FEATURE

Keycloak tutorial: How to secure different application types

IT pros and developers can secure applications with the open source IAM tool Keycloak. When you don't need to worry about passwords, it reduces the potential attack surface.

By **Kyle Johnson,** Technology Editor | **Packt Publishing**

Hackers dig through applications to find any vulnerability to exploit. The easiest method of entry often is to steal digital identities. To reduce the attack surface, companies can implement identity and access management tools. Once developers add them to an application, system admins can do the following:

- *automate tasks;*
- *use role-based access control; and*
- *offload authentication to a third-party system built for IAM.*

Keycloak is an open source authentication tool that suits this mission.

In Keycloak: Identity and Access Management for Modern Applications*, authors Stian Thorgersen and Pedro Igor Silva offer a Keycloak tutorial. They cover everything from reviewing what Keycloak offers companies, management and identity brokering to enabling single-sign on, Lightweight Directory Access Protocol (LDAP) or Active Directory sync, and more. With this IAM tool, developers and system admins do not need to worry about how to store and protect user passwords.*

In the following excerpt from Chapter 6, Thorgersen and Silva guide readers on using Keycloak to secure internal, external, web and server-side applications. "With web technologies, like REST APIs, single-page applications or a server-side application fits quite naturally and works well with Keycloak," Thorgersen said in a call. The rest of the chapter digs into how to provide security for single-page, native and mobile applications, as well as REST APIs and other services.

In this chapter, we will first begin by understanding whether the application we want to secure is an internal or external application. Then, we will look at how to secure a range of different application types, including web, native, and mobile applications. We will also look at how to secure REST APIs and other types of services with bearer tokens.

By the end of this chapter, you will have learned the principles and best practices behind securing different types of applications. You will understand how to secure web, mobile, and native applications, as well as how bearer tokens can be used to protect any type of service, including REST APIs, gRPC, WebSocket, and other types of services.

In this chapter, we're going to cover the following main topics:

- Understanding internal and external applications
- Securing web applications
- Securing native and mobile applications
- Securing REST APIs and services

Technical requirements

To run the sample application included in this chapter, you need to have Node.js (https://nodejs.org/) installed on your workstation.

You also need to have a local copy of the GitHub repository associated with the book. If you have Git installed, you can clone the repository by running this command in a terminal:

```
$ git clone https://github.com/PacktPublishing/Keycloak-
Identity-and-Access-Management-for-Modern-Applications.git
```

Alternatively, you can download a ZIP of the repository from https://github.com/PacktPublishing/Keycloak-Identity-and-Access-Management-for- Modern-Applications/archive/master.zip.

Check out the following link to see the Code in Action video:

https://bit.ly/3b5R0F2

Understanding internal and external applications

When securing an application, the first thing to consider is whether the application is an internal application or an external application.

Internal applications, sometimes referred to as first-party applications, are applications owned by the enterprise. It does not matter who developed the application, nor does it matter how it is hosted. The application could be an off-the-shelf application, and it can also be a **Software as a Service (SaaS)**-hosted application, while still being considered an internal application.

For an internal application, there is no need to ask the user to grant access to the application when authenticating to the user, as this application is trusted and the administrator that registered the application with Keycloak can pre-approve the access on behalf of the user. In Keycloak, this is done by turning off the **Consent Required** option for the client, as shown in the following screenshot:

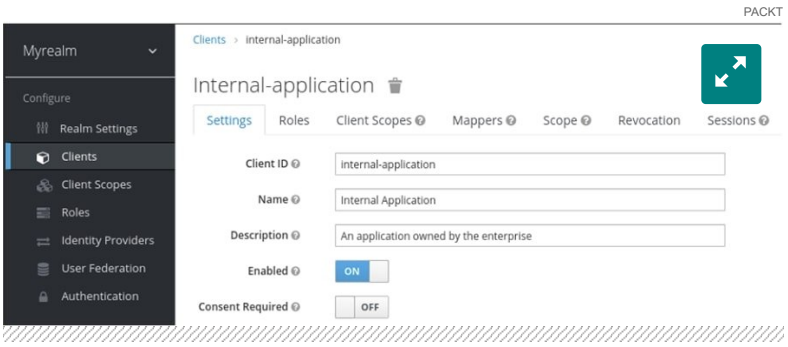


Figure 6.1 -- Internal application configured to not require consent

When a user authenticates or grants access to an internal application, the user is only required to enter the username and password. For external applications, on the other hand, a user must also grant access to the application.

External applications, sometimes referred to as third-party applications, are applications that are not owned and managed by the enterprise itself, but rather by a third party. All external applications should have the **Consent Required** option enabled, as shown in the following screenshot:

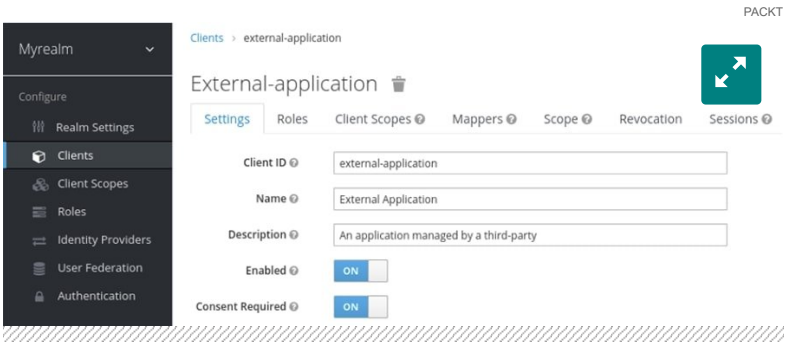


Figure 6.2 -- External application configured to require consent

When a user authenticates or grants access to an external application, the user is required to not only enter the username and password but also to grant access to the application, as shown in the following screenshot:

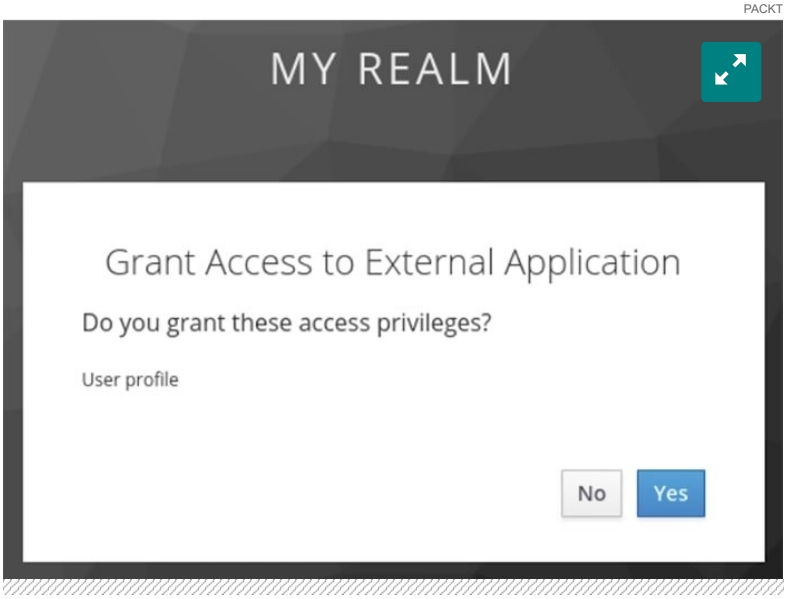


Figure 6.3 -- User granting access to an external application

You should now understand the difference between an internal and an external application, including how to require users to grant access to external applications. In the next section, we will look at how to secure web applications with Keycloak.

Securing web applications

When securing a web application with Keycloak, the first thing you should consider is the architecture of the application as there are multiple approaches:

- First and foremost, is your web application a traditional web application running on the server side or a modern **single-page application (SPA)** running in the browser?
- The second thing to consider is whether the application is accessing any REST APIs, and if so, are the REST APIs a part of the application or external?

If it is a SPA-type application invoking external APIs, then there are two further options to consider. Does the application invoke the external REST API directly, or through a dedicated REST API hosted alongside the application?

Based on this, you should determine which of the following matches the architecture of the application you are securing:

- **Server side**: If the web application is running inside a web server or an application server.
- **SPA with dedicated REST API**: If the application is running in the browser and is only invoking a dedicated REST API under the same domain.
- **SPA with intermediary API**: If the application is running in the browser and invokes external REST APIs through an intermediary API, where the intermediary API is hosted under the same domain as the application
- **SPA with external API**: If the application is running in the browser and only invokes APIs hosted under different domains.

Before we take a look at details specific to these different web application architectures, let's consider what is common among all architectures.

Firstly, and most importantly, you should secure your web application using the Authorization Code flow with the **Proof Key for Code Exchange** (**PKCE**) extension. If you are not sure what the Authorization Code flow is, you should read *Chapter 4*, *Authenticating Users with OpenID Connect*, before continuing with this chapter. The PKCE extension is an extension to OAuth 2.0 that binds the authorization code to the application that sent the authorization request. This prevents abuse of the authorization code if it is intercepted. We are not covering PKCE in detail in this book, as we recommend you use a library. If you do decide not to use a library, you should refer to the specifications on how to implement support for OAuth 2.0 and OpenID Connect yourself.

When porting existing applications to use Keycloak, it may be tempting to keep the login pages in the existing application, then exchanging the username and password for tokens, by using the Resource Owner Password Credential grant to obtain tokens. This would be similar to how you would integrate your application with an LDAP server.

More on *Keycloak -- Identity and Access Management for Modern Applications*

In an interview, author Stian Thorgersen discussed the open source IAM software Keycloak that he helped develop.

To read the rest of Chapter 6, click here.

Click here to learn more about *Keycloak -- Identity and Access Management for Modern Applications* here.

However, this is simply something that you should not be tempted to do. Collecting user credentials in an application effectively means that if a single application is compromised, an attacker would likely have access to all applications that the user can access. This includes applications not secured by Keycloak, as users often reuse passwords. You also do not have the ability to introduce stronger authentication, such as two-factor authentication. Finally, you do not get all the benefits of using Keycloak with this approach, such as **single sign-on** (**SSO**) and social login.

As an alternative to keeping the login pages within your existing applications, you may be tempted to embed the Keycloak login page as an iframe within the application. This is also something that you should avoid doing. With the login page embedded into the application, it can be affected by vulnerabilities in an application, potentially allowing an attacker access to the username and password.

As the login page is rendered within an iframe, it is also not easy for users to see where the login pages are coming from, and users may not trust entering their passwords into the application directly. Finally, with third-party cookies being frequently used for tracking across multiple sites, browsers are becoming more and more aggressive against blocking third-party cookies, which may result in the Keycloak login pages not having access to the cookies it needs to function.

In summary, you should get used to the fact that an application should redirect the user to a trusted identity provider for authentication, especially in SSO scenarios. This is also a pattern that most of your users will already be familiar with as it is widely in use nowadays. The following screenshot shows an example of the Google and Amazon login pages, where you can see that they are not embedded in the applications themselves:

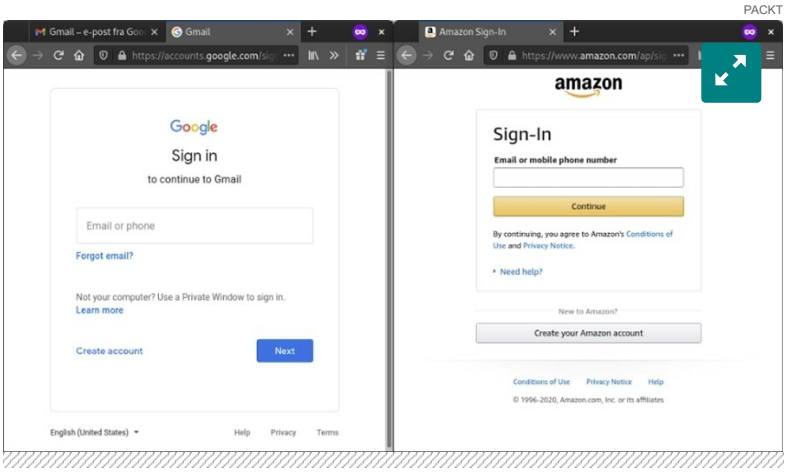


Figure 6.4 -- Example from Google and Amazon showing external login pages

You should now have a good, basic understanding of how to go about securing a web application with Keycloak. In the next section, we will start looking at how to secure different types of web applications, starting with server-side web applications.

Securing server-side web applications

When securing a server-side web application with Keycloak, you should register a confidential client with Keycloak. As you are using a confidential client, a leaked authorization code can't be leveraged by an attacker. It is still good practice to leverage the PKCE extension as it provides protection against other types of attacks.

You must also configure applicable redirect URIs for the client as otherwise, you are creating what is called an open redirect. An open redirect can be used, for example, in a spamming attack to make a user believe they are clicking on a link to a trusted site. As an example, if a spammer sends the `https://trusted-site.com/...?redirect_uri=https://attacker.com` URL to a user in an email, the user may only notice the domain name is to a trusted site and click on the link. If you have not configured an exact redirect URI for your client, Keycloak would end up redirecting the user to the site provided by the attacker.

With a server-side web application, usually, only the ID token is leveraged to establish an HTTP session. The server-side web application can also leverage an access token if it wants to invoke external REST APIs under the context of the user.

The following diagram shows the flow for a server-side web application:

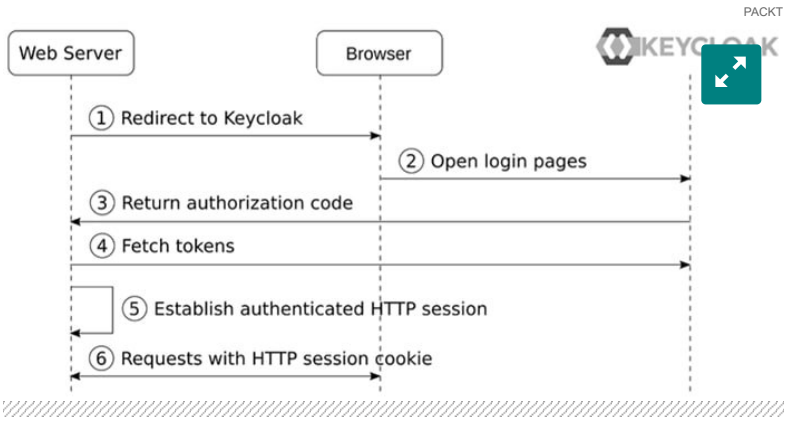


Figure 6.5 -- Server-side web application

In more detail, the steps in the diagram are as follows:

1. The **web server** redirects the browser to the **Keycloak** login pages using the **Authorization Code** flow
2. The user authenticates with Keycloak.
3. The authorization code is returned to the server-side web application.
4. The application exchanges the authorization code for tokens, using the credentials registered with the client in Keycloak.
5. The application retrieves the ID token directly from Keycloak as it does not need to verify the token, and can directly parse the ID token to find out information about the authenticated user, and establish an authenticated HTTP session.
6. Requests from the browser now include the HTTP session cookie.

In summary, the application leverages the Authorization Code flow to obtain an ID token from Keycloak, which it uses to establish an authenticated HTTP session.

For server-side web applications, you can also choose to use SAML 2.0, rather than using OpenID Connect. As OpenID Connect is generally easier to work with, it is recommended to use OpenID Connect rather than SAML 2.0, unless your application already supports SAML 2.0.

About the authors

Stian Thorgersen *has worked at Arjuna Technologies building a cloud federation platform and at Red Hat, looking for ways to make developers' lives easier. In 2013, Thorgersen co-founded the Keycloak project with another developer at Red Hat. Today, Thorgersen is the Keycloak project lead and top contributor to the project. He is a senior principal software engineer at Red Hat, focusing on identity and access management.*

Pedro Igor Silva *has experience with open source projects, such as FreeBSD and Linux, as well as a Java and J2EE. He has worked at an ISP and as a Java software engineer, system engineer, system architect and consultant. Today, Silva is a principal software engineer at Red Hat and one of the core developers of Keycloak. He studies IT security, specifically application security and identity and access management.*

This was last published in July 2021

Related Resources

MicroScope – February 2021: The forecast on channel security
–MicroScope

Making Sure Your Identity and Access Management Program is Doing What You Need

–TechTarget Security

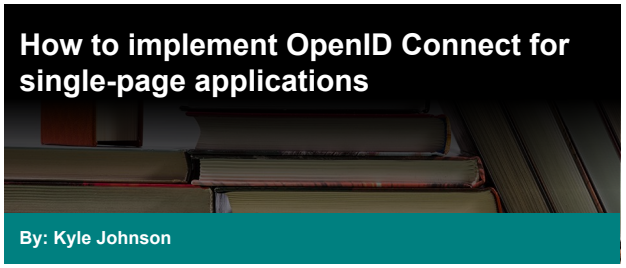
E-Guide: How to tie SIM to identity management for security effectiveness
–TechTarget Security

Extended Enterprise Poses Identity and Access Management Challenges
–TechTarget Security

Dig Deeper on Identity and access management

By: Alex Scroxton

By: Priyank Gupta



By: Kyle Johnson

By: Kyle Johnson

Latest TechTarget
resources

- NETWORKING
- CIO
- ENTERPRISE DESKTOP
- CLOUD COMPUTING
- COMPUTER WEEKLY

Networking

Cisco, HPE plug holes in cloud security portfolios

Hewlett Packard Enterprise also unveiled plans to acquire Athonet, an Italian company that provides cellular technology for ...

9 practice questions on twisted-pair network cables

Take this practice quiz on twisted-pair cables, sampled from 'Networking Essentials: A CompTIA Network+ N10-008 Textbook,' to ...