Implementing security and availability requirements for banking API system using Open Source Software (OSS)

Yoshiyuki Tabata
OSS Solution Center
Hitachi, Ltd.
Self introduction

Yoshiyuki Tabata:
OSS Solution Center, Hitachi, Ltd. @ Yokohama, Japan

- Software engineer
- API Management & Identity Management
  - 3scale, Keycloak
  - Contributor of 3scale
    - Developed “Edge Limiting policy”, “Keycloak Role Check policy”
Contents

1. Introduction: background and requirements
2. Usage of OSS to meet requirements
Background: Banking API and its security in Japan

- The revised banking act was published in Jun 2017 to promote API.
  - Similar to PSD2 in EU

- 83% of banks (114 banks) answered they will open API by 2020/6(*).
  (*): Based on survey of Japanese Bankers Association as of Dec 2017

- Security: **OAuth 2.0** is recognized as a key technology to secure API

---

Figure 1: Basic Framework of Open APIs (OAuth 2.0)

Quoted from Report about open API by the Japanese Bankers Association
Usage of OAuth 2.0: Authentication, Authorization

1. Who is using API? -> User authentication
2. What is using API? -> Client authentication

OAuth 2.0 (RFC 6749) only describes how tokens are issued. We have to use other standards or create something outside of standards.
## Requirements for Authentication/Authorization for banking API

<table>
<thead>
<tr>
<th>#</th>
<th>Category</th>
<th>Description</th>
</tr>
</thead>
</table>
| 1   | Authentication         | • Can support various (customized) authentication in OAuth flow  
• Compliance to OpenID Connect on top of OAuth                                                |
| 2   | Access control         | • Deny/Allow accesses based on claims in token  
• Can be combined with rate limit to protect backend                                           |
| 3   | Manage tokens          | • Revoke tokens triggered by users, administrators  
• Revoke tokens based on policy                                                               |
| 4   | Compliance to the latest standards | • Financial-grade API (FAPI) of OpenID Foundation                                      |
Background: Banking API and its availability

- Non-Functional Requirements 2018(*) was published in Apr 2018 to construct appropriate information systems, and enable stable provision of services. (*) Reported by Information-Technology Promotion Agency, Japan

- Information systems are categorized into 5 levels according to characteristics.

- In our experience, almost all banking API systems belong to over-Level 3.

<table>
<thead>
<tr>
<th>Level</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operating Rate</td>
<td>&gt; 99.999%</td>
<td>&gt; 99.99%</td>
<td>&gt; 99.9%</td>
<td>&gt; 99%</td>
</tr>
<tr>
<td>Total recovery time per year (MTTR)</td>
<td>&lt; 5.26 min</td>
<td>&lt; 52.6 min</td>
<td>&lt; 8.76 h</td>
<td>&lt; 87.6 h</td>
</tr>
<tr>
<td>Recovery time per failure</td>
<td>&lt; 1 min</td>
<td>&lt; 10 min</td>
<td>&lt; 1 h</td>
<td>&lt; 2 h</td>
</tr>
</tbody>
</table>
Achieve Level 3

How to minimize $MTTR \ (< 8h)$ and the recovery time per failure ($< 1 h$).

- Generally, to construct HA configuration and failover the system when a failure has occurred.
- To configure the system to be recovered automatically.  
  $\rightarrow$ **Fault Tolerance**  
  * This takes a high cost for preparing more resources than usual.
- To reduce dependencies of each component.  
  $\rightarrow$ **Fail Soft / Fault Avoidance**

![Diagram showing dependencies between Level 3 and Level 2 banking API systems.](image-url)
<table>
<thead>
<tr>
<th>#</th>
<th>Category</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Fault Tolerance</td>
<td>• HA configuration</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Can be recovered automatically</td>
</tr>
<tr>
<td>2</td>
<td>Fail Soft/</td>
<td>• Reduce dependencies of each component</td>
</tr>
<tr>
<td></td>
<td>Fault Avoidance</td>
<td></td>
</tr>
</tbody>
</table>
Contents

1. Introduction: background and requirements

2. Usage of OSS to meet requirements
   • Which OSS should be used?
   • Security requirements
   • Availability requirements
Open API system

- API Management product is usually used for common functions to open APIs
  - Rate limit, dev portal, analytics etc.
- It is desirable authentication/authorization are integrated into API management
There are various OSSs
We chose “3scale” and “Keycloak”
  • Completeness of feature
  • Activity and future of community

### Open Source Software (OSS) for open API

<table>
<thead>
<tr>
<th>OSS</th>
<th>API Management</th>
<th>Authentication/Authorization</th>
</tr>
</thead>
<tbody>
<tr>
<td>3scale</td>
<td>WSO2</td>
<td>OpenAM</td>
</tr>
<tr>
<td>Kong</td>
<td>tyk</td>
<td>Gluu</td>
</tr>
</tbody>
</table>
What is 3scale

OSS for API Management, community is led by Red Hat: https://github.com/3scale

- Include full functions of API management (not only API GW)
- Cloud native: Works on OpenShift or okd
- OAuth2, OIDC in combination with Keycloak
What is Keycloak

OSS for Identity Management, community is led by Red Hat: [https://www.keycloak.org](https://www.keycloak.org)
1. Introduction: background and requirements

2. Usage of OSS to meet requirements
   • Which OSS should be used?
   • Security requirements
   • Availability requirements
### <Recap> Requirements for Authentication/Authorization for banking API

<table>
<thead>
<tr>
<th>#</th>
<th>Category</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Authentication</td>
<td>• Can support various (customized) authentication in OAuth flow&lt;br&gt;• Compliance to OpenID Connect on top of OAuth</td>
</tr>
<tr>
<td>2</td>
<td>Access control</td>
<td>• Deny/Allow accesses based on claims in token&lt;br&gt;• Can be combined with rate limit to protect backend</td>
</tr>
<tr>
<td>3</td>
<td>Manage tokens</td>
<td>• Revoke tokens triggered by users, administrators&lt;br&gt;• Revoke tokens based on policy</td>
</tr>
<tr>
<td>4</td>
<td>Compliance to the latest standards</td>
<td>• Financial-grade API (FAPI) of OpenID Foundation</td>
</tr>
</tbody>
</table>

Implemented these requirements using 3scale + Keycloak, collaborating with OSS community
Authentication within OAuth/OIDC flow works well, basically

* OAuth 2.0 Dynamic Client Registration Protocol (RFC 7591)
Authentication : Authentication / Issue token

Authentication within OAuth/OIDC flow works well, basically e.g.) Authorization code grant

End user

Application

(1) Redirect to login screen
(2) Authenticates user using user data storage
(3) Authorization code
(4) Token request with client secret
(5) Access token and ID token

Keycloak

User data store
Authentication: Issues

1. PKCE (RFC 7636) is required to protect code
   - Redirect to login screen
   - Authenticates user using user data storage
   - Authorization code
   - Token request with client secret
   - Access token and ID token

2. Login screen is generated by Keycloak. However, it lacks high customizability.
PKCE support for Keycloak

- Keycloak did not support PKCE.
  -> We submitted PR and merged. [https://github.com/keycloak/keycloak/pull/3831](https://github.com/keycloak/keycloak/pull/3831)

- From Keycloak 3.1.0, PKCE was supported.
  - Enabled by default (no switch)
  - Only when PKCE is requested from a client, it works
  - Included in OIDC server metadata from 4.0.0
Highly customized login screen

Besides the template, login screen can be generated by delegated server

End user

Application

1. Delegates login screen by using Identity brokering feature
2. Login screen/logic can be coded as customers like

Keycloak

(1) Redirect to login screen
(3) Authorization code
(4) Token request with client secret
(5) Access token and ID token

AP server

We submitted a patch to enable forward parameters from Keycloak. https://github.com/keycloak/keycloak/pull/5163
Keycloak only issues tokens. Access control is out of scope.

APIcast did not support access control using tokens -> We submitted PRs.
How to access control using tokens

- The format of access token is not standardized neither RFC nor OIDC. It depends on implementation.

- In Keycloak, the format is similar to ID token of OIDC (JWT, claims).

  -> We targeted the Keycloak access token, and developed 2 policies(*).

(*) plugin to extend functions of APIcast

```json
{
  "jti": "c26a32c4-4b48-4c2f-a7da-3b9b8ecad652",
  "exp": 1535424101,
  "nbf": 0,
  "iat": 1535423801,
  "iss": "http://localhost:8080/auth/realms/provider",
  "aud": "broker",
  "sub": "e4b11e2e-9136-409b-8720-57463c627c10",
  "typ": "Bearer",
  "azp": "broker",
  "auth_time": 0,
  "session_state": "ac1767e2-2e30-4d44-b6f3-b77935a7a0bc",
  "acr": "1",
  "allowed-origins": [],
  "realm_access": {
    "roles": [
      "read",
      "additional",
      "write"
    ],
  },
  "name": "Takashi Mogi",
  "preferred_username": "mogi",
  "given_name": "Takashi",
  "family_name": "Mogi",
  "email": "mogi@example.com"
}```
Keycloak Role Check policy

- Checks “role” claims of access token and URL.
- We submitted a patch and included from 3scale 2.3.
  [https://github.com/3scale/apicast/pull/773](https://github.com/3scale/apicast/pull/773)

```
{  
  "jti": "c26a32c4-4b48-4c2f-a7da-3b9b8ecad652",  
  "exp": 1535424101,  
  ...  
  "allowed-origins": [],  
  "realm_access": {  
    "roles": [  
      "role1"  
    ]  
  },
```
Edge Limiting policy

Rate limiting: A kind of access control, to control the upper limit of traffics. APIcast did not support **STRICT** rate limiting to protect backend.

> We implemented patches and “Edge limiting policy” was included in 3scale 2.3.
Keycloak itself has features to revoke tokens

- **Revoke tokens triggered by administrator**
  - Can be revoked from admin console

- **Revoke tokens based on policy**
  - Timeout can be configured in admin console

- **Revoke tokens triggered by users**
  - Keycloak does not support OAuth 2.0 Token Revocation (RFC 7009)
  - Instead, logout endpoint(*) is used.

  
  (*) /auth/realms/<realm>/protocol/openid-connect/logout

  Related access tokens, ID tokens, refresh tokens are revoked.
Manage tokens : Issue

- Only authorization server knows that tokens are revoked…
  API gateways couldn’t deny API requests even if tokens were revoked.

- API gateways MUST ask the authorization server whether tokens were revoked.
  -> token introspection (RFC 7662)
Token Introspection policy

- We implemented patches and “Token Introspection policy” was included in 3scale 2.3. [https://github.com/3scale/APIcast/pull/619](https://github.com/3scale/APIcast/pull/619)

- This policy can cache the result of token introspection for reducing performance impact [https://github.com/3scale/APIcast/pull/656](https://github.com/3scale/APIcast/pull/656)
How API is called in 3scale 2.3 + Keycloak

1) API Request with token

Applications
(Web App, Mobile App)

3scale API Gateway
(APIcast)

2) Token Introspection
(Token Introspection policy)

Keycloak

3) Access control
(Role Check policy, Edge Limiting policy)

4) Extract necessary information from
access token and set header
(Header policy)

REST API Server

5) API Request with necessary
information in header
Compliance to the latest standard: FAPI

FAPI (Financial-Grade API) is being standardized in OpenID Foundation. Part1 (Read Only), Part2 (Read Write), JARM, CIBA

Spec to exchange access token (authorization info). A lots are left to implementers, insecure usage can easily happen.

In addition to OAuth, ID token (authentication info) can be included. Usage of OAuth is a bit hardened.

Secure usage of OAuth and OIDC is standardized.
FAPI in Japan

- FAPI is still implementer’s draft as of today
- However, being strongly promoted in banking industry

Quoted from “Report of Review Committee on Open APIs: Promoting Open Innovation”, Japanese Bankers Association

- We have to prepare for FAPI in advance, because can not implement soon.
Issues toward FAPI in Keycloak

Investigated implementation of Keycloak, and reported issues.

We were developing patches with community, major parts were resolved. Our colleague @tnorimat is mainly working.

<table>
<thead>
<tr>
<th>JIRA</th>
<th>Description</th>
<th>Pull Request</th>
<th>Included version</th>
</tr>
</thead>
<tbody>
<tr>
<td>KEYCLOAK-2604</td>
<td>RFC 7636(PKCE) support</td>
<td>3831</td>
<td>3.1.0</td>
</tr>
<tr>
<td>KEYCLOAK-5661</td>
<td>shall return the list of allowed scopes with the issued access token</td>
<td>4527</td>
<td>3.4.0</td>
</tr>
<tr>
<td>KEYCLOAK-5811</td>
<td>Client authentication client_secret_jwt</td>
<td>4835</td>
<td>4.0.0</td>
</tr>
<tr>
<td>KEYCLOAK-6700</td>
<td>Support of s_hash</td>
<td>5022</td>
<td>4.0.0</td>
</tr>
<tr>
<td>KEYCLOAK-6768</td>
<td>Support of Encrypted ID token</td>
<td>5779</td>
<td>Not yet</td>
</tr>
<tr>
<td>KEYCLOAK-6770</td>
<td>Signature algorithm (PS256 or ES256) support</td>
<td>5533</td>
<td>4.5.0</td>
</tr>
<tr>
<td>KEYCLOAK-8460</td>
<td>Signature algorithm (PS256 or ES256) support (for request object)</td>
<td>5603</td>
<td>4.7.0</td>
</tr>
<tr>
<td>KEYCLOAK-6771</td>
<td>Support for holder of key mechanism</td>
<td>5083</td>
<td>4.0.0</td>
</tr>
</tbody>
</table>
Contents

1. Introduction: background and requirements

2. Usage of OSS to meet requirements
   • Which OSS should be used?
   • Security requirements
   • Availability requirements
### Recap: Requirements for Availability for banking API

<table>
<thead>
<tr>
<th>#</th>
<th>Category</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Fault Tolerance</td>
<td>• HA configuration</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Can recover automatically</td>
</tr>
<tr>
<td>2</td>
<td>Fail Soft/ Fault Avoidance</td>
<td>• Reduce dependencies of each component</td>
</tr>
</tbody>
</table>

Implemented these requirements using 3scale, collaborating with OSS community
OpenShift provides:
- Automatic recovery/Automatic rerouting → **Automatic recovery**  
- Flexible scaling → **HA configuration**
Reduce dependencies of each component

Client Application ➔ apicast-[staging|production] ➔ API Backend

- Backend-worker ➔ Backend-listener
- Backend-redis ➔ Backend-cron
- System-[master|provider] / System-developer ➔ System-redis
- System-sidekiq ➔ System-memcache ➔ System-sphinx ➔ System-mysql ➔ Zync
- Keycloak ➔ Zync-database

Legend:
- API Gateway
- Authentication / Analytics
- Portals
- Data sync
- External components
Reduce dependencies of each component

- Client Application
- API Backend
- backend-worker
- backend-listener
- backend-redis
- backend-cron
- system-redis
- system-sidekiq
- system-master
- system-provider
- system-developer
- system-memcache
- system-sphinx
- system-mysql
- zync
- zync-database
- Keycloak

API Gateway
Authentication / Analytics
Portals
Data sync
External components
Reduce dependencies of each component

Client Application → apicast-[staging|production] → API Backend

- backend-listener
  - Authenticate & Report traffics
  - Download API configuration
  - Enqueue jobs

- backend-worker
  - Execute jobs

- backend-redis

- backend-cron

- system-[master|provider] / system-developer

- system-redis

- system-sidekiq

- system-memcache

- system-sphinx

- system-mysql

- sync

- Keycloak

- sync-database

Mission Critical components
Non-critical components
External components
Reduce dependency (APIcast to Backend)

We have to consider:

1. how to authenticate API requests
2. how to report traffics

when backend-listener is down
We have to consider:

1. **how to authenticate API requests**
   - cache the result of authentication and authenticate using cache
   - cannot authenticate newcomers and results to opportunity loss
   - allow newcomers without cache authentication and with alternative authentications

2. **how to report traffics**
   - cache traffics and report them all together when backend-listener comes back

---

**Caching policy**

- Keycloak Role Check policy, Edge Limiting policy, Token Introspection policy

**Batcher policy**
How API is called in 3scale 2.3

1) API Request with token

2) Token Introspection (Token Introspection policy)

3) Access control (Role Check policy, Edge Limiting policy)

4) Extract necessary information from access token and set header (Header policy)

5) Reduce dependencies (Caching policy, Batcher policy)

6) API Request with necessary information in header
Summary

- OAuth is recognized as a key technology for banking API systems

- Requirements to be considered around OAuth
  - Authentication, Access control, Token management, Latest standard (OIDC, FAPI)

- Requirements to be considered around Availability
  - HA configuration, Dependencies

- Applied OSS (3scale + Keycloak) to achieve them
  - Improved with OSS community
    - 3scale: enhanced rate limit, access control
    - Keycloak: Features required for FAPI
    -> Improvements are included in the latest version

- Let’s work with OSS community! 3scale and Keycloak are great community.
Trademarks

- Red Hat is a trademark or registered trademark of Red Hat, Inc. in the United States and other countries.
- OpenShift is a trademark or registered trademark of Red Hat, Inc. in the United States and other countries.
- WSO2 is a trademark or registered trademark of WSO2 in the United States and other countries.
- OpenID is a trademark or registered trademark of OpenID Foundation in the United States and other countries.
- GitHub is a trademark or registered trademark of GitHub, Inc. in the United States and other countries.
- Twitter is a trademark or registered trademark of Twitter, Inc. in the United States and other countries.
- Facebook is a trademark or registered trademark of Facebook, Inc. in the United States and other countries.
- Other brand names and product names used in this material are trademarks, registered trademarks, or trade names of their respective holders.