Published by :
http://www.ijert.org

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 12 Issue 04, April-2023

API Security: Protecting APIs With Keycloak

Danso Solomon Danquah, Yin Chunyong.
School of Computing and Software
Nanjing University of Information Science and Technology.
No.219 Ningliu Road, Jiangsu Province, China, 210044

***Abstract*—Nowadays, securing APIs is of paramount importance due to the interconnectedness of our world. Keycloak is an open-source tool that offers several security features, including authentication, authorization, and Single Sign-On. In this journal, we delve into the ways of protecting APIs with Keycloak. We investigate how to configure Keycloak with APIs and illustrate how it can be utilized to authenticate and authorize API requests. Additionally, we examine how to set up OAuth2/OpenID Connect integration and merge Keycloak with existing enterprise authentication systems. By employing Keycloak in API security, companies can enhance their overall security measures, reduce the possibility of data breaches, and provide a seamless user experience.**

Keywords—: API Security, Keycloak, Oauth2, OpenID.

I. INTRODUCTION

APIs, or application programming interfaces, are a crucial part of modern software development. They allow developers to access functionality and data from other applications, services, and platforms, making it possible to create complex, integrated systems that can provide a wide range of features and capabilities. However, the widespread use of APIs also presents a significant security risk, as unauthorized access to APIs can result in data breaches, theft of intellectual property, and other forms of cybercrime [4]. As a result, there is an urgent need to protect APIs and ensure that they are only accessed by authorized users and applications.

Keycloak is an open-source identity and access management system that provides authentication, authorization, and other security features for web applications and APIs. Keycloak is often used for protecting APIs due to its ability to provide secure authentication and authorization for both internal and external users, as well as its support for a wide range of authentication protocols and security standards.

Keycloak also provides a range of features for managing users and roles, including support for multi-factor authentication and fine-grained access control. This allows organizations to easily manage user accounts, enforce password policies, and control access to specific APIs and resources based on user roles and permissions. One of the key features of Keycloak is its support for OAuth 2.0 and OpenID Connect, which are widely used authentication protocols for web applications and APIs. These protocols provide a standardized way for applications to authenticate and authorize users, and Keycloak provides built-in support for both protocols, making it easy to integrate with a wide range of applications and APIs [3]. In addition to OAuth 2.0 and OpenID Connect, Keycloak also supports a range of other authentication protocols and security standards, including SAML, LDAP, and Kerberos. This flexibility makes it easy to integrate Keycloak with a wide range of existing systems and applications, regardless of their authentication requirements.

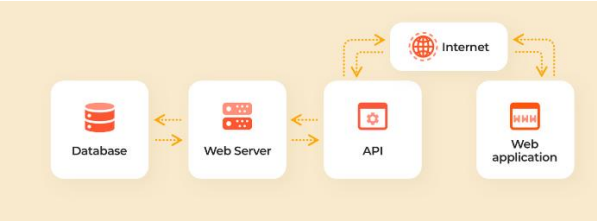


Fig. 1. Regular API Communication

The figure above shows how a regular web application communicates with an API. The communication goes through a network, but in most cases the internet to perform a CRUD (Create Retrieve Update Delete) operation. This action exposes the user from the web application to several security threats. It is therefore of topmost relevance to ensure that users from any application level is well protected when making API calls.

This study elaborates on the relevance of using Keycloak IAM to protect APIs, how to protect the API and the limitations of Keycloak.

II. RELATED WORK

[6] This study presents a case study on how to use the Spring Security framework in combination with KeyCloak-based OAuth2 to secure microservice architecture APIs.

The case study itself describes the development of a web-based application that uses a microservice architecture, with several RESTful APIs exposed to the front-end application. The authors then explain how they used Spring Security to secure the APIs, with KeyCloak as the authentication and authorization provider. They go into detail about the various Spring Security components used, such as authentication providers, security filters, and access control. The authors also describe the configuration steps needed to integrate Spring Security and KeyCloak, including the creation of KeyCloak clients and the configuration of OAuth2 settings. They provide code examples and configuration snippets to illustrate the process, making it easier for readers to replicate the setup. The paper then presents the results of the case study, which show that the Spring Security and KeyCloak-based OAuth2 setup provided effective security for the microservices. The authors also discuss the limitations of their approach, such as the need for additional configuration for more complex authorization scenarios.

[7] Keycloak is an open-source IAM system that provides authentication, authorization, and security for applications.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 12 Issue 04, April-2023

The authors introduce Keycloak, covering its architecture, installation, configuration, and integration with OpenID Connect and OAuth 2.0. They also provide guidance on how to configure Keycloak for different environments, such as development, testing, and production. Managing users and roles in Keycloak includes creating and managing user accounts, setting up groups and roles, and defining permissions and policies. The authors also provide guidance on how to customize user profiles, set up two-factor authentication, and integrate Keycloak with external identity providers. Finally, they provide a step-by-step guide on how to integrate Keycloak with various types of applications, including Java, Node.js, and PHP. Overall, "Keycloak-Identity and Access Management for Modern Applications" is an essential guidebook for anyone looking to implement a secure and scalable identity and access management system for modern applications. With its comprehensive coverage of Keycloak's features and its practical examples and use cases, this book is an excellent resource for both beginners and experienced professionals in the field of IAM.

III. THREAT MODEL

In the case of Keycloak, the threat model can be broken down into several categories, including:

A. Authentication

Keycloak is primarily responsible for providing authentication services for users and applications. Therefore, threats to the authentication process can compromise the entire system's security. Some potential threats include:

- Password guessing and brute force attacks
- Phishing attacks
- Man-in-the-middle attacks
- Session hijacking
- Credential stuffing

To mitigate these threats, Keycloak implements several security measures such as password policies, two-factor authentication, secure cookie settings, and SSL/TLS encryption.

B. Authorization

Keycloak provides access control services that ensure that only authorized users and applications can access protected resources. Threats to authorization can lead to data breaches and other security incidents. Some potential threats include:

- Access token theft and replay attacks
- Cross-site scripting (XSS) attacks
- Cross-site request forgery (CSRF) attacks
- Privilege escalation
- Insufficient access control

To mitigate these threats, Keycloak implements several security measures such as token encryption, token revocation, CORS policies, and role-based access control.

C. Infrastructure

Keycloak relies on infrastructure components such as databases, servers, and network devices. Any vulnerabilities in the infrastructure can lead to compromise the entire system's security. Some potential threats include:

- Server misconfiguration
- Network attacks
- Database vulnerabilities
- Operating system vulnerabilities
- Third-party software vulnerabilities

To mitigate these threats, Keycloak recommends implementing secure system configurations, regular vulnerability scanning, and software patching.

D. Administration.

Keycloak's administrative interface provides powerful tools for managing users, roles, and permissions. Therefore, any vulnerabilities in the administration interface can compromise the entire system's security. Some potential threats include:

- Weak or default passwords
- Insufficient user access control
- Insecure API endpoints
- Lack of audit logging
- Social engineering attacks

To mitigate these threats, Keycloak recommends implementing strong passwords, restricting user access to administrative functions, enabling secure API communication, and enabling audit logging.

IV. DESIGN

APIs are often used to expose critical data and services to external applications and users. Therefore, securing APIs is essential to prevent unauthorized access, data breaches, and other security threats. The consequences of API security breaches can be severe, including loss of reputation, legal liability, and financial losses. API security involves several layers of protection, such as authentication, authorization, encryption, and validation. Authentication verifies the identity of users and applications accessing the API, while authorization determines the level of access granted to them. Encryption ensures that the data transmitted between the API and its consumers is protected from interception and tampering. Validation ensures that the data sent to the API is in the expected format and within the acceptable limits. Keycloak provides a comprehensive set of features to secure APIs and manage identity and access control.

To protect an API with Keycloak, you need to perform the following steps:

a) Configure Keycloak client:

First, you need to create a Keycloak client that represents your API. You can do this by logging in to the Keycloak administration console, selecting the realm you want to use, and creating a new client.

When creating the client, you need to specify the following settings:

- Client ID: A unique identifier for the client.
- Client Protocol: The protocol used by the API (e.g., OpenID Connect, OAuth 2.0).
- Access Type: The type of access granted to the client (e.g., confidential, public).

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 12 Issue 04, April-2023

- Valid Redirect URIs: The URLs that the client is allowed to redirect to after authentication.

ONCE you have created the client, you need to note down its Client ID and Client Secret, which will be used to authenticate the API requests.

b) Secure API Endpoints:
Next, you need to secure the endpoints of your API by adding an authentication mechanism. You can do this by adding a security filter or interceptor to your API code that verifies the authenticity of the requests.

For example, if you are using Java and Spring Boot, you can use the Spring Security framework to secure your API endpoints. You can add the Keycloak Spring Security adapter to your project dependencies and configure it to use the Keycloak client you created earlier.
The Spring Security adapter provides several features, such as:

- Authentication: Verifying the identity of the user or application accessing the API.
- Authorization: Determining the level of access granted to the user or application.
- Session Management: Managing the user session and logout.

c) Test API Authentication:
To test API authentication with Keycloak, you first need to obtain an access token from the Keycloak authentication server. You can use a tool like Postman or cURL to send a request to the Keycloak server and receive an access token in the response. The access token will contain information about the user, client, and scope of the authentication request.
Once you have obtained an access token, you can use it to send authenticated requests to the API endpoint. In the request headers, you should include the access token in the Authorization header, using the Bearer scheme. The API server will then verify the access token with the Keycloak authentication server to ensure that it is valid and has the appropriate permissions to access the requested resource.
After sending an authenticated request to the API endpoint, you should receive a response from the server. The response should contain the requested resource or an appropriate error message if the request was not successful. You should verify that the response contains the expected data and that the authentication and authorization mechanisms are working as intended.
It's important to note that testing API authentication is not a one-time process. You should regularly test your API's authentication and authorization mechanisms to ensure that they are functioning correctly and that there are no vulnerabilities that could be exploited by attackers.

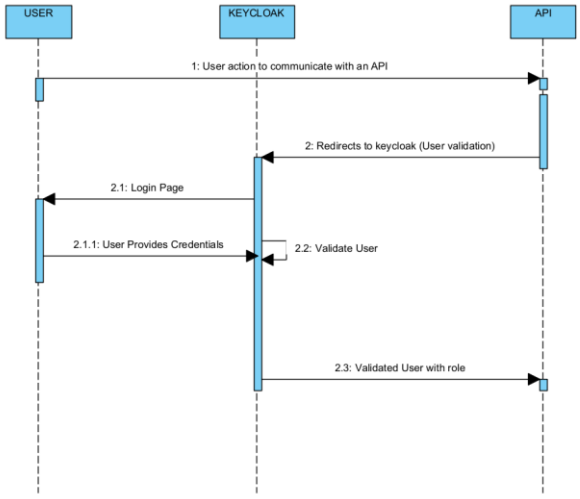


Fig. 2. Keycloak API Communication

Figure 2, shows how communication between an application and an API. First a request is made by the user interface to the API, however the API is structured to be accessed by tokens, hence, the API oauth2 authentication principal throws the user request to keycloak for the user to be authenticated and authorized. During the authentication and authorization process, keycloak provides a login interface for the user to provide the right credentials, when the credentials are right, keycloak then forwards the request back to the API with the tokens, indicating that the user has the right to make such calls to the API.

Results.
In Figure 3, show an error message with the HTTP status code of 401 when an endpoint is accessed in postman without providing the right credentials for keycloak to assign a token to the user, which will be used to access the endpoints.

Fig. 3. 401 Unauthorized

In figure 4, depicts how an authorized user can fully access the resource assigned. This shows an endpoint accessibility through keycloak. And gives HTTP status code of 200.

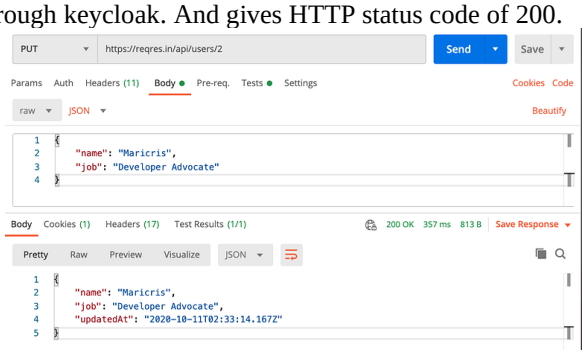


Fig. 4. 200 Success (Created)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 12 Issue 04, April-2023

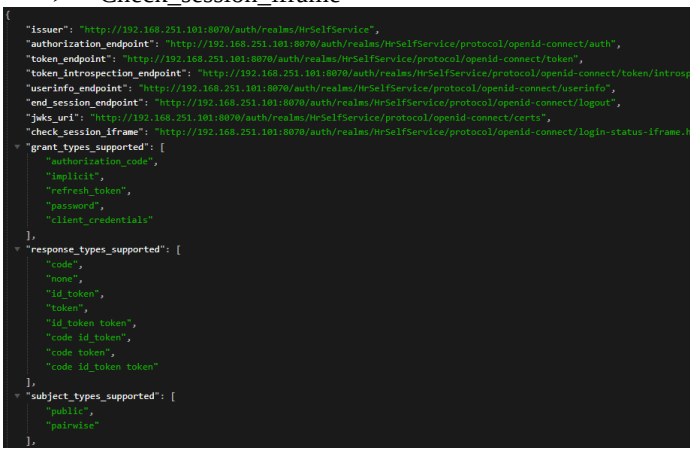
The figure 5, shows the various properties of the keycloak configurations.

Issuer: is a URL that identifies the Keycloak instance that issued a particular access token. This URL is included in the access token's metadata, and can be used by client applications to verify the authenticity and validity of the token.

Authorization_endpoint: is a URL that is used by client applications to obtain authorization from Keycloak before accessing protected resources. When a user tries to access a protected resource, the client application will redirect the user to the authorization endpoint to initiate the authorization process.

Also, there are various endpoints provided by keycloak after the configuration to help identify the user after authentication and authorization have been granted. These endpoints includes, but not limited to:

- Token_introspection_endpoint
- Userinfo_endpoint
- End_session_endpoint
- Jwks_uri
- Check_session_iframe



```
{
  "issuer": "http://192.168.251.101:8070/auth/realms/HrSelfService",
  "authorization_endpoint": "http://192.168.251.101:8070/auth/realms/HrSelfService/protocol/openid-connect/auth",
  "token_endpoint": "http://192.168.251.101:8070/auth/realms/HrSelfService/protocol/openid-connect/token",
  "token_introspection_endpoint": "http://192.168.251.101:8070/auth/realms/HrSelfService/protocol/openid-connect/token/introsp
  "userinfo_endpoint": "http://192.168.251.101:8070/auth/realms/HrSelfService/protocol/openid-connect/userinfo",
  "end_session_endpoint": "http://192.168.251.101:8070/auth/realms/HrSelfService/protocol/openid-connect/logout",
  "jwks_uri": "http://192.168.251.101:8070/auth/realms/HrSelfService/protocol/openid-connect/certs",
  "check_session_iframe": "http://192.168.251.101:8070/auth/realms/HrSelfService/protocol/openid-connect/login-status-iframe.
  "grant_types_supported": [
    "authorization_code",
    "implicit",
    "refresh_token",
    "password",
    "client_credentials"
  ],
  "response_types_supported": [
    "code",
    "none",
    "id_token",
    "token",
    "id_token token",
    "code id_token",
    "code token",
    "code id_token token"
  ],
  "subject_types_supported": [
    "public",
    "pairwise"
  ],
```


Fig. 5 Keycloak OpenID

IV CHALLENGES AND LIMITATIONS

Keycloak is an open-source Identity and Access Management (IAM) solution that offers a range of features such as single sign-on, social login, user federation, and fine-grained access control. However, organizations might face several challenges and limitations when implementing Keycloak.

One of the main challenges that organizations might face is the complexity of the system. Keycloak provides numerous configuration options, making it difficult for organizations to set up and maintain. Another challenge is integrating Keycloak with existing systems, especially in complex IT environments or legacy systems, which could cause issues. Additionally, Keycloak can be resource-intensive, especially when used with large user bases or complex access control policies. This can require significant computing resources and may increase costs. Moreover, while Keycloak provides customization options, there are limitations in how much organizations can customize the system to meet their specific needs. Furthermore, while Keycloak has extensive documentation, some users may find it challenging to navigate or may require additional support to fully understand the system. Additionally, upgrading to newer versions of Keycloak can be challenging, especially for organizations with custom extensions or integrations, which can require significant testing and development resources.

In conclusion, while Keycloak is a robust and flexible IAM solution, organizations need to be aware of the challenges and limitations they might face when implementing it. Proper planning and implementation can help mitigate these challenges and ensure a successful deployment of Keycloak.

V. CONCLUSION

Keycloak is a secure IAM solution that provides robust authentication and authorization features, along with comprehensive user management and security features. It is widely used in production environments and has a strong track record for security. However, as with any software, it is important to keep up with security updates and best practices to ensure continued security.

REFERENCES

[1] Tuecke, S., Ananthakrishnan, R., Chard, K., Lidman, M., McCollam, B. and Foster, I., 2016. Globus Auth: A research identity and access management platform. In 12th IEEE International Conference on e-Science.

[2] Nakandala, S., Gunasinghe, H., Marru, S. and Pierce, M., 2016. Apache Airavata Security Manager: Authentication and Authorization Implementations for a Multi-Tenant eScience Framework

[3] Sakimura, N., Bradley, J., Jones, M., de Medeiros, B. and Mortimore, C., 2014. Openid connect core 1.0. The OpenID Foundation, p.S3

[4] Hardt, D., 2012. The OAuth 2.0 authorization framework. [3] Basney, J., Fleury, T. and Gaynor, J., 2014. CILogon: A federated X. 509 certification authority for cyberinfrastructure logon. Concurrency and Computation: Practice and Experience, 26(13), pp.2225-2239.

[5] Christie, M. A., Bhandar, A., Nakandala, S., Marru, S., Abeysinghe, E., Pamidighantam, S., & Pierce, M. E. (2017). Using keycloak for gateway authentication and authorization.

[6] Chatterjee, A., & Prinz, A. (2022). Applying spring security framework with KeyCloak-based OAuth2 to protect microservice architecture APIs: a case study. Sensors, 22(5), 1703.

[7] Thorgersen, S., & Silva, P. I. (2021). Keycloak-identity and access management for modern applications: harness the power of Keycloak, OpenID Connect, and OAuth 2.0 protocols to secure applications. Packt Publishing Ltd.