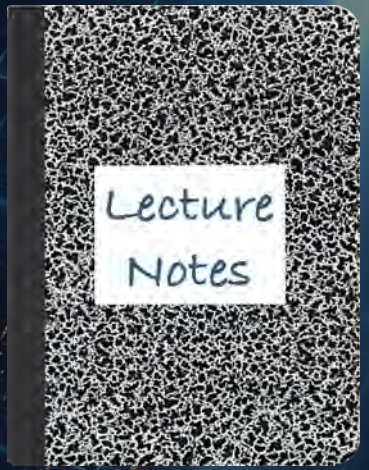
CS 417 – DISTRIBUTED SYSTEMS

Week 12: Recitation
Security Review



Paul Krzyzanowski

Security Goals

Security Goals

- **Confidentiality**: data kept secret from unauthorized parties
- **Integrity**: unauthorized modification can be detected
- **Authentication**: establishing who is on the other end of a connection
- **Authorization**: deciding what an authenticated principal can do
- **Non-repudiation**: a principal cannot deny having approved data
  - **Principal**: any identifiable entity (user, service, device, process)

Cryptographic Building Blocks

Symmetric vs. Asymmetric Cryptography

- **Symmetric encryption**: same key for encryption and decryption
  - Fast; used for bulk data encryption
  - But both sides need the same key
  - Key distribution problem: h*ow do parties share a key securely?*

- **Asymmetric cryptography**: public key + private key pair
  - Encrypt with public key; only private key can decrypt
  - Sign with private key; anyone with public key can verify
  - Enables key establishment and digital signatures without pre-sharing a key
    - Encrypt a key for a symmetric algorithm using the other side's public key
    - Only they can decrypt

MACs and Digital Signatures

- **Hash function**: arbitrary input produces a fixed-size digest
  - Problem: an attacker can modify the message and recompute the hash

- **MAC (Message Authentication Code)**: hash of message + shared secret key
  - Proves message is unmodified and from a party holding the key
  - HMAC is the common mechanism – essentially a hash of the message and key

- **Digital signature:** sender signs a hash(message) using their private key
  - Anyone with the public key can verify -- no pre-shared secret needed
  - Also supports non-repudiation (the creator cannot dely

Comparing Integrity Mechanisms

- **Hash**: no secret key required
  - Does not resist active attackers -- hash can be recomputed
  - No origin authentication

- **MAC**: requires a shared secret key
  - Resists active attackers; authenticates origin to any parties that have the key

- **Digital signature**: uses private/public key pair
  - Resists active attackers; anyone with the public key can verify
  - Supports non-repudiation – only the owner of the key could have created it
  - No pre-shared secret required

Replay Attacks and Freshness

Problem: integrity checks verify a message is unmodified, not that it is fresh

- **Replay attack**: attacker captures and retransmits a valid message later
  - Original is genuine, so integrity check passes
  - Example: replay a signed "transfer $1000" request multiple times

Freshness mechanisms:

- **Nonce**: per-session random value created by the receiver
  - A replayed message carries the old session's nonce: the receiver rejects them
- Also: **timestamps** (origin time or expiration time), **sequence numbers**

Secure Channels

Transport Layer Security (TLS)

Secures communication over untrusted networks (HTTPS, gRPC, and more)

- **Handshake**: server presents certificate
  - Client validates against a CA
  - Parties derive symmetric session keys via asymmetric key exchange
- **TLS provides**: confidentiality, integrity, and server authentication
  - **Certificate**: signed statement binding a public key to an identity
  - **CA (Certificate Authority)**: issues and signs certificates
  - Chain of trust: root CA → intermediate CAs → end-entity certificates

Mutual TLS (mTLS)

Standard TLS authenticates only the server

- **mTLS**: both sides present and verify each other's certificates
  - Each party gets a cryptographically verified identity for the other
- Example: a *payments* service receives a connection
  - Without mTLS: only knows the source is inside the cluster based on the IP address
  - With mTLS: verifies the caller is specifically the orders service
- **Challenge**: certificate management at scale
  - Every service needs a certificate
  - **Service meshes** (Istio, Linkerd) automate the issuance and rotation of certificates for services

Identity and Authorization

Authentication vs. Authorization

- **Authentication**: *who are you?* — establishes identity
- **Authorization**: *what are you allowed to do?* — enforces policy
- These are separate questions
  - getting authentication right does not imply correct authorization
- **Broken object-level authorization**:
  - Service authenticates caller correctly, but does not check whether this caller may access this specific resource
  - Example: change the order ID in a request to be able to read any customer's order
- **Authorization must happen at every layer:**
  - **Edge** (API access), **service** (specific operation), **data layer** (specific data record)

OAuth and OpenID Connect

- **OAuth: authorization framework**
  - Answers: what access is being delegated?
  - Produces a scoped, short-lived access token
    - Allow an app to read my Google calendar but nothing else
  - User authenticates at the auth server; client receives an access token

- **OpenID Connect (OIDC)**: identity layer built on top of OAuth
  - Answers: *who is the authenticated user?*
  - Produces an ID token describing the user's identity
  - Identity Providers: Google, Microsoft Entra, Apple, Facebook, Okta, Auth0, …

- **OIDC handles login**
- **OAuth access tokens handle API authorization**

JSON Web Tokens (JWTs)

- **JWT**: compact, signed format for carrying claims – not a protocol
  - Claims = list of attributes: identity, audience, permissions, …
  - Structure: **header** + **payload** (claims) + **signature**
  - OAuth and OIDC tokens are often encoded as JWTs
- **Local validation**: receiver verifies signature without calling the issuer
  - Fast: no runtime dependency on the identity provider, no need to contact a database
- **Revocation tradeoff**:
  - Local validation means that the token cannot be revoked
  - Token remains valid until expiration even after account suspension
  - Solution: short-lived tokens + refresh token
    - Revocation = no renewal next time
    - Refresh token is used for renewal requests

Architecture and Design

Zero Trust

- **Traditional perimeter model**: anything inside the network = trusted
  - Breaks down with cloud, remote workers, and compromised nodes

- **Zero Trust**: network location should not by itself imply trust
  - Every request must be authenticated and authorized regardless of origin

- **Design under Zero Trust**:
  - **Authenticate every workload**, not just edge users
  - **Encrypt all traffic** – internal as well as external
  - **Short-lived credentials**, with **least privilege access** at every layer
  - **Assume partial compromise**: design to contain blast radius

Least Privilege and Blast Radius

Principle of Least Privilege: a service should have only the permissions it needs

Blast radius: the extent of damage from a single compromised credential

- Over-privileged services cause larger incidents when compromised

- Applies at every layer:
  - Cloud identity & access management: only the resources this service needs
  - Service permissions: only the operations each caller may invoke
  - Database: only the tables and rows this service accesses

- **Micro-segmentation**: fine-grained trust domains between services
  - Reduce the blast radius
  - Compromised service can only reach explicitly permitted destinations

API Gateway and Service Mesh

- **API gateway**: single entry point for external traffic
  - TLS termination, token validation, rate limiting, routing
  - Not responsible for resource-level authorization

- **Service mesh**: infrastructure for service-to-service traffic
  - Sidecar proxy alongside each service intercepts all traffic to/from the service
  - Handles mTLS, workload identity, policy, telemetry automatically
  - Application code does not change – it can be ignorant about secure communications

- API gateway and mesh are complementary
  - External entry vs. internal service traffic

The End