Programming for Automation


TEKsystems
Global Services
WORKFORCE
DEVELOPMENT

Why APIs Need Authentication

- Most real-world APIs are not publicly accessible and require authentication. Authentication allows servers to identify who is making a request.
    - Prevents unauthorized access
    - Enables per-user permissions
    - Allows usage tracking and auditing

© 2025 by Innovation In Software Corporation

What Is an API Key?

- An API key is a long, random value used to authenticate API requests. It is sent by the client with each request to prove identity.
  - Acts like a password for programs
  - Usually generated by the server
  - Must be kept secret

© 2025 by Innovation In Software Corporation

How API Keys Are Sent

- API keys are typically sent in HTTP headers rather than URLs. Headers avoid leaking secrets into logs and browser history.
  - Authorization: Bearer <token>
  - X-API-Key: <token>
  - Avoid query parameters for secrets

© 2025 by Innovation In Software Corporation

API Keys in the Real World

- Most cloud and SaaS platforms rely on API keys for automation access. Keys are designed for scripts, CI/CD, and system-to-system communication.
  - Used by AWS, GitHub, Docker, and others
  - Common in CLI tools and pipelines
  - Often permission-scoped

© 2025 by Innovation In Software Corporation

Security Expectations

- API keys must be treated with the same care as passwords. Poor key handling leads directly to security breaches.
  - Never commit keys to source control
  - Rotate keys regularly
  - Revoke compromised keys immediately

© 2025 by Innovation In Software Corporation

Custom Headers vs Authorization

- APIs commonly accept keys using the Authorization header or a custom header. Both approaches work, but Authorization is more standardized.
  - Authorization: Bearer <key>
  - X-API-Key: <key>
  - Be consistent across endpoints

© 2025 by Innovation In Software Corporation

Why Not Put Keys in URLs

- URLs are logged, cached, and stored in browser history. Placing secrets in URLs increases the risk of accidental exposure.
  - URLs appear in logs
  - Browsers store URL history
  - Headers are safer for secrets

© 2025 by Innovation In Software Corporation

Treat API Keys Like Passwords

- API keys grant access to protected resources. Anyone with the key can act as the associated user or service.
  - Never share keys publicly
  - Rotate keys periodically
  - Revoke keys when compromised

© 2025 by Innovation In Software Corporation

Randomness Matters

- API keys must be difficult to guess. Predictable keys are vulnerable to brute-force attacks.
  - Use secrets module
  - Avoid sequential or short keys
  - Prefer long random values

© 2025 by Innovation In Software Corporation

Key Rotation

- Key rotation limits the damage of leaked credentials. Good systems support replacing keys without downtime.
  - Generate new key
  - Update clients
  - Revoke old key

© 2025 by Innovation In Software Corporation

Key Revocation

- Revocation allows immediate removal of access. This is critical when a key is exposed or abused.
  - Disable or delete key
  - Return 401 for revoked keys
  - Log revocation events

© 2025 by Innovation In Software Corporation

What API Keys Do

- API keys identify the caller and allow access decisions. They enable rate limits, permissions, and auditing.
  - Identify who is calling
  - Control access scope
  - Support monitoring

© 2025 by Innovation In Software Corporation

What API Keys Do Not Do

- API keys do not encrypt network traffic. They rely on HTTPS for confidentiality.
  - No encryption by themselves
  - Visible on the wire without HTTPS
  - Must be combined with TLS

© 2025 by Innovation In Software Corporation

HTTPS Is Still Required

- Without HTTPS, API keys can be intercepted in transit. Encryption is mandatory for any authenticated API.
  - Protects against sniffing
  - Prevents man-in-the-middle attacks
  - Required for secure cookies

© 2025 by Innovation In Software Corporation

Storing API Keys Safely

- Storing raw API keys creates unnecessary risk. Production systems store only derived representations.
  - Avoid plaintext storage
  - Use hashing techniques
  - Limit blast radius of leaks

© 2025 by Innovation In Software Corporation

Why Hash API Keys

- Storing raw API keys creates a single point of failure. Hashing ensures leaked databases cannot be used directly for access.

  - Protects against database leaks
  - Industry best practice
  - Same principle as password storage

© 2025 by Innovation In Software Corporation

What Is a Hash Function

- A hash function converts input data into a fixed-length output. The process is one-way and deterministic.
  - Same input always produces same output
  - Output length is fixed
  - Original value cannot be recovered

© 2025 by Innovation In Software Corporation

One-Way by Design

- Hash functions are intentionally irreversible. This prevents attackers from recovering secrets from stored hashes.
  - No decryption step exists
  - Verification is done by comparison
  - Security relies on math

© 2025 by Innovation In Software Corporation

SHA-256 Overview

- SHA-256 is a cryptographic hash function standardized and widely trusted. It produces a 256-bit output suitable for security applications.
  - Collision resistant
  - Deterministic
  - Fast enough for API keys

© 2025 by Innovation In Software Corporation

Why Python hash() Is Not Secure

- Python’s built-in hash() is designed for hash tables, not security. Its output is intentionally unstable across runs.
  - Changes between executions
  - Not cryptographically secure
  - Never store security data with it

© 2025 by Innovation In Software Corporation

Hashing API Keys in Practice

- When an API key is presented, the server hashes it before comparison. The raw key is never stored or logged.
  - Store only hashes
  - Hash on every request
  - Compare hash values

© 2025 by Innovation In Software Corporation

What Is HMAC

- HMAC combines a hash function with a server-side secret. This prevents offline guessing even if hashes are leaked.
  - Uses a shared secret
  - Secret is not stored in database
  - Stronger than plain hashing

© 2025 by Innovation In Software Corporation

Why Use HMAC for API Keys

- API keys are long and random, making HMAC practical and secure. Attackers cannot validate guesses without the server secret.
  - Protects against brute-force
  - Secret stored separately
  - Industry-recommended approach

© 2025 by Innovation In Software Corporation

HMAC vs Salting

- Salts are stored with hashes, while HMAC secrets are not. This changes the attacker’s threat model significantly.
  - HMAC secret is server-only
  - Salts are public
  - HMAC blocks offline verification

© 2025 by Innovation In Software Corporation

Secure Comparison

- Comparing hashes must be done carefully. Naive string comparison can leak timing information.
  - Use hmac.compare_digest()
  - Avoid == for secrets
  - Prevents timing attacks

© 2025 by Innovation In Software Corporation

Generating API Keys Securely

- API keys must be long, random, and unguessable. Python provides a safe standard library for this purpose.
  - Use secrets.token_hex()
  - Avoid predictable values
  - Generate once, show once

© 2025 by Innovation In Software Corporation

Key Length Matters

- Short keys are easier to brute-force. Long random keys make guessing computationally infeasible.
  - Minimum 128 bits recommended
  - Hex encoding is common
  - Random beats complexity

© 2025 by Innovation In Software Corporation

Where API Keys Live

- Clients store API keys outside of source code. Servers never expose keys after creation.
  - Environment variables
  - Secret managers
  - CI/CD secrets

© 2025 by Innovation In Software Corporation

Authorization Header Flow

- Most APIs expect keys in the Authorization header. This keeps credentials out of URLs and logs.
  - Authorization: Bearer <key>
  - Standardized pattern
  - Works with proxies

© 2025 by Innovation In Software Corporation

Custom API Key Headers

- Some APIs use custom headers for clarity or tooling reasons. Security properties remain the same.
  - X-API-Key header
  - Easier for scripts
  - Still requires HTTPS

© 2025 by Innovation In Software Corporation

Authentication vs Authorization

- Authentication identifies who is calling the API. Authorization determines what they are allowed to do.
  - Key validates identity
  - Role controls access
  - Separate concerns

© 2025 by Innovation In Software Corporation

Correct Error Responses

- Clients rely on HTTP status codes to understand failures. Authentication errors must be explicit and consistent.
  - 401 for missing or invalid keys
  - 403 for insufficient permissions
  - Avoid leaking details

© 2025 by Innovation In Software Corporation

Why Error Messages Matter

- Overly descriptive errors help attackers. Good errors help developers without exposing secrets.
  - Do not reveal valid usernames
  - Do not echo keys
  - Be precise but vague

© 2025 by Innovation In Software Corporation

Logging Authentication Attempts

- Authentication events are security-relevant signals. Logs help detect abuse and misconfiguration.
  - Log failures
  - Track source IP
  - Avoid logging secrets

© 2025 by Innovation In Software Corporation

Lab 1 Summary

- API keys are simple but powerful when handled correctly. Security depends on storage, transport, and validation.
  - Headers not URLs
  - Hash and HMAC
  - HTTPS always

© 2025 by Innovation In Software Corporation

Which header is most commonly used to send API keys?

- A. Cookie
- B. Authorization
- C. Content-Type
- D. Host

© 2025 by Innovation In Software Corporation

Why should API keys not be placed in URLs?

- A. URLs cannot carry headers
- B. URLs are slower
- C. URLs are logged and cached
- D. Browsers block them

© 2025 by Innovation In Software Corporation

What is the main purpose of hashing API keys?

- A. Compress keys
- B. Encrypt traffic
- C. Protect against database leaks
- D. Improve performance

© 2025 by Innovation In Software Corporation

Which Python function is safest for comparing API key hashes?

- A. ==
- B. equals()
- C. hmac.compare_digest()
- D. str.compare()

© 2025 by Innovation In Software Corporation

Which status code should be returned for a missing API key?

- A. 400
- B. 401
- C. 403
- D. 404

© 2025 by Innovation In Software Corporation

Answer: Which header is most commonly used to send API keys?

- A. Cookie
- **B. Authorization**
- C. Content-Type
- D. Host

© 2025 by Innovation In Software Corporation

Answer: Why should API keys not be placed in URLs?

- A. URLs cannot carry headers
- B. URLs are slower
- **C. URLs are logged and cached**
- D. Browsers block them

© 2025 by Innovation In Software Corporation

Answer: What is the main purpose of hashing API keys?

- A. Compress keys
- B. Encrypt traffic
- **C. Protect against database leaks**
- D. Improve performance

© 2025 by Innovation In Software Corporation

Answer: Which Python function is safest for comparing API key hashes?

- A. ==
- B. equals()
- **C. hmac.compare_digest()**
- D. str.compare()

© 2025 by Innovation In Software Corporation

Answer: Which status code should be returned for a missing API key?

- A. 400
- **B. 401**
- C. 403
- D. 404

© 2025 by Innovation In Software Corporation

Lab 2: JWT vs Session Authentication

- This lab compares traditional session-based authentication with modern JWT-based approaches. Both solve the same problem but scale very differently.
  - Authentication model comparison
  - Stateful vs stateless
  - Real-world usage patterns

© 2025 by Innovation In Software Corporation

What Is Authentication State

- Authentication state tracks who a user is across requests. Different systems store this state in different places.
  - Session IDs
  - Cookies
  - Tokens

© 2025 by Innovation In Software Corporation

Session-Based Authentication

- Traditional web apps rely on server-managed session identifiers. The server must remember every active user session.
  - Random session ID
  - Stored server-side
  - Mapped to user

© 2025 by Innovation In Software Corporation

How Sessions Work

- After login, the server creates a session and sends the ID to the client. Every request must be validated against session storage.
  - Session ID in cookie
  - Lookup on each request
  - Stateful design

© 2025 by Innovation In Software Corporation

Session Storage Options

- Sessions can be stored in memory or external systems. Scaling requires shared session storage.
  - In-memory
  - Redis
  - Databases

© 2025 by Innovation In Software Corporation

Problems With Sessions

- Sessions increase server complexity and coupling. Scaling horizontally becomes harder.
  - Sticky sessions
  - Extra infrastructure
  - Operational overhead

© 2025 by Innovation In Software Corporation

Introducing JWT

- JSON Web Tokens embed authentication claims inside the token itself. The server does not store per-user session state.
  - Self-contained token
  - Signed not encrypted
  - Stateless

© 2025 by Innovation In Software Corporation

What JWT Contains

- JWTs carry claims about the user and expiration time. Anyone can read the payload, but only the server can sign it.
  - Username
  - Expiration
  - Signature

© 2025 by Innovation In Software Corporation

JWT Verification Flow

- Each request presents the token to the server. The server validates signature and expiration.
  - No database lookup
  - Signature verification
  - Time-based expiry

© 2025 by Innovation In Software Corporation

JWT vs Sessions Summary

- JWTs trade revocation simplicity for scalability. Sessions trade scalability for control.
  - Stateless vs stateful
  - Scale vs revoke
  - Choose by use case

© 2025 by Innovation In Software Corporation

JWT Issuance Flow

- JWTs are created after successful authentication. They represent a signed statement from the server.
  - Issued at login
  - Signed with secret
  - Contains claims

© 2025 by Innovation In Software Corporation

JWT Claims

- Claims are pieces of information stored inside the token. They describe identity and validity.
  - username
  - expiration (exp)
  - optional permissions

© 2025 by Innovation In Software Corporation

JWT Expiration

- JWTs always include an expiration timestamp. Expired tokens must be rejected.
  - Limits damage if stolen
  - Encourages re-authentication
  - Time-based control

© 2025 by Innovation In Software Corporation

Why Short-Lived Tokens

- JWTs cannot be instantly revoked. Short lifetimes reduce risk.
  - Minutes not days
  - Rotate often
  - Delete cookie on logout

© 2025 by Innovation In Software Corporation

JWT Transport Options

- JWTs can be sent via headers or cookies. Cookies are preferred for browsers.
  - Authorization header
  - HttpOnly cookie
  - SameSite support

© 2025 by Innovation In Software Corporation

HttpOnly Cookies

- HttpOnly cookies cannot be accessed by JavaScript. This protects tokens from XSS.
  - Browser-managed
  - Not readable via JS
  - Safer default

© 2025 by Innovation In Software Corporation

SameSite Cookies

- SameSite controls cross-site cookie behavior. It reduces CSRF risk.
  - Strict
  - Lax
  - None

© 2025 by Innovation In Software Corporation

Secure Cookie Flag

- Secure cookies are only sent over HTTPS. They prevent token leakage on plaintext connections.
  - Requires HTTPS
  - Protects credentials
  - Mandatory in production

© 2025 by Innovation In Software Corporation

JWT Validation Errors

- JWT verification can fail for multiple reasons. Each failure must be handled safely.
  - Missing token
  - Expired token
  - Invalid signature

© 2025 by Innovation In Software Corporation

JWT Logout Behavior

- Logging out deletes the JWT cookie. The token itself remains valid until expiration.
  - Client-side removal
  - No server revocation
  - Expiration enforced

© 2025 by Innovation In Software Corporation

JWT Is Not Encryption

- JWT payloads are base64 encoded, not encrypted. Anyone can read the contents if they have the token.
  - Readable payload
  - Signature enforces trust
  - Do not store secrets inside

© 2025 by Innovation In Software Corporation

JWT Signature Purpose

- The signature proves the token was created by the server. Clients cannot forge valid tokens.
  - Server-only secret
  - Tamper detection
  - Trust boundary

© 2025 by Innovation In Software Corporation

What Happens If a JWT Is Stolen

- A stolen JWT is valid until it expires. This is why short lifetimes are critical.
  - No instant revocation
  - Expiration limits damage
  - Logout deletes cookie only

© 2025 by Innovation In Software Corporation

CSRF and JWT Cookies

- Cookies are sent automatically by browsers. CSRF protections are still required.
  - SameSite cookies
  - CSRF tokens
  - Origin checks

© 2025 by Innovation In Software Corporation

JWT in Authorization Headers

- APIs sometimes send JWTs via headers instead of cookies. This is common for mobile and service clients.
  - Authorization: Bearer
  - Manual attachment
  - Different threat model

© 2025 by Innovation In Software Corporation

JWT for Browsers vs APIs

- Browser-based apps and API clients have different needs. Transport choice matters.
  - Cookies for browsers
  - Headers for services
  - Security tradeoffs

© 2025 by Innovation In Software Corporation

Stateless Scaling Benefits

- JWTs allow horizontal scaling without shared session stores. Any server can validate any token.
  - No Redis needed
  - No sticky sessions
  - Simpler infrastructure

© 2025 by Innovation In Software Corporation

JWT Operational Tradeoffs

- Stateless design moves complexity elsewhere. Expiration and key rotation become critical.
  - Rotate secrets carefully
  - Short expirations
  - Plan revocation strategy

© 2025 by Innovation In Software Corporation

JWT Rotation

- JWT signing secrets must be rotated periodically. Old tokens become invalid when secrets change.
  - Key rotation events
  - Forced logout
  - Operational planning

© 2025 by Innovation In Software Corporation

Lab 2 Summary

- JWTs replace server-side sessions with signed tokens. They scale well but require careful security design.
  - Stateless auth
  - Short-lived tokens
  - Cookie security

© 2025 by Innovation In Software Corporation

What makes JWT authentication stateless?

- A. Cookies store sessions
- B. Tokens store claims
- C. Server stores sessions
- D. Redis is optional

© 2025 by Innovation In Software Corporation

Why are JWTs usually short-lived?

- A. They are encrypted
- B. They are cached
- C. They cannot be revoked
- D. They are compressed

© 2025 by Innovation In Software Corporation

Which cookie flag prevents JavaScript access?

- A. Secure
- B. SameSite
- C. HttpOnly
- D. Path

© 2025 by Innovation In Software Corporation

What happens when a JWT signing secret is rotated?

- A. Tokens are refreshed
- B. Old tokens become invalid
- C. Cookies persist
- D. Sessions migrate

© 2025 by Innovation In Software Corporation

Which risk still exists when using JWT cookies?

- A. SQL injection
- B. XSS
- C. CSRF
- D. Brute force

© 2025 by Innovation In Software Corporation

Answer: What makes JWT authentication stateless?

- A. Cookies store sessions
- **B. Tokens store claims**
- C. Server stores sessions
- D. Redis is optional

© 2025 by Innovation In Software Corporation

Answer: Why are JWTs usually short-lived?

– A. They are encrypted
– B. They are cached
– **C. They cannot be revoked**
– D. They are compressed

© 2025 by Innovation In Software Corporation

Answer: Which cookie flag prevents JavaScript access?

- A. Secure
- B. SameSite
- **C. HttpOnly**
- D. Path

© 2025 by Innovation In Software Corporation

Answer: What happens when a JWT signing secret is rotated?

- A. Tokens are refreshed
- **B. Old tokens become invalid**
- C. Cookies persist
- D. Sessions migrate

© 2025 by Innovation In Software Corporation

Answer: Which risk still exists when using JWT cookies?

- A. SQL injection
- B. XSS
- **C. CSRF**
- D. Brute force

© 2025 by Innovation In Software Corporation

Lab 3: Securing APIs with HTTPS

- HTTPS encrypts data between clients and servers. It is mandatory for protecting credentials and tokens.
    - Encryption
    - Authentication
    - Integrity

© 2025 by Innovation In Software Corporation

What Is HTTPS

- HTTPS is HTTP layered on top of TLS. It prevents attackers from reading or modifying traffic.
  - Encrypted transport
  - Identity verification
  - Tamper resistance

© 2025 by Innovation In Software Corporation

Why HTTP Is Dangerous

- Plain HTTP sends data in cleartext. Anyone on the network can intercept credentials.
  - Passwords exposed
  - Tokens leaked
  - Man-in-the-middle attacks

© 2025 by Innovation In Software Corporation

TLS in Simple Terms

- TLS creates a secure tunnel before any data is exchanged. All application data flows through this encrypted channel.
  - Handshake
  - Key exchange
  - Secure session

© 2025 by Innovation In Software Corporation

Public and Private Keys

- TLS relies on asymmetric cryptography. Public keys are shared while private keys remain secret.
  - Public key advertised
  - Private key protected
  - Trust model

© 2025 by Innovation In Software Corporation

Certificates

- Certificates bind a public key to a server identity. Clients trust certificates signed by known authorities.
  - Contains public key
  - Signed by CA
  - Proves identity

© 2025 by Innovation In Software Corporation

Digital Signatures

- Servers prove ownership of private keys during TLS handshakes. Clients verify signatures using public keys.
  - Authentication
  - Integrity
  - Non-repudiation

© 2025 by Innovation In Software Corporation

Encryption After Handshake

- Once TLS is established, symmetric encryption is used. This keeps communication fast and secure.
  - Session keys
  - Efficient encryption
  - Secure channel

© 2025 by Innovation In Software Corporation

HTTPS and Cookies

- Secure cookies are only sent over HTTPS. Authentication cookies require HTTPS.
  - secure flag
  - HttpOnly flag
  - SameSite flag

© 2025 by Innovation In Software Corporation

Lab 3 Scope

- This lab focuses on enabling HTTPS locally and understanding TLS. Production certificate management is out of scope.
  - Local development
  - Self-signed certs
  - Conceptual grounding

© 2025 by Innovation In Software Corporation

Self-Signed Certificates

- Local development often uses self-signed certificates. These provide encryption but not public trust.
  - Encrypted traffic
  - Browser warnings expected
  - Dev-only usage

© 2025 by Innovation In Software Corporation

Why Browsers Warn

- Browsers trust certificates signed by known authorities. Self-signed certificates cannot be verified.
  - No trusted CA
  - Manual acceptance required
  - Expected behavior

© 2025 by Innovation In Software Corporation

OpenSSL Overview

- OpenSSL is a command-line tool for cryptographic operations. It is commonly used to generate TLS certificates.
  - Key generation
  - Certificate creation
  - Industry standard

© 2025 by Innovation In Software Corporation

Generating a Private Key

- TLS starts with generating a private key. This key must be kept secret.
  - openssl genrsa
  - 2048-bit minimum
  - Never share

© 2025 by Innovation In Software Corporation

Creating a Certificate

- Certificates bind a public key to an identity. Self-signed certs are sufficient for learning.
  - Certificate signing request
  - Self-signing
  - Expiration dates

© 2025 by Innovation In Software Corporation

Certificate Files

- Flask requires both a certificate and a private key. These files enable HTTPS locally.
  - cert.pem
  - key.pem
  - Filesystem permissions

© 2025 by Innovation In Software Corporation

Flask HTTPS Configuration

- Flask can serve HTTPS using an SSL context. This enables encrypted local testing.
  - ssl_context parameter
  - cert and key paths
  - Debug mode supported

© 2025 by Innovation In Software Corporation

Secure Cookies with HTTPS

- Cookies marked Secure are only sent over HTTPS. Browsers enforce this automatically.
  - secure=True
  - Required for auth cookies
  - HTTPS-only transport

© 2025 by Innovation In Software Corporation

Testing HTTPS with curl

- curl can test HTTPS endpoints locally. Self-signed certs require explicit trust bypass.
  - curl -k
  - Dev-only flag
  - Never use in production

© 2025 by Innovation In Software Corporation

Lab 3 Summary

- HTTPS protects credentials and tokens in transit. Authentication is unsafe without encryption.
  - TLS encryption
  - Secure cookies
  - Production requirement

© 2025 by Innovation In Software Corporation

What does HTTPS primarily protect?

- A. Application logic
- B. Data in transit
- C. Server uptime
- D. Database encryption

© 2025 by Innovation In Software Corporation

Why are self-signed certificates acceptable in labs?

- A. They are faster
- B. They provide encryption
- C. Browsers trust them
- D. They scale better

© 2025 by Innovation In Software Corporation

Which file must remain secret on the server?

- A. cert.pem
- B. csr.pem
- C. key.pem
- D. openssl.conf

© 2025 by Innovation In Software Corporation

Why will browsers refuse Secure cookies over HTTP?

- A. Performance reasons
- B. Missing headers
- C. Encryption required
- D. Cookie size limits

© 2025 by Innovation In Software Corporation

What does curl -k do?

- A. Encrypts traffic
- B. Enables cookies
- C. Ignores cert trust errors
- D. Generates certificates

© 2025 by Innovation In Software Corporation

Answer: What does HTTPS primarily protect?

- A. Application logic
- **B. Data in transit**
- C. Server uptime
- D. Database encryption

© 2025 by Innovation In Software Corporation

Answer: Why are self-signed certificates acceptable in labs?

- A. They are faster
- **B. They provide encryption**
- C. Browsers trust them
- D. They scale better

© 2025 by Innovation In Software Corporation

Answer: Which file must remain secret on the server?

- A. cert.pem
- B. csr.pem
- **C. key.pem**
- D. openssl.conf

© 2025 by Innovation In Software Corporation

Answer: Why will browsers refuse Secure cookies over HTTP?

- A. Performance reasons
- B. Missing headers
- **C. Encryption required**
- D. Cookie size limits

© 2025 by Innovation In Software Corporation

Answer: What does curl -k do?

- A. Encrypts traffic
- B. Enables cookies
- **C. Ignores cert trust errors**
- D. Generates certificates

© 2025 by Innovation In Software Corporation