## API Security Best Practices Report

## Week 6 LAB – API Gateway Security

## Tester: Priyadharshini Rengaramanujam

## Environment: Postman, Amzon AWS, Amazon API Gateway, Amazon Cognito, AWS Lambda

## Report Date: 01/17/2026

**Format:** Discussed the API Security Best Practices learnt from APISec course API Gateway Security Best Practices and attached documents of course completion. Created API gateway using AWS cloud provider and connected to backend API. Applied strong authentication such as OAuth2, OIDC, JWT and basic security controls such as HTTPS only, Rate Limiting and Access Controls. Attached the screenshots the setup created for API Security Best Practices and implemented fixes for OWASP API Top 10 vulnerabilities.

**Executive Summary:**

API Security – Securing APIs and external APIs, using API specific security strategies. Security secures API vulnerabilities or misconfigurations and prevent exploitation by attackers.

**Benefits of API security**:

- Better developers experience
- Greater room for innovation
- Increased standardization of API program
- Better trust and assurance built with customers and community
- Less room for breaches, hackers and issues.

**API Gateway** – It acts as an intermediary between client applications and backend services within microservices architecture. Software layers consolidate multiple API into single endpoint. Provide centralized control allowing developers to focus on building individual services.

**Traffic management and routing** - API Gateways distribute incoming requests to microservices based on predefined rules and conditions.

**Benefits:**

- Application resources used efficiently
- Prevent bottlenecks and provides load balancing capabilities
- Improves reliability and scalability
- Enhance user experience and scaling

**Authentication** – verifies user or application identity, safeguarding against unauthorized access and enables accountability. Safeguards systems and information against unwanted access, data breaches, hacks and mistakes. Secure ecosystem to safeguard the integrity and confidentiality of transmitted data. Clear boundaries between sensitive user data, authentication credentials and valuable resources.

**Authorization** – authenticated users only access permitted resources and actions, protecting data integrity and enforcing business rules.

**Benefits**:

- Essential for preventing breaches, complying with regulations and maintaining user's trust making integral to success and reliability of applications.

**Rate limiting** – services remain protected and operate within intended capacity limits.

**Benefits**:

- Enhances stability and reliability of applications
- Fair resource allocation among clients, positive user experience and efficient resource utilization.

**Kubernetes** – dynamic scaling of containers and microservices to handle varying workloads.

**Logging and monitoring** – API traffic and errors

Track active requests and traffic patterns, scaling needs and resource allocations are handled. Fluctuations without performance degradation and main key topics include Latency, Traffic, Errors and Saturation.

**Types of API Gateways**:

**Generic use** – Kong, Gloo, Granitee, Apigee, Tyk

- Pros – full suite offerings, widely used, more versatile
- Cons – less scalable, more expensive, more plugins needed, more manual configuration

**Kubernetes specific options** – EdgeStack, Traefik, Istio

- Pros – more scalable, Kubernetes friendly, less plugins
- Cons – multiple environments or migrate away from Kubernetes, integrating Kubernetes-specific API with non-Kubernetes components or external systems, more challenging than generic solutions.

**Open-source options** – Emissary Ingress, Kong Gateway OSS, Tyk OSS, Kraken D, Grasitee OSS

- Pros – free, no additional security and auth built out, customizable potential
- Cons – not heavily maintained or strongly supported, limited capabilities and features, lots of manual configuration.

**Tips for better API Gateways**:

- Use HTTPS communication
- Leverage serverless functions
- Use centralized authentication server
- Limit requests and maintain regularly
- Implement monitoring and analytics
- Implement API-based connectivity
- Manage deprecated APIs

**Benefits:**

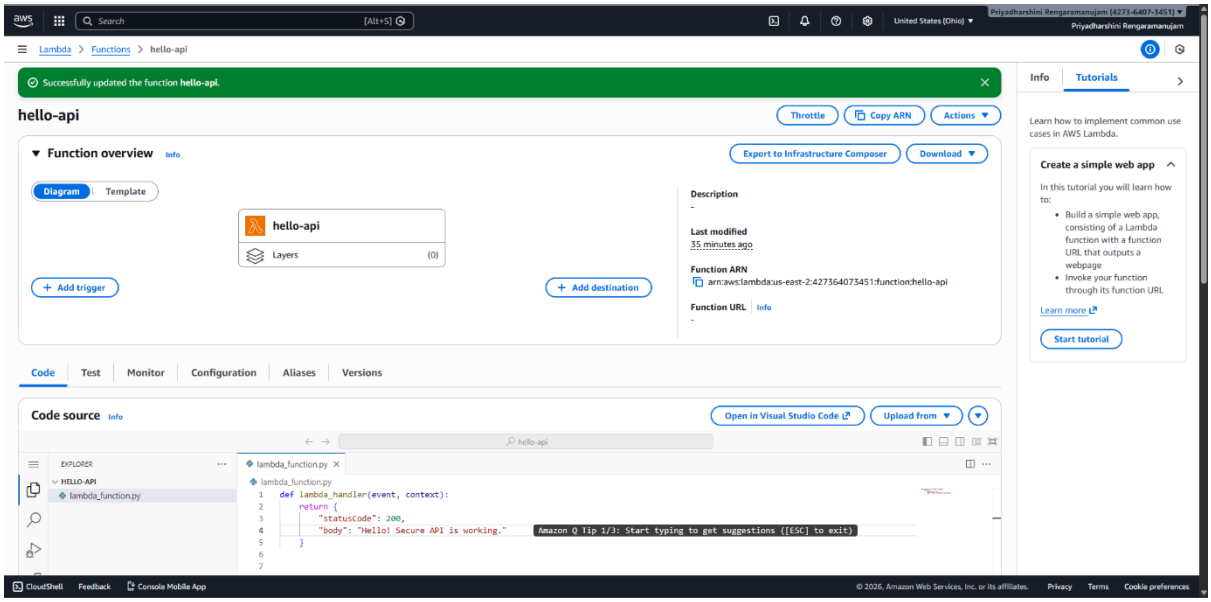
- Improved security
- Better standardization and centralization
- Enhanced developer experience
- More reliability

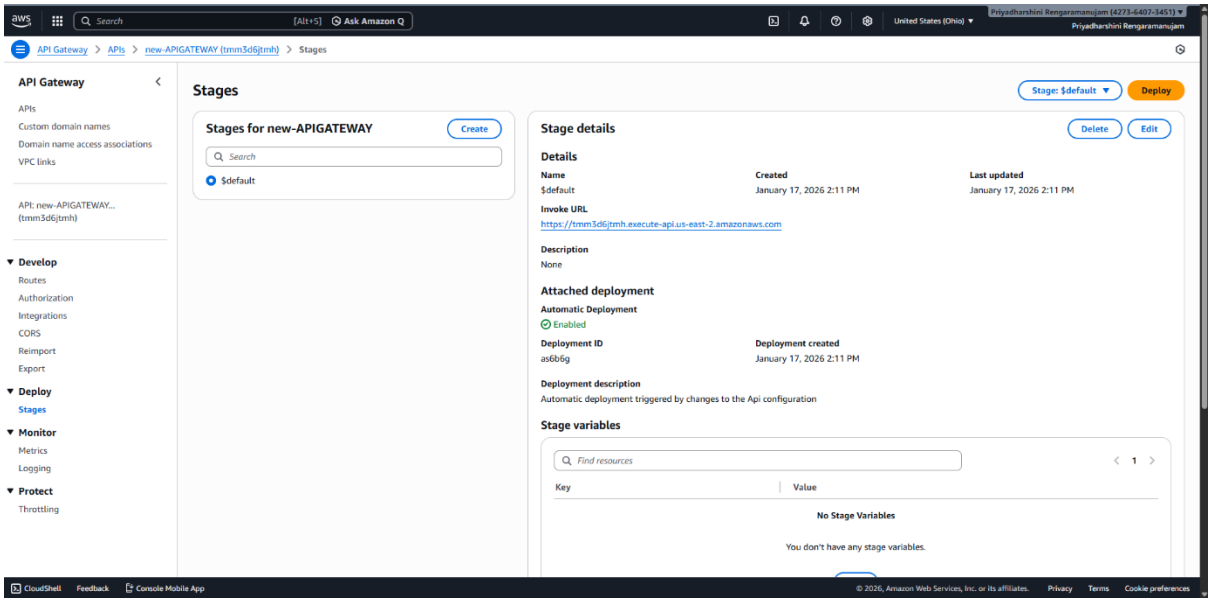

**LAB Report:**

This lab was implemented using AWS and services used – Amazon API Gateway to create an API Gateway endpoint and protect routes with authentication/authorization. Amazon Cognito to provide OAuth 2.0 / OpenID Connect authentication and issue JWT tokens. API Gateway route /hello is integrated to a backend endpoint configured in API Gateway Integrations.

The integration forwards requests that pass authentication to the backend and returns the backend response.

The invoked URL https://tmm3d6jtmh.execute-api.us-east-2.amazonaws.com



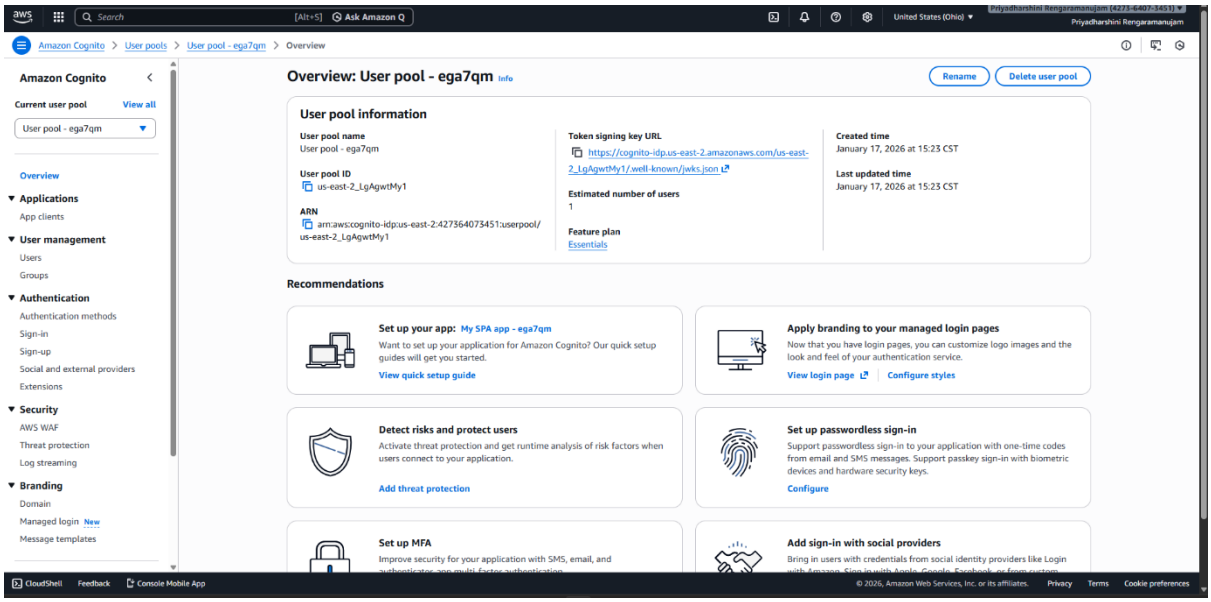


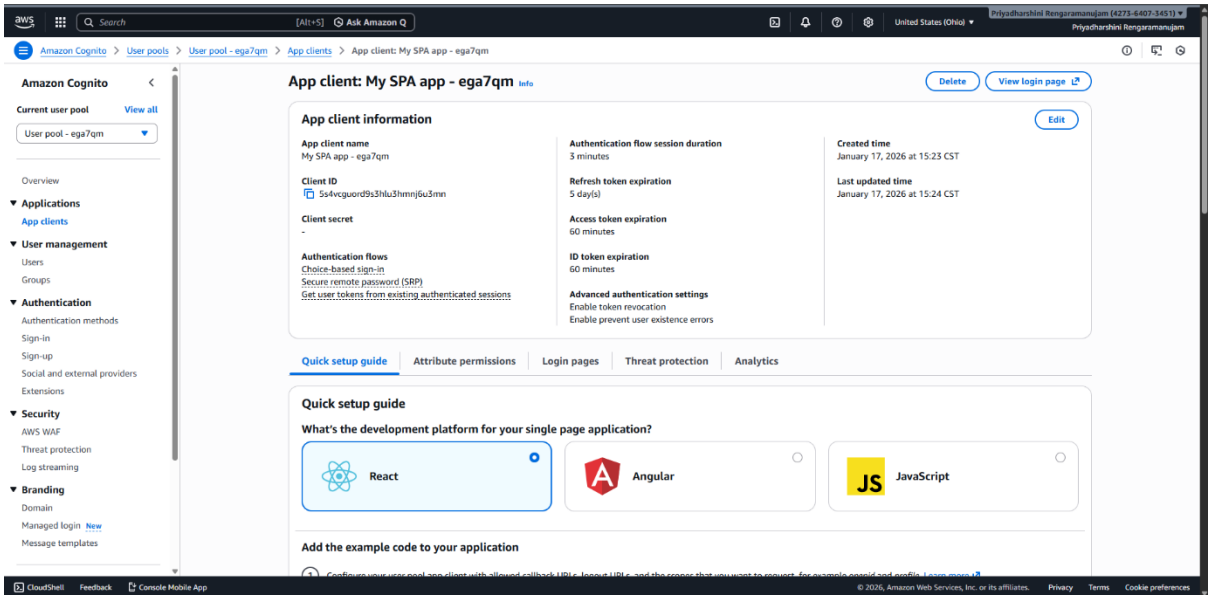

Authentication and authorization were implemented using Amazon Cognito User Pools as an OpenID Connect provider with OAuth 2.0 Authorization Code Grant. This approach provides strong authentication and issues JWT tokens that are verified by API Gateway.

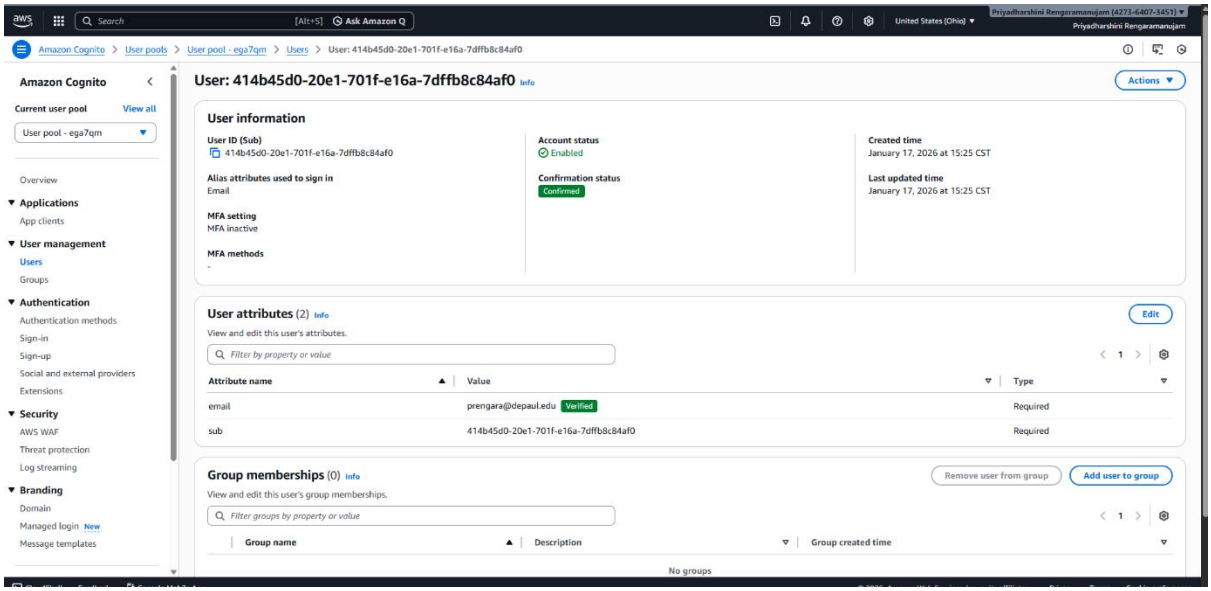
A Cognito User Pool was created to manage users and authentication. A test user account was created in the pool and used during login or token generation process.

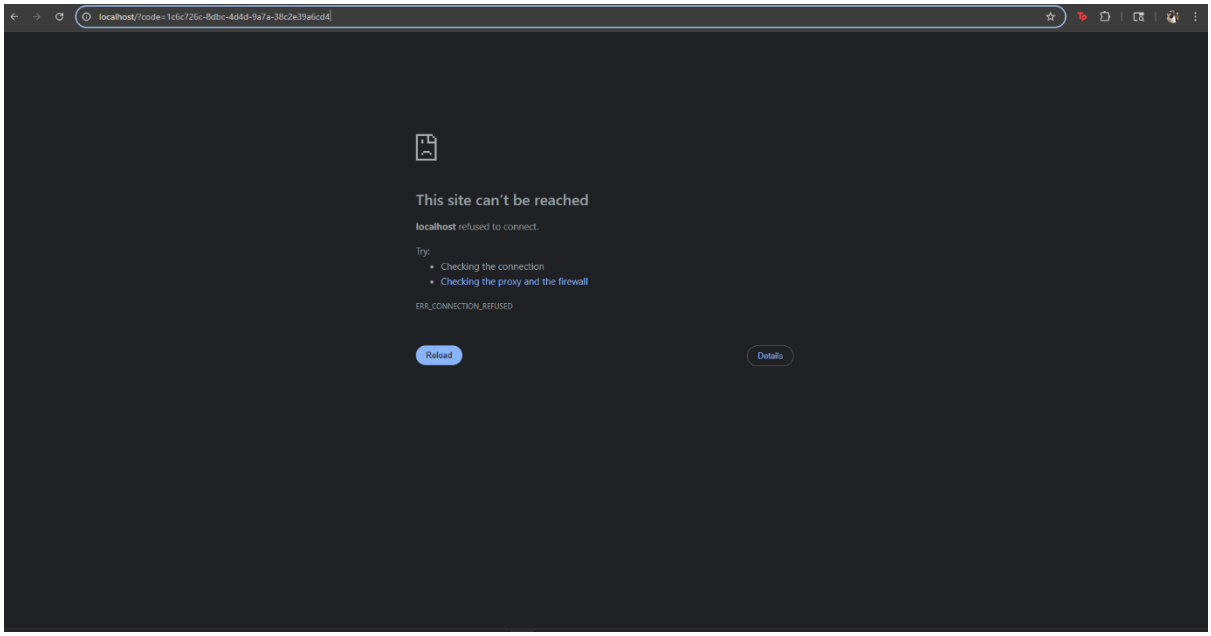
The user signs in using Cognito's hosted or managed login experience, Cognito returns an authorization code to redirect URI as token endpoint: /oauth2/token

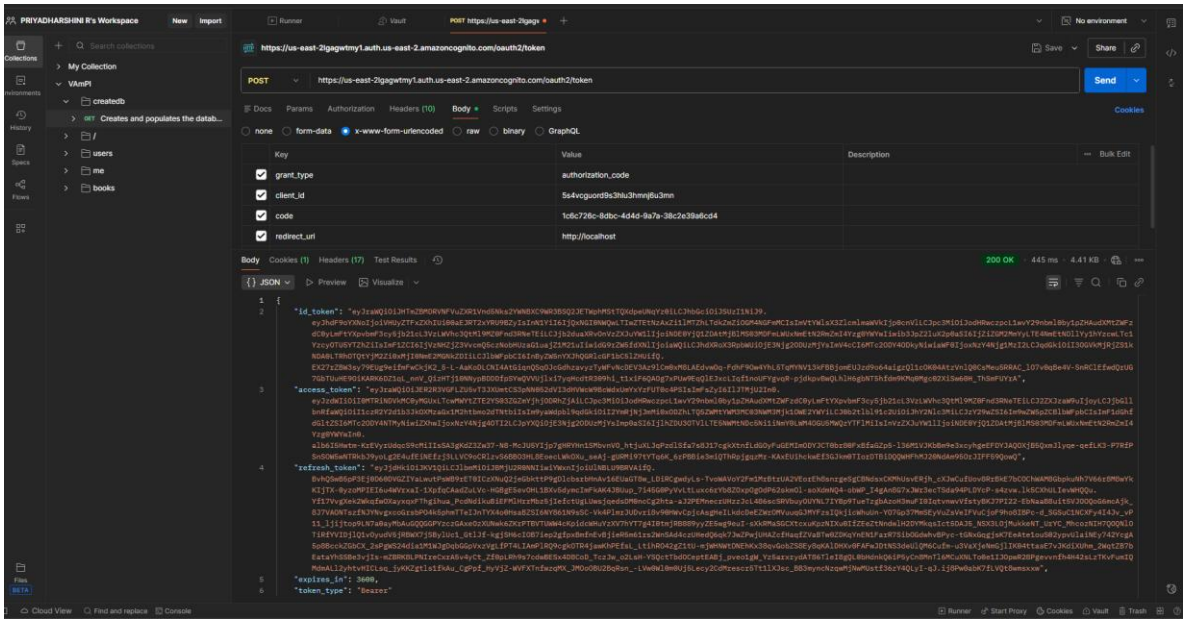
Cognito returns JWT tokens: access_token, id_token, refresh_token



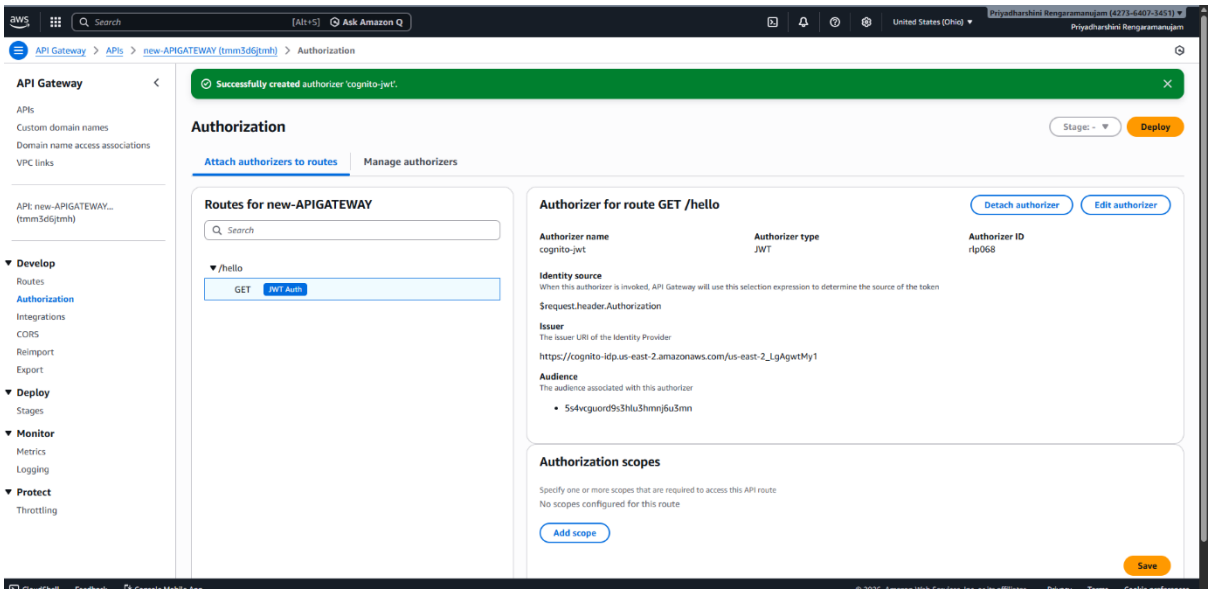

Amazon Cognito
App client: My SPA app - ega7qm
Delete
View login page
App client information
Edit
App client name
My SPA app - ega7qm
Client ID
5s4vcguord9s3hlu3hmnj6u3mn
Client secret
-
Authentication flows
Choice-based sign-in
Secure remote password (SRP)
Get user tokens from existing authenticated sessions
Authentication flow session duration
3 minutes
Refresh token expiration
5 day(s)
Access token expiration
60 minutes
ID token expiration
60 minutes
Advanced authentication settings
Enable token revocation
Enable prevent user existence errors
Created time
January 17, 2026 at 15:23 CST
Last updated time
January 17, 2026 at 15:24 CST
Quick setup guide
Attribute permissions
Login pages
Threat protection
Analytics
Quick setup guide
What's the development platform for your single page application?
React
Angular
JavaScript
Add the example code to your application


Amazon Cognito
User: 414b45d0-20e1-701f-e16a-7dffb8c84af0
Actions
User information
User ID (Sub)
414b45d0-20e1-701f-e16a-7dffb8c84af0
Alias attributes used to sign in
Email
MFA setting
MFA inactive
MFA methods
-
Account status
Enabled
Confirmation status
Confirmed
Created time
January 17, 2026 at 15:25 CST
Last updated time
January 17, 2026 at 15:25 CST
User attributes (2)
Edit
View and edit this user's attributes.
Attribute name
Value
Type
email
prengara@depaul.edu
Verified
Required
sub
414b45d0-20e1-701f-e16a-7dffb8c84af0
Required
Group memberships (0)
Remove user from group
Add user to group
View and edit this user's group memberships.
Group name
Description
Group created time
No groups


localhost/?code=1c6c726c-8dbc-4d4d-9a7a-38c2e39a6cd4
This site can't be reached
localhost refused to connect.
Try:
Checking the connection
Checking the proxy and the firewall
ERR_CONNECTION_REFUSED
Reload
Details



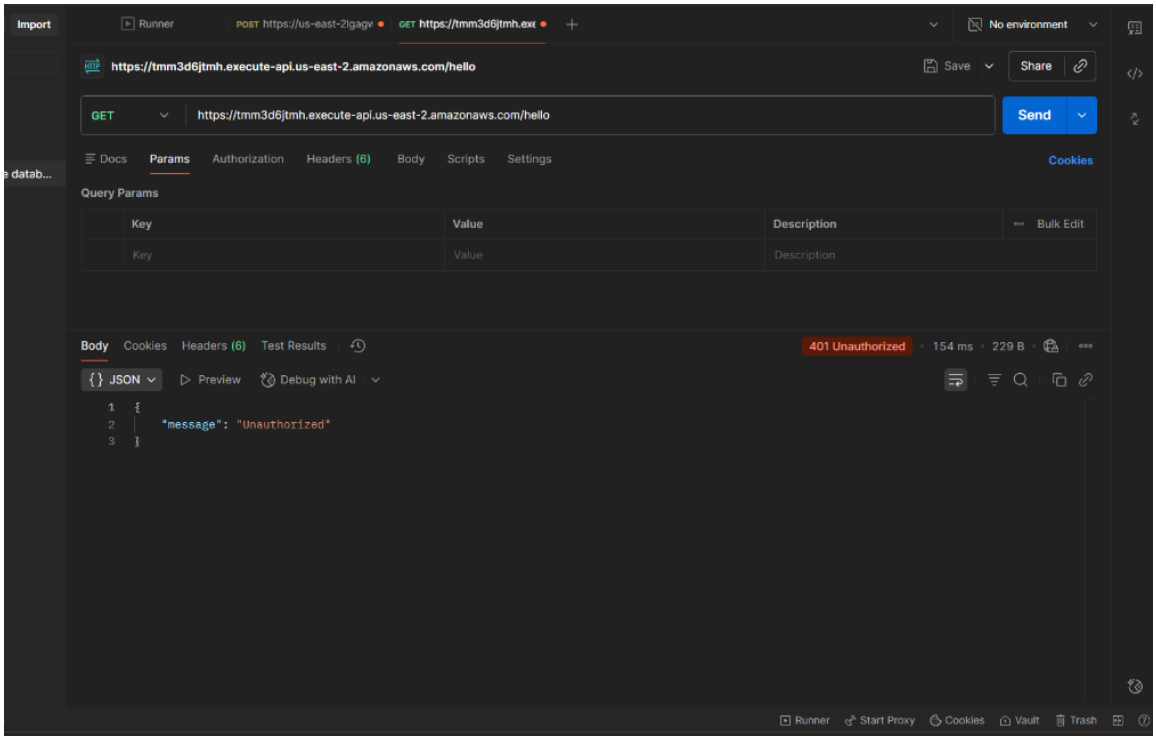

In API Gateway, JWT Authorizer was created and attached to the GET /hello route. This authorizer enforces token validation before requests can reach the backend. It was configured with Identity Source reading the Bearer token from the Authorization header. When a request is sent to GET /hello, API Gateway checks the token signature and claims. If the JWT is missing or invalid the request is blocked and the backend is not called. Only requests with a valid Cognito issued JWT are allowed.

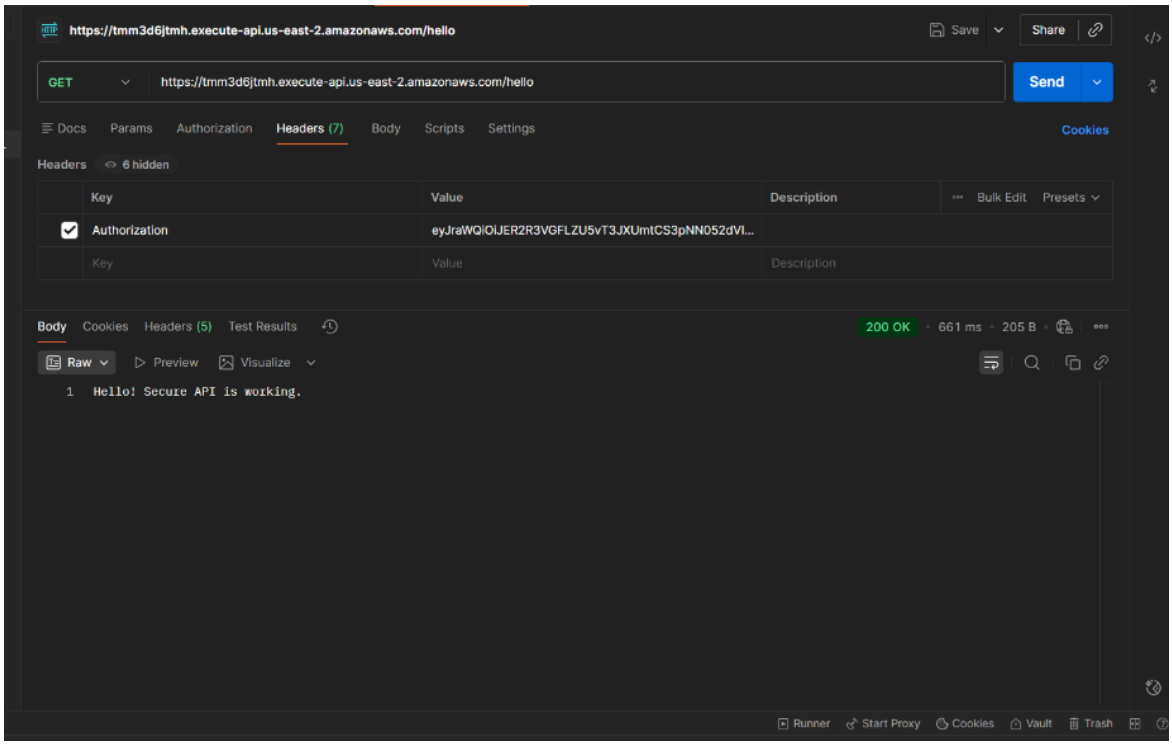



HTTPS is enforced by design in this deployment and API is exposed using the AWS-managed HTTPS Invoke URL, and all traffic between clients and API Gateway is encrypted using TLS.

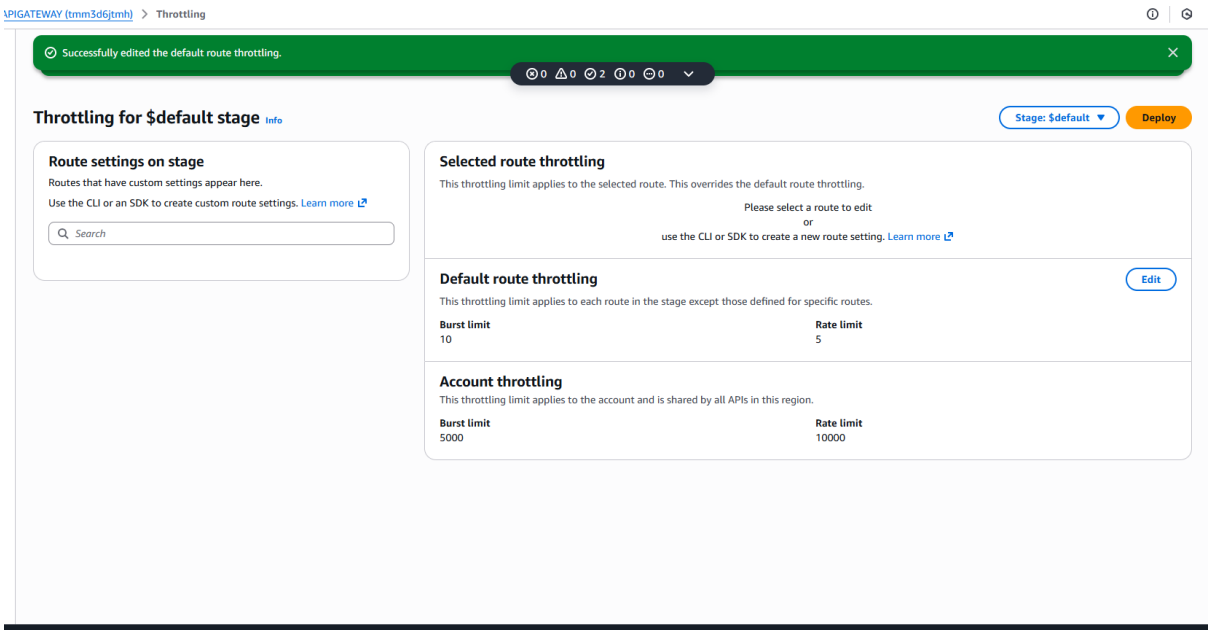
Strong authentication was implemented using OAuth 2.0 Authorization Code Flow, OpenID Connect Scopes, JWT tokens validated at API Gateway via JWT authorizer. This is stronger than basic auth or API keys because it supports standardized identity flows, token expiration, and cryptographically signed tokens.

Access control is enforced by API Gateway using JWT Authorizer, requests without an authorization header are denied, only authenticated users holding a valid JWT access token issued by Cognito can access the protected route.




Rate limiting was implemented using API Gateway Throttling, this protects the API from excessive requests and basic abuse scenarios. Default route throttling is configured with a rate limit and burst limit which ensures the API rejects or limits traffic that exceeds configured thresholds. The following screenshots were collected as evidence that the gateway, authentication and security controls are working correctly.




As part of this lab, previously identified and exploited API security weaknesses were mitigated by applying security best practices aligned with the OWASP API Security Top 10.

**API1: Broken Object Level Authorization (BOLA)**

Risk – attackers can access or modify objects belonging to other users by manipulating objects identifiers in API requests.

Fix implemented – authentication was enforced at the API Gateway layer using JWT-based authorization. Each API request now requires a valid access token, ensuring only authenticated users can access protected endpoints.

**API2: Broken Authentication**

Risk – weak or missing authentication allows attackers to impersonate users or gain unauthorized access.

Fix implemented – authentication was fully delegated to Amazon Cognito using OAuth 2.0 Authorization Code Grant, OpenID Connect, JWT access tokens with expiration and signature validation. API Gateway validates token using a JWT Authorizer, preventing requests with invalid or expired tokens.

**API3: Broken Object Property Level Authorization (BOPLA)**

Risk – attackers can modify sensitive fields in request bodies that should not be user-controllable.

Fix implemented – the API now only accepts requests from authenticated users. Sensitive attributes are not trusted from client input and should be validated or enforced at the backend using identity claims from the JWT.

**API4: Unrestricted Resource Consumption**

Risk – APIs without rate limiting can be abused for denial-of-service attacks or resource exhaustion.

Fix implemented – API Gateway  throttling has rate limits that controls the number of requests per second and burst limits control traffic spikes.

**API5: Broken Function Level Authorization (BFLA)**

Risk – users can access privileged API functions that should be restricted.

Fix implemented – API routes are protected using a JWT Authorizer, ensuring only authenticated users can invoke protected routes. API Gateway enforces authorization before forwarding requests to the backend.

**API6: Unrestricted Access to Sensitive Business Flows**

Risk – attackers automate or abuse sensitive API flows

Fix implemented – rate limiting via API Gateway Throttling restricts how frequently sensitive endpoints can be accessed.

**API7: Server-Side Request Forgery  (SSRF)**

Risk – attackers trick the API into making unauthorized internal request.

Fix implemented – all external access to backend services in mediated through API Gateway. Direct access to backend resources is restricted, and only validated requests from API Gateway are forwarded.

**API8: Security Misconfiguration**

Risk – misconfigured APIs expose unnecessary functionality or sensitive information.

Fix implemented – only required routes were exposed, authentication was enforced by default using AWS-managed TLS, no sensitive credentials were embedded in the API.

**API9: Improper Inventory Management**

Risk – undocumented or forgotten API versions increase attack surface.

Fix implemented – API Gateway provides centralized API management, allowing all routes, stages, and authorizers to be inventoried and managed in one place.

**API10: Unsafe Consumption of APIs**

Risk – APIs trust external services without validation.

Fix implemented – authentication and authorization are enforced at API Gateway before requests reach backend integrations. Only validated, authenticated requests are allowed to invoke services.

**Summary:**

This lab successfully implemented a secure API Gateway using AWS, An HTTP API was created in API Gateway, connected to a backend route, and protected using strong authentication. API Gateway validates JWT access tokens using a JWT Authorizer, HTTPS encryption is enforced using AWS-managed invoke URLs. Rate limiting was applied using API Gateway throttling, and access control was enforced by requiring valid tokens for the protected /hello endpoint.