysis using a sniffer.
        - **Active Attacks:** Involve modifying data or disrupting service. Examples include replay attacks, message modification, denial of service, and masquerade attacks.
    - **Security Mechanisms:** Processes or devices designed to detect, prevent, or recover from attacks.
    - **Security Services:** Enhance the security of data processing systems and information transfers. Includes policies and procedures.

Other Security Architectures

- **NIST Cybersecurity Framework (CSF):** Provides a framework for managing cybersecurity risk.
- **OWASP - Open Web Application Security Project:** Focuses on web application security.
    - **OWASP Application Security Verification Standard (ASVS):** A standard for verifying the security of web applications.
    - **OWASP Web Security Testing Guide:** A guide for testing the security of web applications.

Threat Model

- **Definition:** A model that describes the potential attackers, their motivations, and the resources they have.
- Understanding your threat model helps you design effective security measures.
- **Story of the Bear Race:** Illustrates that even imperfect defense is better than none, emphasizing the importance of staying on top of security best practices.

Human Factors in Security

- **Users:** Prioritize convenience, may find ways to circumvent security measures for ease of use.
- **Programmers:** Can make mistakes, use tools that can lead to errors (e.g., C and C++).
- **Social Engineering Attacks:** Exploit trust to gain unauthorized access.
- **Importance of Security by Design:** Integrate security considerations from the start, rather than patching vulnerabilities later.

Symmetric Encryption

- **Definition:** Sender and recipient share a common secret key for both encryption and decryption.
- **Components:**

    - **Plaintext:** The original message.
    - **Encryption Algorithm:** Transforms the plaintext into ciphertext.
    - **Secret Key:** Used by both the encryption and decryption algorithms.
    - **Ciphertext:** The encrypted message.
    - **Decryption Algorithm:** Recovers the plaintext from the ciphertext using the same secret key.
- **Requirements:**
    - Strong Encryption Algorithm
    - Secure Key Distribution: Requires a secure channel to share the key.
- **Security:** Relies on the secrecy of the key.
- **Types:**
    - **Block Ciphers:** Process data in fixed-size blocks. Examples include DES, 3DES, and AES.
    - **Stream Ciphers:** Encrypt data bit-by-bit using a keystream. Examples include RC4.

Block Ciphers and Feistel Structure

- **Block Ciphers:** Operate on fixed-size blocks of data, typically 64 or 128 bits.
- **Feistel Cipher Structure:**
    - Most symmetric block ciphers use this structure.
    - Based on substitution (S-box) and permutation (P-box) operations.
    - Provides confusion and diffusion of the message.
    - Invented by Horst Feistel in 1973.
    - Divides the input block into two halves and processes them through multiple rounds.
    - Each round performs substitution on one half based on a function of the other half and a subkey, followed by permutation (swapping halves).

Data Encryption Standard (DES)

- Uses 64-bit plaintext blocks and a 56-bit effective key length.
- **Weakness:** Short key length makes it vulnerable to brute-force attacks.

Triple DES (3DES)

- Applies DES three times with different keys, increasing the effective key length to 168 bits.
- **FIPS-approved symmetric encryption algorithm.**
- **Weakness:** Slow encryption speed.

Advanced Encryption Standard (AES)

- Developed as a replacement for DES due to its vulnerabilities and slow speed.
- Selected by NIST in 2000 and standardized as FIPS PUB 197 in 2001.
- **Symmetric block cipher using 128-bit data blocks and 128/192/256-bit keys.**

- Stronger and faster than Triple-DES.
- **Rijndael Algorithm:**
  - Iterative cipher, not a Feistel cipher.
  - Processes data as four groups of four bytes (128 bits).
  - Performs several rounds of byte substitution, shift rows, mix columns, and add round key operations.
  - Number of rounds depends on the key size (10 for 128-bit keys, 12 for 192-bit keys, 14 for 256-bit keys).
- **Advantages:**
  - Efficient and fast due to the use of XOR and table lookup operations.

Random and Pseudorandom Numbers

- **Uses in Cryptography:**
  - Generating stream keys for symmetric stream ciphers.
  - Generating keys for public-key algorithms (e.g., RSA).
  - Creating session keys for secure communication protocols (TLS, Wi-Fi, email security, IP security).
  - Key distribution scenarios (e.g., Kerberos).
- **Types:**
  - **True Random Numbers:** Non-deterministic and unpredictable, generated from physical sources of randomness (e.g., radioactive decay, electrical noise).
  - **Pseudorandom Numbers:** Deterministically generated but appear random, produced using pseudorandom number generators (PRNGs).
- **Properties of Random Numbers:**
  - Randomness: Lack of predictable patterns.
  - Uniformity: Even distribution of bits.
  - Independence: Subsequences cannot be inferred from each other.
  - Unpredictability: Satisfies the "next-bit test," meaning the next bit in the sequence cannot be predicted based on previous bits.
- **Entropy:**
  - A measure of uncertainty and unpredictability.
  - High entropy is desirable in cryptography as it means the output is more random.
  - Measured in bits. 3 bits of entropy represent a uniform distribution over 8 values.
- **True Random Number Generators (TRNGs):**
  - Utilize natural sources of randomness such as:
    - Radioactive decay.
    - Electrical noise.
    - Audible noise.
    - Keyboard timings.
    - Disk electrical activity.
    - Mouse movements.
    - Physical unclonable functions (PUFs).
  - Combining multiple sources, including weak and strong ones, can improve entropy.

- **Pseudorandom Number Generators (PRNGs):**
  - Deterministic algorithms that use a small amount of true randomness (seed) to generate a larger sequence of random-looking output.
  - Efficient and less expensive than TRNGs.
  - Security relies on the computational indistinguishability of their output from true randomness.
- **PRNG Definition and Properties:**
  - Functions:
    - `PRNG.Seed(randomness)`: Initializes the internal state using entropy from truly random bits.
    - `PRNG.Generate(n)`: Generates n pseudorandom bits, updating the internal state as needed.
  - Properties:
    - Correctness: Output is deterministic based on the seed.
    - Efficiency: Fast generation of pseudorandom bits.
    - Security: Output should be computationally indistinguishable from random to an attacker.
    - Rollback Resistance: Past output cannot be deduced from the current state.
- **Example of PRNG Construction:**
  - Using a block cipher in counter (CTR) mode.
  - Concatenates the output of multiple block cipher encryptions with an incrementing counter value to generate a stream of pseudorandom bits.
- **PRNG Security:**
  - Not truly random as output is deterministic given the seed.
  - A secure PRNG is computationally indistinguishable from random, meaning an attacker cannot predict future output.
- **Creating Pseudorandom Numbers in Programming:**
  - The `rand()` function (in C's `stdlib.h`) generates pseudorandom numbers within a defined range.
  - `arc4random()` (available in some operating systems) provides a more secure and higher-quality source of random numbers compared to `rand()`.

Stream Ciphers

- **Definition:** Process data bit-by-bit using a keystream generated from a key.
- **Encryption:** Plaintext bits are XORed with the keystream to produce ciphertext.
- **Key Reuse:** Reusing the same keystream compromises security as it allows attackers to recover the message.
- **Design Considerations:**
  - Statistically random keystream.
  - Sufficiently large key length.
  - Large linear complexity.
  - Correlation immunity.

    - Confusion and diffusion properties.
- **Keystream Generation:**
    - Use Pseudorandom Number Generators (PRNGs).
    - Expand a short random seed (secret key) into a long, random-looking keystream.
- **Encrypting Multiple Messages:**
    - To avoid key reuse, seed the PRNG with the key and a random Initialization Vector (IV) or nonce.
    - Send the IV along with the ciphertext, allowing the receiver to correctly initialize their PRNG.

RC4 Stream Cipher

- **Description:**
    - Proprietary cipher designed in 1987.
    - Simple, fast, and efficient, especially in software.
    - Adaptable to any key length.
    - Used in web SSL/TLS, wireless WEP, and WAP.
- **Key Schedule:**
    - Initializes an array S with numbers 0 to 255.
    - Uses the key to shuffle the array, creating the internal state of the cipher.
- **Encryption:**
    - Continuously shuffles array values.
    - The sum of a shuffled pair selects the "stream key" value.
    - The stream key is XORed with the plaintext byte to produce the ciphertext byte.
- **Security:**
    - Considered secure against known attacks.
    - Key reuse is a major vulnerability.
    - Known biases in the output, especially in the first part of the keystream.
    - **RC4-Drop[n]:** A variant that discards the first n bytes of the keystream to mitigate biases. Recommended to set n = 3072.

Public-Key Cryptography

- **Definition:** Uses two keys - a public key for encryption and a private key for decryption.
- **Advantages:**
    - Eliminates the need for a secure channel to share keys.
- **Disadvantages:**
    - Significantly slower than symmetric-key cryptography due to complex mathematical operations.
- **Key Pairs:**
    - Each user has a public key that is known to everyone and a private key that is kept secret.
    - Messages encrypted with the public key can only be decrypted with the corresponding private key.
- **Applications:**

    - Encryption/Decryption: For confidentiality.
    - Digital Signatures: For authentication and non-repudiation.
    - Key Exchange: Securely sharing symmetric keys.

RSA Public-Key Encryption

- **Description:**
    - Invented by Rivest, Shamir, and Adleman in 1977.
    - Widely used in internet security, including PKI products, SSL/TLS, and secure email.
    - Based on exponentiation in a finite field over integers modulo a prime number.
- **Key Setup:**
    1. Choose two large prime numbers, p and q.
    2. Calculate the modulus n = p * q.
    3. Calculate Euler's totient function: $\varphi(n) = (p - 1) * (q - 1)$.
    4. Choose an encryption key e such that $1 < e < \varphi(n)$ and $\gcd(e, \varphi(n)) = 1$ (e and $\varphi(n)$ are relatively prime).
    5. Calculate the decryption key d such that $ed \equiv 1 \pmod{\varphi(n)}$.
    6. Public key: (e, n).
    7. Private key: (d, p, q).
- **Encryption:**
    - Ciphertext $C = M^e \bmod n$, where M is the plaintext message.
- **Decryption:**
    - Plaintext $M = C^d \bmod n$.
- **Key Generation Requirements:**
    - Primes p and q must be large and difficult to derive from the modulus n.
    - Exponents e and d are calculated using the inverse modulo operation.
- **Correctness:**
    - Based on Euler's theorem and Fermat's little theorem.
    - Decryption recovers the original message because $(M^e)^d \equiv M \pmod{n}$.
- **Attack Approaches:**
    - Mathematical Attacks: Attempt to factor the modulus n to obtain the private key. Prevented by using large key sizes.
    - Timing Attacks: Exploit variations in decryption time to gain information about the private key.
    - Chosen Ciphertext Attacks: Target weaknesses in the RSA algorithm. Mitigated by using padding schemes like PKCS1 v1.5.

Homomorphic Encryption

- **Definition:** Enables computation on encrypted data without decryption.
- **Types:**
    - Partially Homomorphic Encryption: Supports either addition or multiplication on ciphertexts, but not both (e.g., RSA).

    - Fully Homomorphic Encryption (FHE): Allows both addition and multiplication on ciphertexts.
- **Applications:**
    - Cloud Computing: Allows computations on sensitive data stored in the cloud without compromising confidentiality.

Message Authentication

- **Purpose:**
    - Ensuring message integrity (protection against alteration).
    - Validating the sender's identity (authentication).
    - Providing non-repudiation (proof of origin).
- **Methods:**
    - Message Encryption: Symmetric encryption provides authentication as only the sender and receiver know the key.
    - Message Authentication Code (MAC): A cryptographic checksum generated using a shared secret key.
    - Digital Signature: Uses asymmetric cryptography to verify the authenticity and integrity of a message.
- **Symmetric vs. Public-Key Encryption for Authentication:**
    - Symmetric encryption provides authentication because only the sender and receiver share the key.
    - Public-key encryption alone does not provide authentication as anyone can encrypt using the public key.
    - Digital signatures, using the sender's private key, can be used with public-key encryption to provide both confidentiality and authentication.

Hash Functions

- **Definition:** A one-way function that maps an input of any size to a fixed-size output (hash value).
- **Use Case Example:** Using a shared secret key to generate a hash of a message for integrity verification.
- **Requirements for Secure Hash Functions:**
    1. Can be applied to any size message.
    2. Produces a fixed-length output.
    3. Easy to compute.
    4. **One-way Property (Preimage Resistance):** Infeasible to find an input that produces a given hash value.
    5. **Weak Collision Resistance (Second Preimage Resistance):** Given an input, it's infeasible to find a different input that produces the same hash.
    6. **Strong Collision Resistance:** Infeasible to find any two different inputs that produce the same hash.
- **Collision Resistance:**
    - Collisions occur when two different inputs produce the same hash.

        - Collision resistance makes finding collisions computationally infeasible.
- **Secure Hash Functions:**
    - Meet the first five requirements (including weak collision resistance).
    - Exhibit randomness and unpredictability in their output.
    - Small changes in input result in significant changes in output.
- **Message Digest:**
    - A cryptographic hash used to ensure data integrity.
    - Examples: SHA-1, SHA-2, SHA-3, MD5.
- **Hash Function Examples and Security:**
    - MD5: 128-bit output, completely broken.
    - SHA-1: 160-bit output, broken in 2017.
    - SHA-2: 256, 384, or 512-bit output, considered secure but some variants have vulnerabilities.
    - SHA-3 (Keccak): 256, 384, or 512-bit output, designed as an alternative to SHA-2, considered secure.
- **Length Extension Attacks:**
    - Exploit vulnerabilities in some hash functions (e.g., SHA-2) to append data to a message without knowing the original message.
    - SHA-3 is not vulnerable to length extension attacks.
- **Integrity and Threat Model:**
    - Hashes provide integrity only if the hash value itself is protected from modification.
    - Example: Downloading a file and verifying its hash against a value published on a secure website provides integrity.
    - **Man-in-the-Middle Attack:** An attacker can modify both the message and the hash, rendering the hash ineffective for integrity.
- **Solutions to Integrity Issues with Hashes:**
    - Message Authentication Codes (MACs): Use a secret key to generate the hash, preventing modification by attackers who don't have the key.
    - Digital Signatures: Encrypt the hash using the sender's private key, providing authentication and non-repudiation.

Message Authentication Code (MAC)

- **Definition:** A small, fixed-size block generated from a message and a secret key, providing integrity and authentication.
- **Usage:**
    - The sender computes a MAC using a shared secret key and appends it to the message.
    - The receiver computes the MAC on the received message and compares it with the received MAC to verify integrity and authenticity.
- **Definition and Properties:**
    - Functions:
        - `KeyGen() → K`: Generates a secret key K.

        - `MAC(K, M) → T`: Generates a tag T for the message M using key K.
    - Properties:
        - Correctness: Deterministic output for the same input.
        - Efficiency: Fast computation.
        - Security: Existentially unforgeable under chosen plaintext attacks.

HMAC (Hash-based MAC)

- **Description:**
    - Defined in RFC 2104.
    - Used in IP Security, TLS, and SET.
- **Algorithm:**
    - `HMAC(K, M) = H[(K+ ⊕ opad) || H[(K+ ⊕ ipad) || M]]`, where:
        - K+ is the key padded to the block size of the hash function.
        - opad is a constant outer padding (0x5c repeated).
        - ipad is a constant inner padding (0x36 repeated).
        - H is the hash function (e.g., SHA-256).
    - Derives two keys from the original key K using ipad and opad.
- **Properties:**
    - Inherits the properties of the underlying hash function (collision resistance, one-way property).
    - Provides integrity and authenticity if the key is kept secret.
    - Unforgeable: Attackers cannot create valid MACs without the key.

Authenticated Encryption

- **Definition:** Provides both confidentiality and integrity (and authenticity).
- **Approaches:**
    1. Combining separate schemes for confidentiality (encryption) and integrity/authenticity (MAC).
    2. Using a scheme designed for both confidentiality and integrity/authenticity.
- **Methods for Combining Encryption and MAC:**
    - MAC-then-Encrypt: `Enc(K1, M || MAC(K2, M))`
    - Encrypt-then-MAC: `MAC(K2, Enc(K1, M))`
- **Encrypt-then-MAC vs. MAC-then-Encrypt:**
    - Both can be secure if implemented correctly.
    - **Encrypt-then-MAC is more robust to mistakes.**
    - MAC-then-encrypt can be vulnerable to side-channel attacks because decryption is performed before integrity verification.

TLS 1.0 "Lucky 13" Attack

- **Description:**
    - Exploited a vulnerability in TLS 1.0's use of MAC-then-encrypt with AES-CBC encryption.

    - Attackers could guess plaintext bytes by measuring the time it took for the MAC verification to fail.
- **Takeaways:**
    - Highlights the importance of side-channel attack prevention.
    - Reinforces the preference for encrypt-then-MAC.

Digital Signatures

- **Definition:** Provide authentication, integrity, and non-repudiation using asymmetric cryptography.
- **Definition and Properties:**
    - Functions:
        - `KeyGen() → PK, SK`: Generates a public/private key pair.
        - `Sign(SK, M) → sig`: Signs the message M using the private key SK to produce signature sig.
        - `Verify(PK, M, sig) → {0, 1}`: Verifies the signature using the public key PK, outputting 1 for valid and 0 for invalid.
    - Properties:
        - Correctness: Verifies correctly for valid signatures.
        - Efficiency: Fast signing and verification.
        - Security: Unforgeable, even with access to the public key.

RSA Signatures

- **Key Generation:**
    1. Select two large primes, p and q.
    2. Compute n = pq.
    3. Choose an exponent e relatively prime to (p - 1)(q - 1) and 2 < e < (p - 1)(q - 1).
    4. Calculate d = e^{-1} mod (p - 1)(q - 1).
    5. Public Key: (e, n)
    6. Private Key: d
- **Signing:**
    - Compute `H(M)<sup>d</sup> mod n`, where H(M) is the hash of the message.
- **Verification:**
    - Check if `H(M) ≡ sig<sup>e</sup> mod n`.
- **RSA Digital Signature Algorithm:**
    - Involves hashing, padding, and encryption using the signer's private key.
- **Correctness:**
    - Mathematical proof based on modular arithmetic and Euler's theorem.
- **Security:**
    - Relies on the hardness of factoring large numbers (factoring hardness assumption) and the discrete logarithm problem.
    - Using a salt adds security by making signatures unique even for the same message and key.

Hybrid Encryption

- **Motivation:**
    - Public-key encryption is slow and inefficient for large messages.
- **Definition:**
    - Uses symmetric encryption for speed and asymmetric encryption for key exchange.
- **Process:**
    1. Generate a random symmetric key K.
    2. Encrypt the message using symmetric encryption with key K.
    3. Encrypt the symmetric key K using the recipient's public key.
    4. Send both the encrypted message and the encrypted symmetric key.
- **Benefits:**
    - Combines the speed of symmetric encryption with the security of asymmetric encryption.
    - Enables secure and efficient encryption of large messages.

This response is based on information directly taken from the sources you provided.