Kong

TECHNICAL GUIDE

Secure Your Web, Mobile Applications and APIs using the Kong Gateway

Krishnaraj Subburayalu
Senior Technical Account Manager
Kong Inc.

Content

API Strategy: Risk and Rewards

Today, Application Programming Interfaces (APIs) and microservices are the engines powering the digital economy. The benefits of APIs and microservices in such a rapidly evolving world are manifold; at a top level, they help accelerate time-to-market, enhance customer experience and increase the speed of innovation.

However, the complexity and highly distributed nature of these modern web and mobile applications introduce new challenges, new attack vectors and require a new approach to security. Without proper security, enterprises may accidentally expose sensitive data or open themselves up to cyberattacks, compliance violations and other security issues. Enterprises need to have a comprehensive API management (APIM) strategy for securing their APIs and microservices. APIM is a broad category that includes an API gateway and supporting capabilities such as a developer portal, security, observability/analytics, API lifecycle management and more.

Benefits of developing robust API strategies

- Accelerate time-to-market
- Enhance Customer Experience
- Increase the Velocity of Innovation
- Improve Operational Efficiency

Without proper security, organizations run the risk of

- Losing Customer Trust and Confidence
- Cyberattacks
- Compliance Violations and Fines
- Slowing Speed of Application Development

Authentication and Authorization using an API Gateway

An API gateway decouples the upstream microservices from your applications, providing centralized traffic routing, integration and security policy for all API traffic. While this simplifies access for client applications, it also provides a centralized platform for implementing and enforcing policies, including security policies consistently to all your APIs.

The API gateway can be configured to enable security policies such as authentication and authorization. Authentication is the verification that somebody is who they say they are. Beyond the basic login steps of entering a username and password, other means of facilitating authentication include OAuth2.0, OpenID Connect, mutual TLS and token-based authentication. Authorization is the determination of what resources somebody is allowed to access. Open Policy Agent (OPA), an open-source authorization engine, has become increasingly popular to apply fine-grained authorization to APIs and microservices.

A best practice for authenticating API consumers is token-based authentication and authorization, where users or applications get tokens from an Identity Provider (IdP) and send tokens to the service/API. The service/API validates the token with the IdP and allows access. These tokens are usually time-bound and expire within a time limit and are revocable. This identity tokens exchange provides greater security of not sending passwords/credentials often over the network, reducing the risk of identity theft. The token-based approach to authentication allows separating the issuing of tokens from their validation, thus facilitating the centralization of identity management.

With the benefit of centralized identity management, all applications (web, mobile, legacy) and upstream APIs use the same IdP to manage API consumer identities. Adopting a centralized identity management strategy enables architects to implement consistent security best practices and standards across the organization as they can easily define and manage access controls and the consumers across all the systems, including their applications and API gateways. This strategy also helps developers to focus their efforts on application design and feature development, not on writing redundant code to integrate the IdP with each application.

Like centralized identity management, validation of a token and its authentication management can be centralized or delegated to a modern API gateway like Kong. A legacy gateway approach would use the IdP for authentication and gateway to define authorization per endpoint to the groups you want to grant access to the backend services. This approach increases management overhead as administrators have to dedicatedly manage users and maintain group memberships in the gateway to grant or revoke permissions. With Kong Gateway, management of keys, tokens and users happen in the IdP versus the gateway removing the need to manage a separate silo of identity.

This guide will walk through how the Kong Gateway can secure and protect access to applications and APIs in a unified way.

Kong Authentication and Authorization Capabilities

Kong Gateway is a lightweight API Gateway that lets you secure, manage, and extend APIs and microservices across hybrid or multi-cloud infrastructure. Kong Gateway is available as an on-premise or private cloud and with Konnect, our SaaS platform. The power and flexibility of the Kong API gateway comes through its plugins that integrate seamlessly with your deployments. There are multiple Kong plugins available to handle authentication, request transformations, rate limiting and more. Check out Kong's Plugins Hub for more information.

In this document, we will be focusing on Kong's authentication plugin - OpenID Connect (OIDC). Kong supports integration with federated identity management through the OIDC plugin. The OIDC plugin supports several types of grants/credentials such as opaque access tokens, refresh tokens, authorization code, session cookies, client credentials and more. It also supports several OIDC identity management providers such as Okta, Keycloak, PingFederate, Azure Active Directory, Microsoft Active Directory and more.

Secure Applications and APIs with Kong Gateway and an IdP

The Kong Gateway authenticates the applications and users using the IdP and maintains the session. The best practice is that access tokens and refresh tokens are never exposed to browsers but only the session cookies that Kong Gateway generates. Sometimes mobile applications cannot handle the session cookies and make use of refresh tokens directly. These refresh tokens (with offline access scope) could be short or long-lived based on the organization's security requirement. The session validity time is configurable both in the IdP and Kong Gateway.

Authentication takes place in two distinct phases:

1. **Login Workflow:** Used by the applications to authenticate the end-users.
2. **API Access flow:** Second phase of the flow, where applications consume the API to retrieve/update relevant information.

Login Workflow

In this workflow, the application delegates user authentication to Kong Gateway. The below diagram describes the login workflow.

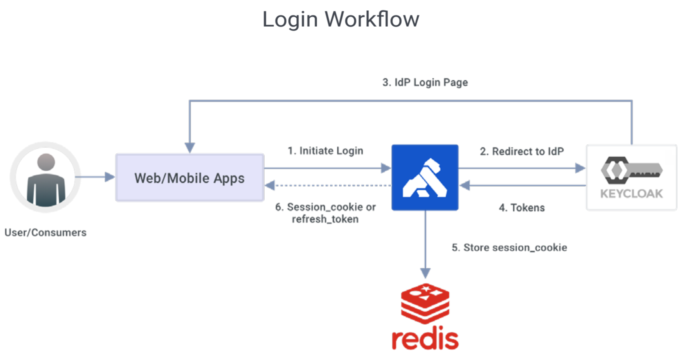

Diagram 1: Login Workflow

1. The user goes to the application's login page, which is a proxy endpoint hosted in the Kong Gateway.
2. The Kong Gateway redirects to the Keycloak IdP.
3. IdP redirects the user to its own login page, where user submit their credentials.
4. The IdP validates the user credentials and sends an authorization code which is used by Kong Gateway to get the access tokens. The gateway uses an "offline access" scope to get refresh token along with access tokens.
5. Kong Gateway creates a session cookie and stores it in the Redis cache.
6. Kong Gateway responds to applications with the session cookie or refresh token based on the configuration. The application will then use the session cookie or refresh token to access the APIs as described in the API access flow below.

API Access flow

In this workflow, the application delegates user authentication to Kong Gateway. The below diagram describes the login workflow.

API Access Workflow

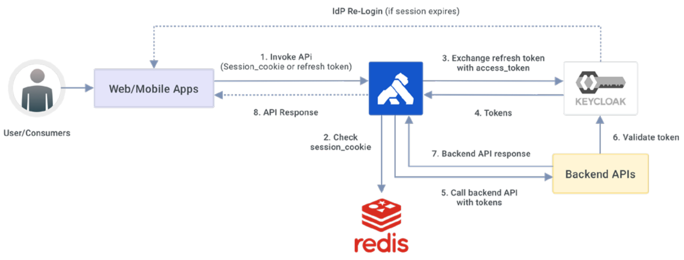
IdP Re-Login (if session expires)
User/Consumers
Web/Mobile Apps
1. Invoke API (Session_cookie or refresh token)
8. API Response
2. Check session_cookie
3. Exchange refresh token with access_token
4. Tokens
KEYCLOAK
6. Validate token
7. Backend API response
Backend APIs
5. Call backend API with tokens
redis

Diagram 2: API Access Workflow

1. The application invokes the API endpoint hosted in the Kong Gateway, passing the session cookie or refresh token in the header. If a session cookie or refresh token is not sent in the header, the gateway redirects the user for authentication as described in the login flow above.
2. Kong Gateway validates if a session cookie exists in the Redis cache.
3. Kong Gateway retrieves the refresh token from the cache or from the header and requests an access token from IdP using the refresh token.
4. The IdP exchanges the refresh token for an access token. Kong Gateway extracts the claim from the access token and validates (using consumer, roles or groups mapping) if the application has access to the upstream API.
5. Kong Gateway invokes the upstream API passing the token in the header.
6. Upstream API re-validates the access token with the IdP. This step is not required in all cases as the token has already been validated by the Kong Gateway, but some upstream applications require additional validation.
7. Upstream API sends the response back to the Kong Gateway.
8. The Kong Gateway sends the response back to the application.

Keycloak Implementation Details

- The application users and associated groups are onboarded to Keycloak. This group information is sent as part of the access token claims.
- OAuth defines two types of clients, confidential clients and public clients. Confidential clients are applications that are able to securely authenticate with the authorization server, for example being able to keep their registered client secret safe. Public clients are unable to use registered client secrets, such as applications running in a browser or on a mobile device. In our case, Kong will be acting as the frontend integrating with the IdP, so both mobile and web applications can be created as “confidential” clients. But if these were already existing, they would have been created as “public” clients for mobile applications and “confidential” clients for web applications.
- Make sure appropriate users are added to these applications’ access.

Kong Implementation Details

To support web, mobile and API authentication, we will create four Kong proxy endpoints (routes and services) and the OIDC plugins applied at each route level. The web browser automatically manages the session cookie, but the application must manage the refresh token.

Following are the required parameters for the OIDC plugin to support our requirement:

1. Authentication methods (grants and credentials): The plugin supports several authentication methods, but for this requirement, we would be using “session”, “authorization_code” and “refresh_token.”
2. Scope: Scope of the access, for example, openid, offline_ access, profile etc.
3. OIDC issuer: The discovery endpoint of the IdP, for example: https://keycloak.iam.svc.cluster.local/auth/realms/master
4. Application client id/client secret: Client application’s id and secret. For public access clients, there is no secret.
5. Consumer claim: Claim attribute from which application is mapped.
6. Session cookie lifetime: The session cookie lifetime in seconds.

7. Session storage: Session cookie storage medium - in memory, database or redis.
8. Session secret: Session cookie encryption key, so session cookie is not stored in plain text.
9. Login action: Action to be performed after successful login (or validation) - options are:
    a. Redirect: forward the request to some endpoint
    b. Response: send the response back to the client or
    c. Upstream: forward to upstream service configured.
10. Client authentication: Indicator on if authentication is required when Kong connects to IdP for token introspection or for getting access tokens.
11. Upstream headers: Headers to be passed to upstream services.
12. Unauthorized redirect: If authentication fails or the session has expired, this parameter provides the URL/endpoint where the user can re-login.
13. Downstream headers: Headers to be passed to the client or consumer applications.
14. Configure Redis cache for session storage management, supporting cluster and externalizing the session storage

Please refer to the OIDC documentation for information on additional parameters.

For your convenience, attached below is an example declarative YAML configuration which you can modify for your environment. Replace the issuer, client id, client secret, login and unauthorized URI(s), and Redis cache configuration to your specific instance values.

```yaml
_format_version: "1.1"
_workspace: DemoAppSecurity
services:
- connect_timeout: 60000
  host: mockbin.org
  name: DemoAPI
  port: 443
  protocol: https
  read_timeout: 60000
  retries: 5
  write_timeout: 60000
  routes:
  - name: MobileAPI
    paths:
    - /m/api/*
    path_handling: v0
```

```
      preserve_host: false
      protocols:
      - https
      regex_priority: 0
      strip_path: true
      https_redirect_status_code: 426
      request_buffering: true
      response_buffering: true
      plugins:
      - name: openid-connect
        config:
          auth_methods:
          - refresh_token
          client_auth:
          - none
          client_id:
          - ChangeClientId
          consumer_optional: true
          issuer: https://keycloak.iam.svc.cluster.local/
  auth/realms/master
          refresh_token_param_name: refresh_token
          session_cookie_lifetime: 36000
          session_cookie_renew: 6000
          session_redis_host: 172.18.0.5
          session_redis_port: 6379
          session_redis_prefix: sessions
          session_secret: changesecretvalue
          session_storage: redis
          unauthorized_error_message: Unauthorized
          unauthorized_redirect_uri:
          - https://mobile.app.com/unauthorized
          upstream_access_token_header:
  authorization:bearer
        enabled: true
        protocols:
        - grpc
        - grpcs
        - http
        - https
    - name: WebAPI
      paths:
      - /web/api/*
      path_handling: v0
      preserve_host: false
      protocols:
      - https
      regex_priority: 0
      strip_path: true
      https_redirect_status_code: 426
      request_buffering: true
      response_buffering: true
      plugins:
      - name: openid-connect
        config:
          auth_methods:
          - session
          client_id:
          - ChangeClientID
          client_secret:
          - ChangeClientSecret
        consumer_optional: true
        issuer: https://keycloak.iam.svc.cluster.local/
auth/realms/master
        scopes:
        - openid
        session_cookie_lifetime: 3600
        session_cookie_renew: 600
        session_redis_host: 172.18.0.5
        session_redis_port: 6379
        session_redis_prefix: sessions
        session_secret: changesecretvalue
        session_storage: redis
        unauthorized_error_message: Unauthorized
        unauthorized_redirect_uri:
        - https://example.web.com/unauthorized
      enabled: true
      protocols:
      - grpc
      - grpcs
      - http
      - https
routes:
- name: MobileLogin
  paths:
  - /mlogin
  path_handling: v0
  preserve_host: false
  protocols:
  - https
  regex_priority: 0
  strip_path: true
  https_redirect_status_code: 426
  request_buffering: true
  response_buffering: true
  plugins:
  - name: openid-connect
    config:
      auth_methods:
      - session
      - authorization_code
      client_auth:
      - none
      client_id:
      - ChangeClientId
      client_secret: null
      consumer_by:
      - username
      - custom_id
      consumer_optional: true
      issuer: https://keycloak.iam.svc.cluster.local/
auth/realms/master
      login_action: redirect
      login_methods:
      - authorization_code
      login_redirect_mode: query
      login_redirect_uri:
      - https://mobile.app.com/somepage
      login_tokens:
      - id_token
      - refresh_token
```

```
        logout_methods:
        - POST
        - DELETE
        scopes:
        - openid
        - offline_access
        scopes_claim:
        - scope
        session_cookie_lifetime: 36000
        session_cookie_renew: 6000
        session_redis_host: 172.18.0.5
        session_redis_port: 6379
        session_redis_prefix: sessions
        session_secret: changesecretvalue
        session_storage: redis
        unauthorized_error_message: Unauthorized
        unauthorized_redirect_uri:
        - https://mobile.app.com/unauthorized
      enabled: true
      protocols:
      - grpc
      - grpcs
      - http
      - https
  - name: WebLogin
    paths:
    - /wlogin
    path_handling: v0
    preserve_host: false
    protocols:
    - https
    regex_priority: 0
    strip_path: true
    https_redirect_status_code: 426
    request_buffering: true
    response_buffering: true
    plugins:
    - name: openid-connect
      config:
        auth_methods:
        - authorization_code
        - session
        client_id:
        - ChangeClientId
        client_secret:
        - ChangeClientSecret
        consumer_optional: true
        issuer: https://keycloak.iam.svc.cluster.local/
  auth/realms/master
        login_action: redirect
        login_methods:
        - authorization_code
        login_redirect_mode: fragment
        login_redirect_uri:
        - https://example.web.com/landingpage
        scopes:
        - openid
        scopes_claim:
        - scope
        session_cookie_lifetime: 3600
        session_cookie_renew: 600
        session_redis_host: 172.18.0.5
        session_redis_port: 6379
        session_redis_prefix: sessions
        session_secret: changesecretvalue
        session_storage: redis
        unauthorized_error_message: Unauthorized
        unauthorized_redirect_uri:
        - https://example.web.com/unauthorized
      enabled: true
      protocols:
      - grpc
      - grpcs
      - http
      - https
```

Following are the sequence of events that occur when accessing APIs from either web or mobile application

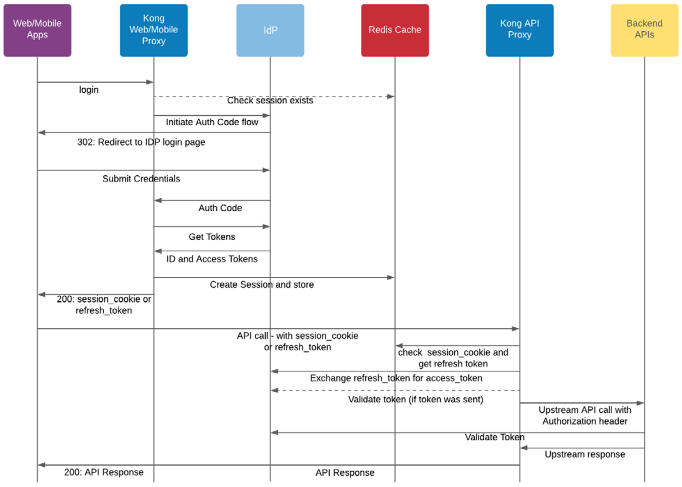
Web/Mobile Apps
Kong Web/Mobile Proxy
IdP
Redis Cache
Kong API Proxy
Backend APIs
login
Check session exists
Initiate Auth Code flow
302: Redirect to IDP login page
Submit Credentials
Auth Code
Get Tokens
ID and Access Tokens
Create Session and store
200: session_cookie or refresh_token
API call - with session_cookie or refresh_token
check session_cookie and get refresh token
Exchange refresh_token for access_token
Validate token (if token was sent)
Upstream API call with Authorization header
Validate Token
Upstream response
200: API Response
API Response

Diagram 3: Sequence of events that occur when accessing APIs from an application

Conclusion

In this document, we saw how the Kong Gateway can be used to protect web/mobile applications and APIs through a federated IdP. With Kong, you can leverage the IdP for both authentication and authorization without having to manage users or groups in Kong, giving you the ability to leverage a single centralized IdP to control access to upstream APIs. This helps in eliminating the additional overhead of writing redundant code to integrate the IdP with each application while ensuring consistent security best practices and standards are followed across the organization.

To learn more about the OIDC plugin configuration parameters, refer to the OpenID Connect documentation. For step-by-step OIDC installation refer to our “How to Secure APIs and Services Using OpenID Connect” blog post.

Kong

Konghq.com

Kong Inc.
contact@konghq.com

150 Spear Street, Suite 1600
San Francisco, CA 94105
USA