Session Management Techniques

in the Curity Identity Server

CURITY

Session Management Techniques
in Curity Identity Server

Table of Contents

Introduction

Authenticating a user is not a trivial task. In complicated projects, authentication is very often a multi-step process driven by different scenarios. Eventually, the goal is to identify a user to grant them access to the protected application. Once an application authenticates a user, it will normally keep the authentication data in the form of a session. Therefore, the user does not have to complete the complicated authentication process with every request.

As the authentication process became more complex, filled with different features to increase security and certainty, session management became more and more critical. It's no longer enough to create a session cookie, let the user log out when they want, or let the browser remove the cookie on exit. Applications require more complex user sessions management, especially between different devices used to access the same application.

Certain session management features are becoming common throughout the market. These three, in our opinion, deserve attention:

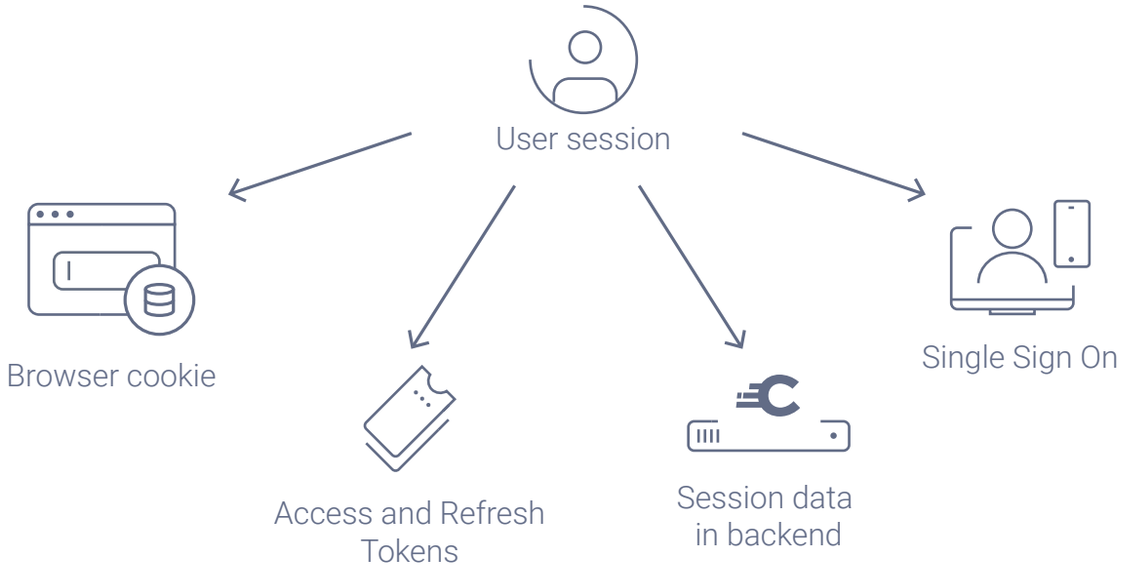
1. One active session. Ability to have only one session between instances of the same application.
2. Showing notifications in one instance of an application when another instance changes the session state.
3. Listing active user sessions and manually revoking some of them.

What Is a Session? (What to Revoke)

When we think about a user session, a session cookie usually comes to mind. This association is, of course, a valid one. However, complex session management involves many more features.

In a general sense, the user session is a feature that allows the application and the resource server to identify the user and authorize their actions (to perform authentication and authorization). Different mechanisms can achieve that goal — both on the frontend and the backend. A session cookie is one such mechanism.

The cookie will often contain a session identifier that corresponds to a session object on the server side. Access tokens and refresh tokens can enable this. An access token gives direct access to a user's resources, whereas the refresh token obtains access tokens. Single Sign-On (SSO) features also enable an application to retain a session. SSO features might not represent a user session per se, but they allow the application to authenticate a user and thus access a session seamlessly.



All the different techniques used for maintaining a user session need to be kept in mind when designing any session management feature. This is especially true when we consider session revocation. When a session is to be revoked, what should be cleared? The frontend cookie, the access token, or the backend session storage?

Some projects don't care about revoking access tokens because their tokens are only valid for one minute. Other projects might not even require SSO features at all. Others will want to ensure that once a user's session is terminated on a device, no one on that device can easily re-authenti-

cate using SSO functionality.

It's also worth noting that revoking a session, while resembling logout, does not always mean the same thing as a logout. If an application cancels a session only on the backend, the user may still be logged-in on the frontend and would have to log out actively. The server can remove access to the resources, but the client application may still retain information about the user identity. Session specialists must consider this at design time.

Client vs. Project

In OAuth, the application requesting tokens from an Authorization Server is called the Client (or Relying Party (RP) when OpenID Connect nomenclature is used). In this document, however, we will refer to a **project**, which in some cases, has a broader meaning than the Client or RP.

Take, for example, a banking application. It is accessible from a browser and Android, but both apps have their own client IDs. A system like this may be architected for a few different reasons:

- The two apps have different confidentiality levels (one is a public client, and one is private).
- There might be a need to differentiate them for statistical purposes.
- They might have different configurations that require various authentication methods.

Since both client types make up the same broader application, session manipulation for one of those clients should affect them all.

You can group clients using Curity Identity Server in a few different ways:

- When a high level of separation is needed between projects, you can create separate Profile Groups for each of them. This will give you a clean separation but will also mean that you won't be able to reuse some elements, e.g., you must configure authentication methods in every Profile Group. Note also that every Profile Group will use a different path for the OAuth and OpenID Connect endpoints.
- In many cases, when plugins are used to introduce the different session management features, it is sufficient to create configurable lists of clients that make up the project. You can add such lists as part of the plugin configuration.
- If the grouping is only used to manage SSO sessions, it is enough to differentiate the groups using different authenticator instances. For example, if client1 and client2 are supposed to be in one group, they would use authenticator google-1. The client3, which is not part of the group, would use authenticator google-2. Google-1 and google-2 could both have the same configuration but would be separate authenticator entities.

One Active Session

User sessions are closely tied to the frontend instance used to create it, whether it's a browser, mobile application, or a desktop app. If a user authenticates on a device or browser, they must log in again and create a new session when they open the same application on another device or browser. Only then can the user freely access their resources from either a browser or device.

In many situations, this is a desirable outcome, but some scenarios must ensure that a user logs in to only one instance of your project at a time. It might be because the application has sensitive data. Or, the content may be behind a paywall, and the user is forbidden access. In these situations, any active user session (any permit issued to access resources) should be immediately revoked when the user initiates another session from a different device or browser.

In general, the Authorization Server should listen to any new access credentials issued, check whether such credentials are already issued for the same project, and revoke older ones. As noted before, what exactly is revoked depends on the concrete setup of a given project. It might be the access token, the delegation, or the SSO session as well.

The problem and solution described above pertain to maintaining one active user session. Of course, one could easily modify this strategy to cater to any number of active sessions.

How to Achieve "One Session" in Curity

One can achieve a “one session” feature in Curity Identity Server through plugins and configuration. In this section, we'll explain how to implement this behavior.

Prerequisites

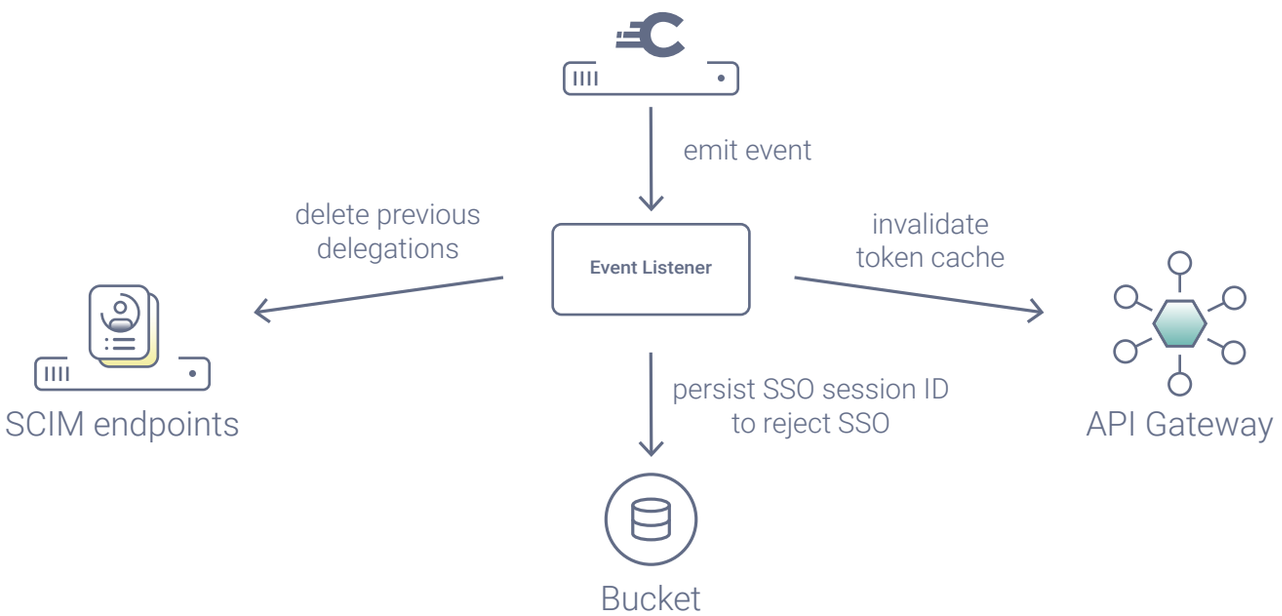
Below are some prerequisites to the settings used in the Curity Identity Server. Only some are applicable, depending on the exact result you want to achieve.

- To easily revoke access tokens, one should use the Phantom Token or Split Token approach. This approach shares an opaque token with the client, and it is introspected at the API Gateway and exchanged for a JWT.
- If you use JWTs externally, then they can’t be revoked. In such a situation, it’s best to use reasonably short expiration times for the tokens. The user will then have access from all the places they logged in from only for as long as the access token is valid.
- If you want to revoke access or refresh tokens through the management of delegations, the User Management Profile needs to be enabled.

- If revoking the SSO session should be a part of this solution, all clients who make up the project should share the same authenticators. All clients that should lose access to the SSO session should share the same instances of authenticators.

Solution

Curity Identity Server issues access tokens based on delegations kept in a data source. Thus, to revoke a token, you must remove a delegation from the data source. You can manage delegations through SCIM endpoints exposed by the User Management Profile.



The "one session" mechanism can be summarized as "whenever X happens, remove any sessions for the given user." Curity Identity Server issues events on many different occasions, enabling you to react accordingly. In the case of a "one session" feature, the concrete event will also vary depending on what exactly needs to be achieved. If you're only interested in revoking any issued access or refresh tokens, then it would be best to listen to the `IssuedDelegationOAuthEvent`, which occurs every time a new delegation is given. It's worth noting that the delegation is created only when a new token is issued to the client. This could mean that a user can successfully authenticate and create an SSO session but does not have to eventually get an access token. For example, they could reject consent, or could not finish the code flow. If such cases are critical to your project, it would be better to listen to the `CreatedSsoSessionEvent`.

An event listener can use an HTTP client to send requests to a SCIM delegations endpoint to find and revoke any active delegations for the user.

SSO sessions can't be deleted directly from a plugin in Curity Identity Server, so you must use

another solution if SSO sessions are a concern. Whenever an SSO session would have to be revoked you can instead add the session id to a "revoked list" kept in a bucket. You can then create an authentication action that reads the list from that bucket and checks whether the session used to authenticate the user in the current flow is on the revoked list. If it is, the action can deny authentication.

For such situations, the action would return an authentication error, which would mean that the user would have to actively log out to log in again from the given device (or the client could automatically initiate the logout). You would have to add such an authentication action to the SSO flow for all of the authenticators across your project.

Note on Split and Phantom Token flows

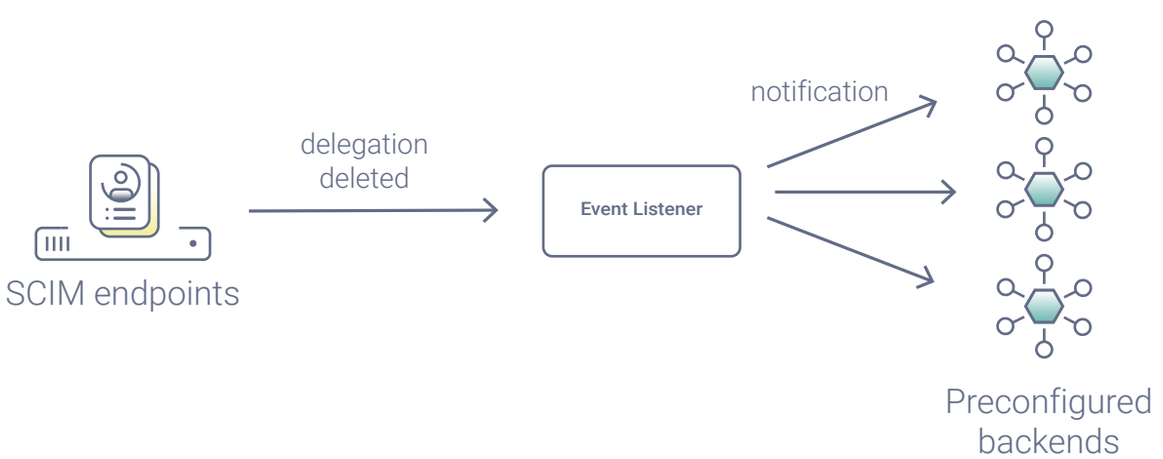
If you use either the Split Token flow or the Phantom Token flow, then the API Gateway may use its cache to exchange opaque tokens for JWTs. In such a case, deleting a delegation kept at the Curity Identity Server will not be enough to revoke an access token. Additional behavior would have to be added to the server, where, whenever a delegation is revoked, Curity Identity Server sends a request to the API Gateway instructing it to revoke any tokens connected to the removed delegation. This might mean that the API Gateway would also have to keep a map of its token entries and delegation IDs.

Notification on Session Revocation

Usually, from a security or business-value point of view, the most important thing is to terminate the user's session. You must perform an action that disallows access to any protected resources from the other device or browser until the user actively logs in again. From a User Experience point of view, it is desirable to notify the user that their session on a given device has been terminated. You can use this in conjunction with the "one session" feature, but it is not necessarily limited to that feature. It may be necessary for different situations to inform the user that the state of their sessions has changed. For example, perhaps the administrator logged out the user due to suspicious behavior or reset the user's password and wants to terminate the session.

How to Achieve It in Curity

An excellent way to send notifications to clients from Curity Identity Server is to use the Security Events Push mechanism standard draft. Using this standard, you can create an event listener that knows which clients to notify, along with the endpoints they expose. The listener can then use an HTTP Client to send security events to those endpoints. Note that these events should be sent to endpoints exposed by the backends of those clients.



When an event is received, the backend will notify a concrete frontend instance of the change. For example, the front-end client polls the backend periodically, asking for changes in the session. Perhaps a two-way communication is established between the front-end client and a backend to deliver notifications to the user seamlessly.

Session Listing and Revoking

Automated session management is not always a required feature. There are situations where we want the users to have control over their sessions. Your project may require a user to view all devices and clients they are logged into. You may want to allow the user to terminate a particular session they don't recognize. Or, perhaps they don't use a given device any longer.

As said previously, what exactly constitutes session termination will depend on the concrete configuration of your project. Sometimes it will be enough to delete the delegation so that access tokens can't be used or issued (with the use of a refresh token). At other times you will also need to ensure a user can't reuse an SSO session.

How to Achieve It in Curity

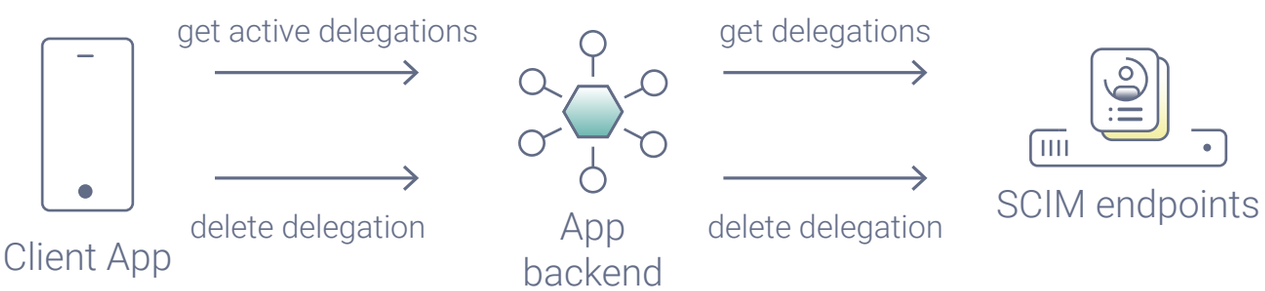
The Curity Identity Server exposes SCIM endpoints, which can be used to manage data concerning users. Among that, the delegations for a user can be listed and can be deleted if necessary. You can use this feature to provide a user with information on an active session and allow them to manage those sessions.

Prerequisites

- The User Management Profile needs to be enabled so that it is possible to list the delegations belonging to a user.
- When it comes to revocation of the session, all the prerequisites described in the "One active session" part apply here as well.

Solution

When the User Management is enabled you can access the SCIM endpoints, and thus the `/Delegations` endpoint. You can send a filter query parameter to the endpoint to filter the results. As the SCIM endpoints are sensitive, the recommended method would be to have a backend client communicating with the Curity Identity Server so that security can be maintained at a high level.



You should also remember to secure the User Management profile properly so that only authorized clients can communicate with the SCIM endpoints. This is especially important if you want to let public clients communicate directly. Remember that access should be restricted accordingly by properly configuring Authorization Managers. For example, limit a public client's access to only read and modify the current authenticated user's data.

The delegation list currently returns, among other things, the ID of the client and the time of the authentication. This is the minimum that the user might need to be able to identify the sessions.

If your session revocation also requires a revoked SSO session, you could use the mechanism described in detail in the “One Active Session” section. When withdrawing the delegation, you can add the SSO session to a “revoked list” in a bucket. Then, authenticators should act in the SSO flow, checking whether the current user session is one from the revoked list. If so, the authentication should fail.

Remember that a delegation is not equal to an SSO session. There are scenarios in which there might not be a delegation in the database, but the user will already have an active SSO session. This can happen when you enable your users to deny consent to the client requesting the login. At the time when the user denies consent, they will already have an active SSO session, but no delegation will be created, as no tokens are issued. Keep that in mind if the SSO sessions are vital to the features you want to implement.

Conclusion

Managing user sessions can be an important part of a business. Session management can add additional features to improve the user experience, like the ability to list and revoke one's sessions or be notified whenever the session is terminated from another place. But it can also add automatic features which may be vital to some businesses, like the "one session" feature.

The ability to limit how many active sessions one user maintains can be used in many scenarios. It increases application security by ensuring no one else is using the same account on a different device. It also helps curb any abuses that might happen in paid services.

Whatever the case, it is possible to implement all these solutions using Curity Identity Server. Should you have any questions about the solutions presented here, or you need help with the implementation, don't hesitate to contact us.

For Further Information

If you have questions about anything written in this paper or would like to learn more about how to apply them to your situation, contact us by email at info@curity.io. You can also connect with us on Twitter where we are @curityio.

More information can also be found on our website https://curity.io.

CURITY

Address
Curity AB
S:t Göransgatan 66
112 33 Stockholm
Sweden

Website
curity.io

Email
info@curity.io

Twitter
@curityio