Safewalk authentication API

Release 3.0.x

AltiPeak

Feb 07, 2021

Contents

Safewalk authentication API is an OAuth2 Restful API that can be called by any user with an authentication access key.

It can be used with three different variants:

- Safewalk input-based authentication API - Used for validation of all Safewalk's input-based supported authenticators.
- Safewalk Second-Step authentication API - Similar to the above input-based authentication API, but when this API is called Safewalk will skip the first step validation and jump directly to the second-step part.
- Safewalk Fast Auth Passwordless, 2-Way, Anonymous and Input-less (2WAIL) authentication API - A Passwordless QR/Deep-link (mobile) authentication method that allows for a smooth authentication experience for users to access a protected website using a browser.

1 Safewalk input-based authentication API

The Safewalk input-based authentication API is used for validation of all Safewalk's input-based supported authenticators. That includes all authenticators licenses type excluding the QR/Deep-link Passwordless methods.

1.1 Creating an OAUTH token for the application account

If you develop an application or need to integrate with Safewalk authentication mechanism you will first need to create a user with an access key so it can be used to call the authentication api. Once the access-key has been created you can use it to query the Safewalk server on its authentication API to authenticate users on the system using your choice of programming language.

Follow the steps below to generate an access token:

- Open a browser and sign-in to the Safewalk super-admin console (*https://SAFEWALK_ADDRESS:8443*).
- In the *Internal users & groups* box click the *Users* link and then the *Add user* button at the top right of the screen to create a new internal user.

Note: The password field is required however it is not used by the plug-in for this account so it is recommended to enter a very long and cumbersome password (there is no need to remember it or store it anywhere).

- Click the *Save* button and you will be redirected to a new page.
- In the 'User Type' section, check the box for 'System user' so that this user will be flagged as a system user and will not be manageable from the management console.
- In the 'Permissions' section, remove the user from any group that appear by clicking the 'Remove all'button under the 'Groups' combo box.
- In the 'Access Tokens' section select 'Create Access Token'
- **Copy the value of the access token that was generated** - You will need it for the configuration of the plug-in/application.

Note: This parameter will be sent to the Safewalk authentication api to identify the application account every time an authentication request is sent.

- Click *Save*

1.2 API description

`API Reference:`
Method: `POST`

API Path: `/api/v1/auth/authenticate/`

To make sure you are able to communicate with the authentication API it is advisable to start by querying the server using a simple command line tool.

Below is an example of such a query using the *curl* command:

```
curl --insecure -X POST -H "Authorization: Bearer ACCESS_KEY"
--data "username=USER_TO_AUTHENTICATE&password=CODE_TO_VERIFY"
https://SERVER_ADDRESS:PORT/api/v1/auth/authenticate/
```

The parameters values that are provided in the API call are:

ACCESS_KEY
The access-key of the application account.

USER_TO_AUTHENTICATE
The username to authenticate.

Note: This is not the application account username but any user that appears on Safewalk.

CODE_TO_VERIFY
The password/code/OTP of the user.

SERVER_ADDRESS
The server address that will receive the authentication request.

Note: If you are using the authentication API service on the Safewalk Gateway you will need to provide the Safewalk Gateway address here.

If you are querying the authentication API directly on Safewalk you will need to provide the Safewalk server address here.

PORT
The port the authentication api is listening on (443 for the Safewalk Gateway and 8445 for the Safewalk server).

Below is a sample valid query using the *curl* command:

```
curl --insecure -X POST -H "Authorization: Bearer 5bf43a08c50ac784d858676287674d20"
--data "username=demo&password=87619981"
https://192.168.223.29:8445/api/v1/auth/authenticate/
```

1.3 API response

The API returns two main HTTP codes:

- `200 OK` - When the authentication was completed successfully and the user should be granted access (in this case the `code` parameter in the json body will contain the value `ACCESS_ALLOWED`).
- `401 UNAUTHORIZED` - When the authentication did not complete successfully and the user either failed to be authenticated (in this case the `code` parameter in the json body will contain the value `ACCESS_DENIED`) or is expected to enter additional input (in this case the `code` parameter in the json body will contain the value `ACCESS_CHALLENGE` - for example after a valid password was entered and the user is expected to enter an OTP).

When the call to the API returns with `HTTP CODE : 200 OK` the following parameters are expected:

```
{
  "code": "ACCESS_ALLOWED",
  "username": "FULL_USERNAME",
  "auth-method": "AUTHENTICATION_METHOD",
  "username_stripped": "STRIPPED_USERNAME",
  "transaction-id": "TRANSACTION_ID"
}
```

When the call to the API returns with `401 UNAUTHORIZED` and the `code` parameter in the json body contains the value `ACCESS_CHALLENGE`, the following parameters are expected:

```
{
  "code": "ACCESS_CHALLENGE",
  "username": "FULL_USERNAME",
  "auth-method": "AUTHENTICATION_METHOD",
  "username_stripped": "STRIPPED_USERNAME",
  "reply-message": "USER_REPLY_MESSAGE",
  "transaction-id": "TRANSACTION_ID"
}
```

When the call to the API returns with `401 UNAUTHORIZED` and the `code` parameter in the json body contains the value `ACCESS_DENIED`, the following parameters are expected:

```
{

  "code": "ACCESS_DENIED",
  "reply-message": "USER_REPLY_MESSAGE"
  "transaction-id": "TRANSACTION_ID",
}
```

code (ACCESS_ALLOWED | ACCESS_CHALLENGE | ACCESS_DENIED)
The value that is set in this parameter defines the authentication result:

Returned value	Description
ACCESS_ALLOWED	The authentication completed successfully.
ACCESS_CHALLENGE	The authentication did not complete and the user is expected to input additional code.
ACCESS_DENIED	The authentication was denied.

username (FULL_USERNAME)
The value that is set in this parameter contains the full username of the user in the form of `user@domain`.

auth-method (AUTHENTICATION_METHOD)
The value that is set in this parameter contains the authentication method that was used by the user:

- `FAST:AUTH:QR`
- `FAST:AUTH:PUSH`
- `PASSWORD:LDAP`
- `PASSWORD:LOCAL`
- `Virtual` (SMS/E-Mail)
- `Backup`
- `HOTP`
- `SwisscomMobileID`
- `TOTP`
- `TOTP:Flex`
- `TOTP:Flex:Hybrid`
- `TOTP:Flex:PIN`
- `TOTP:Flex:PIN:Hybrid`
- `TOTP:Hybrid`
- `TOTP:Mobile`
- `TOTP:Mobile:Hybrid`
- `...`

username_stripped (STRIPPED_USERNAME)
The value that is set in this parameter contains the simple username form (excluding the domain part).

reply-message (USER_REPLY_MESSAGE)
The value that is set in this parameter contains a message that can be displayed to the user (in case of challenge or authentication failure).

transaction-id (TRANSACTION_ID)
The transaction id inside the Safewalk server that can be used to trace the transaction at a later time in Safewalk's transaction log.

1.4 Calling the API from Java

A self-explanatory sample code/SDK that is implemented in Java can be found at https://github.com/altipeak/safewalk-java-sdk

---

Note:

- In case a certain API call is not implemented on the Safewalk server side it will return an http error 404.
- Make sure you set the correct values for the attributes *host*, *AUTHENTICATION_API_ACCESS_TOKEN* and *ADMIN_API_ACCESS_TOKEN*.
- Pay attention to change the username, tokens type and serial number to users and tokens that exist on your system.

1.5 Calling the API from .Net/CSharp

A self-explanatory sample code/SDK that is implemented in .Net/CSharp can be found at https://github.com/altipeak/safewalk-net-sdk

---

Note:

- In case a certain API call is not implemented on the Safewalk server side it will return an http error 404.
- Make sure you set the correct values for the attributes *host*, *AUTHENTICATION_API_ACCESS_TOKEN* and *ADMIN_API_ACCESS_TOKEN*.
- Pay attention to change the username, tokens type and serial number to users and tokens that exist on your system.

1.6 Calling the API from Python

A self-explanatory Python sample code/SDK can be found at https://github.com/altipeak/safewalk-flask-integration

---

Note: This sample uses *requests.py* which is a general python library.

---

2 Safewalk Second-Step authentication API

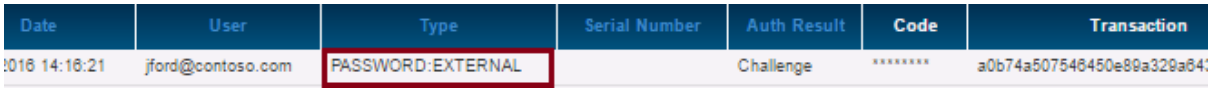
The Safewalk Second-Step authentication API "notifies" Safewalk that the user that is trying to authenticate has already authenticated with its first authentication factor (usually a password). When this API is called Safewalk will skip the first step validation and jump directly to the second-step part.

The following is the expected behavior when this API is used:

- The user is assigned with a Safewalk Fast Auth Push license - In this case Safewalk will send the user a Push notification and wait for the user to accept the transaction.

  Note: You can use this method to form a Passwordless experience for the user.

- The user is assigned with a Virtual license - In this case the Safewalk code will automatically be sent to the user when the API is called.
- The user is assigned with a 2-Step authentication device license - In this case the user will **not** be required to enter its password again before entering its Safewalk code.
- The user is not assigned with any license and is set to allow authentication with only a static password - In this case the user will be considered as authenticated and you will also notice a new entry with a value of `PASSWORD:EXTERNAL` in the `Type` column of the management console TRANSACTIONS view.



Date	User	Type	Serial Number	Auth Result	Code	Transaction
:016 14:16:21	jford@contoso.com	PASSWORD:EXTERNAL		Challenge	********	a0b74a507546450e89a329a643

Fig. 1: Management console TRANSACTIONS | PASSWORD:EXTERNAL

2.1 Creating an OAUTH token for the application account

If you develop an application or need to integrate with Safewalk authentication mechanism you will first need to create a user with an access key so it can be used to call the authentication api. Once the access-key has been created you can use it to query the Safewalk server on its authentication API to authenticate users on the system using your choice of programming language.

Follow the steps below to generate an access token:

- Open a browser and sign-in to the Safewalk super-admin console (*https://SAFEWALK_ADDRESS:8443*).
- In the *Internal users & groups* box click the *Users* link and then the *Add user* button at the top right of the screen to create a new internal user.

Note: The password field is required however it is not used by the plug-in for this account so it is recommended to enter a very long and cumbersome password (there is no need to remember it or store it anywhere).

- Click the *Save* button and you will be redirected to a new page.
- In the 'User Type' section, check the box for 'System user' so that this user will be flagged as a system user and will not be manageable from the management console.
- In the 'Permissions' section, remove the user from any group that appear by clicking the 'Remove all'button under the 'Groups' combo box.
- In the 'Access Tokens' section select 'Create Access Token'
- **Copy the value of the access token that was generated** - You will need it for the configuration of the plug-in/application.

Note: This parameter will be sent to the Safewalk authentication api to identify the application account every time an authentication request is sent.

- Click *Save*

2.2 API description

`API Reference:`

Method: `POST`

API Path: `/api/v1/auth/pswdcheckedauth/`

To make sure you are able to communicate with the authentication API it is advisable to start by querying the server using a simple command line tool.

Below is an example of such a query using the *curl* command:

```
curl --insecure -X POST -H "Authorization: Bearer ACCESS_KEY"
--data "username=USER_TO_AUTHENTICATE&exclude_hybrid=EXCLUDE_HYB_FUNC"
https://SERVER_ADDRESS:PORT/api/v1/auth/pswdcheckedauth/
```

The parameters values that are provided in the API call are:

ACCESS_KEY
The access-key of the application account.

USER_TO_AUTHENTICATE
The username to authenticate.

Note: This is not the application account username but any user that appears on Safewalk.

EXCLUDE_HYB_FUNC
A boolean value (`true` or `false`) that when set to `true` a user with a Hybrid type license will not receive a Safewalk code automatically but instead it will need to submit its static password in order to request a code (using the standard Safewalk input-based authentication API).

Note: This parameter should usually be set to `true` to prevent a situation where a user that is associated with a Hybrid type license is "annoyed" with Safewalk code messages that it does not intend to use.

SERVER_ADDRESS
The server address that will receive the authentication request.

Note: If you are using the authentication API service on the Safewalk Gateway you will need to provide the Safewalk Gateway address here.

If you are querying the authentication API directly on Safewalk you will need to provide the Safewalk server address here.

PORT
The port the authentication api is listening on (443 for the Safewalk Gateway and 8445 for the Safewalk server).

Below is a sample valid query using the *curl* command:

```
curl --insecure -X POST -H "Authorization: Bearer 5bf43a08c50ac784d858676287674d20"
--data "username=demo&exclude_hybrid=true"
https://192.168.223.29:8445/api/v1/auth/pswdcheckedauth/
```

2.3 API response

The API returns two main HTTP codes:

- `200 OK` - When the authentication was completed successfully and the user should be granted access (in this case the `code` parameter in the json body will contain the value `ACCESS_ALLOWED`).
- `401 UNAUTHORIZED` - When the authentication did not complete successfully and the user either failed to be authenticated (in this case the `code` parameter in the json body will contain the value `ACCESS_DENIED`) or is expected to enter additional input (in this case the `code` parameter in the json body will contain the value `ACCESS_CHALLENGE` - for example after a valid password was entered and the user is expected to enter an OTP).

When the call to the API returns with `HTTP CODE : 200 OK` the following parameters are expected:

```
{
  "code": "ACCESS_ALLOWED",
  "username": "FULL_USERNAME",
  "auth-method": "AUTHENTICATION_METHOD",
  "username_stripped": "STRIPPED_USERNAME",
  "transaction-id": "TRANSACTION_ID"
}
```

When the call to the API returns with `401 UNAUTHORIZED` and the `code` parameter in the json body contains the value `ACCESS_CHALLENGE`, the following parameters are expected:

```
{
  "code": "ACCESS_CHALLENGE",
  "username": "FULL_USERNAME",
  "auth-method": "AUTHENTICATION_METHOD",
  "username_stripped": "STRIPPED_USERNAME",
  "reply-message": "USER_REPLY_MESSAGE",
  "transaction-id": "TRANSACTION_ID"
}
```

When the call to the API returns with `401 UNAUTHORIZED` and the `code` parameter in the json body contains the value `ACCESS_DENIED`, the following parameters are expected:

```
{

  "code": "ACCESS_DENIED",
  "reply-message": "USER_REPLY_MESSAGE"
  "transaction-id": "TRANSACTION_ID",
}
```

code (ACCESS_ALLOWED | ACCESS_CHALLENGE | ACCESS_DENIED)
The value that is set in this parameter defines the authentication result:

Returned value	Description
ACCESS_ALLOWED	The authentication completed successfully.
ACCESS_CHALLENGE	The authentication did not complete and the user is expected to input additional code.
ACCESS_DENIED	The authentication was denied.

username (FULL_USERNAME)
The value that is set in this parameter contains the full username of the user in the form of `user@domain`.

auth-method (AUTHENTICATION_METHOD)
The value that is set in this parameter contains the authentication method that was used by the user:

- `FAST:AUTH:QR`
- `FAST:AUTH:PUSH`
- `PASSWORD:LDAP`
- `PASSWORD:LOCAL`
- `Virtual` (SMS/E-Mail)
- `Backup`
- `HOTP`
- `SwisscomMobileID`
- `TOTP`
- `TOTP:Flex`
- `TOTP:Flex:Hybrid`
- `TOTP:Flex:PIN`
- `TOTP:Flex:PIN:Hybrid`
- `TOTP:Hybrid`
- `TOTP:Mobile`
- `TOTP:Mobile:Hybrid`
- ...

username_stripped (STRIPPED_USERNAME)
The value that is set in this parameter contains the simple username form (excluding the domain part).

reply-message (USER_REPLY_MESSAGE)
The value that is set in this parameter contains a message that can be displayed to the user (in case of challenge or authentication failure).

transaction-id (TRANSACTION_ID)
The transaction id inside the Safewalk server that can be used to trace the transaction at a later time in Safewalk's transaction log.

3 Safewalk Fast Auth Passwordless, 2-Way, Anonymous and Input-less (2WAIL) authentication API

Safewalk Fast Auth authenticator includes an authentication method that allows a smooth authentication experience for users to access a protected website using a browser.

This authentication method provides the following security and usage benefits:

- **2-Ways authentication** - Safewalk authenticates to the user and only then the user authenticates to Safewalk.
- **Input-less** - The user does not need to key in any username, password or code.
- **Anonymous** - The user identity is disclosed to the website only **after** the user has **successfully** authenticated to Safewalk. As oppose to standard authentication flows, where even if the user fails to authenticate, knowledge of the user identity exist (for example the username of the user who tried to access).

3.1 The Safewalk Fast Auth 2WAIL authentication end user flow

The Safewalk Fast Auth 2WAIL authentication experience is provided in two forms:

- **QR code** - In this form, the user is presented with a QR code that it scans using its registered mobile Application. This form of authentication is usefull for cases where the user authenticates to a website while outside of its registered mobile device (i.e. the browser to access the website is openned on another device - a pc, a laptop, etc.).

  When this method is used the end user experience will look like this:

Fig. 2: Safewalk Fast Auth Mobile - QR user flow

1. Go to website.
2. Launch and identify to Safewalk mobile application.
3. Scan QR code.
4. Receive access.

- **Deep linking** - In this form, the user is presented with a special link that when clicked opens its registered mobile Application.

  This form of authentication is useful for cases where the user authenticates to a website directly from its registered mobile device (i.e. the browser to access the website is opened on the same device the registered Safewalk Fast Auth authenticator application is installed on).

  When this method is used the end user experience will look like this:

Fig. 3: Safewalk Fast Auth Mobile - 1-Click deep-link mobile access user flow

1. Go to website using a mobile browser and click the login button.

Note: The same scenario can be used where instead of using a mobile browser the login button is embedded into another mobile application.

> **Attention:** This step should be performed on the same device where the Safewalk Fast Auth app is installed, or the Safewalk mobile applications technology is used within another branded or embedded mobile application.

2. Confirm transaction on Safewalk Fast Auth app (fingerprint, FaceID, PIN, pattern, etc.).
3. Receive access.

Note: It is possible to combine the above two methods in a single page where if the website detects that the user is coming from a mobile-based browser it will display the deep-link for the user to press, and if the website detects that the request is coming from a standard (non-mobile) browser it will display a page with a QR code for the user to scan using its Safewalk Fast Auth authenticator.

3.2 The Safewalk Fast Auth 2WAIL authentication overall workflow

The Safewalk Fast Auth 2WAIL authentication has the following general workflow:

1. The user navigates to the website.
2. The website calls Safewalk to request a session key.
3. Safewalk verifies the authenticity of the calling website, generates a random session key and store it in a temporary table.

Note: The session key that is generated by Safewalk:

- Is anonymous and is generated by Safewalk without knowing the identity of the user.
- Contains a unique digital signature of the Safewalk server that will later be checked by the application to verify the authenticity of the server. Thus providing two way authentication - The user will sign the authentication transaction and send it to Safewalk only after verifying the authenticity of the signature that was sent by the Safewalk server.

4. The session code is sent back to the calling website.
5. The calling website presents the session code to the user (in either QR code or deep-link format)
6. The Safewalk Fast Auth authenticator mobile application is launched.
7. The user authenticates to the application and the session code is signed using the user private key.
8. The signed session code is sent to Safewalk for verification.

Note: This communication is done out-of-band, from the user application directly to the Safewalk server without passing through the website.

9. The signed session code is verified by Safewalk and the validity of the session along with the user identifier is written to the temporary session code table.
10. The website queries Safewalk for the validity of the session code and Safewalk replies with the validity and user identifier data so that the website can grant the user access.

When used with the QR code end user flow method the general workflow will be the following:

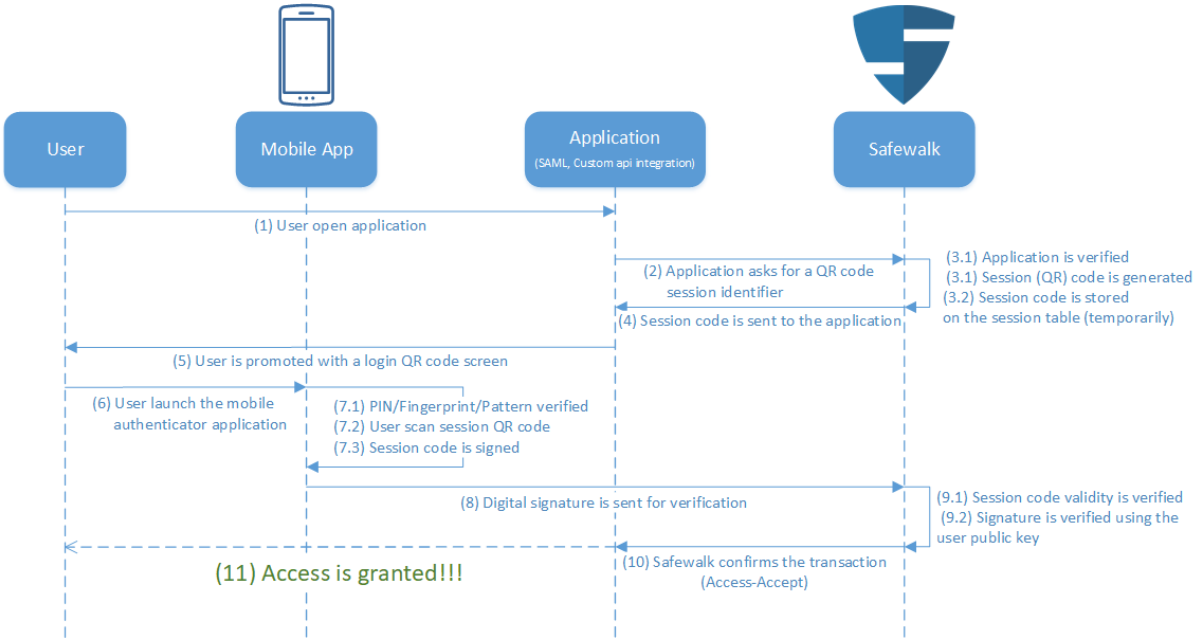


Fig. 4: Safewalk Fast Auth Mobile - QR overall flow

When used with the deep-link end user flow method the general workflow will be the following:

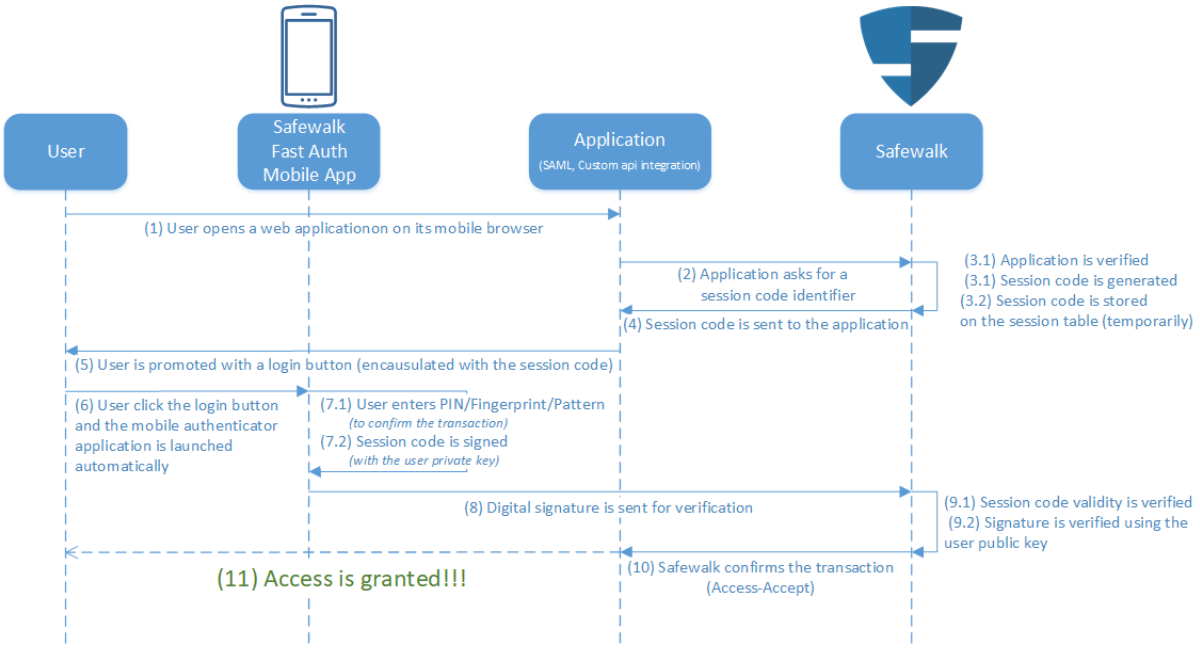


Fig. 5: Safewalk Fast Auth Mobile - 1-Click deep-link mobile access method

An application/website that would like to implement the above authentication scenario with Safewalk would need to implement the following 3 steps:

- Step 1 - Request a session key from Safewalk by calling the session key API (step 2 in the diagram)
- Step 2 - Encapsulate the key into a QR code or a deep-link (step 5 in the diagram).
- Step 3 - Call a Safewalk API to check periodically if the transaction for the session code was accepted (step 10 in the diagram).

3.3 Creating an OAUTH token for the application account

If you develop an application or need to integrate with Safewalk authentication mechanism you will first need to create a user with an access key so it can be used to call the authentication api. Once the access-key has been created you can use it to query the Safewalk server on its authentication API to authenticate users on the system using your choice of programming language.

Follow the steps below to generate an access token:

- Open a browser and sign-in to the Safewalk super-admin console (*https://SAFEWALK_ADDRESS:8443*).
- In the *Internal users & groups* box click the *Users* link and then the *Add user* button at the top right of the screen to create a new internal user.

Note: The password field is required however it is not used by the plug-in for this account so it is recommended to enter a very long and cumbersome password (there is no need to remember it or store it anywhere).

- Click the *Save* button and you will be redirected to a new page.
- In the ‘User Type‘ section, check the box for ‘System user‘ so that this user will be flagged as a system user and will not be manageable from the management console.
- In the ‘Permissions‘ section, remove the user from any group that appear by clicking the ‘Remove all‘button under the ‘Groups‘ combo box.

- In the ‘Access Tokens‘ section select ‘Create Access Token‘
- **Copy the value of the access token that was generated** - You will need it for the configuration of the plug-in/application.

Note: This parameter will be sent to the Safewalk authentication api to identify the application account every time an authentication request is sent.

- Click *Save*

3.4 Step 1: Request a session key (server side code):

Request a session key from Safewalk by calling the session key API:

`API Reference:`

API to request a session key from Safewalk.

Method: *POST*

API Path: */api/v1/auth/session_key/*

Below is an example of such a query using the *curl* command:

```
curl --insecure -X POST -H "Authorization: Bearer ACCESS_KEY" https://SERVER_ADDRESS:PORT/api/v1/auth/
↪session_key/
```

The parameters values that are provided in the API call are:

`ACCESS_KEY`
The access-key of the application account.

`SERVER_ADDRESS`
The server address that will receive the authentication request.

Note: If you are using the authentication API service on the Safewalk Gateway you will need to provide the Safewalk Gateway address here.

If you are querying the authentication API directly on Safewalk you will need to provide the Safewalk server address here.

`PORT`
The port the authentication api is listening on (443 for the Safewalk Gateway and 8445 for the Safewalk server).

Below is sample valid query using the *curl* command:

```
curl --insecure -X POST -H "Authorization: Bearer c02ec58b3f1b5ae872cc0f67e811490e0fa0efab" https://192.
↪168.0.5:8445/api/v1/auth/session_key/
```

API response

When the call to the API returns with `HTTP CODE : 200 OK` the following parameters are expected:

```
{
  "session-key": "SESSION_KEY"
}
```

`SESSION_KEY`
A unique, random and identity-free session identifier that will be used to query the server for the session authentication status and for a matching authenticated user after it has successfully authenticated.

An example of such json output is shown below:

```
{"session-key": "B157DD60289EA6481D289EEA54...28C6265F1B"}
```

3.5 Step 2: Encapsulate the session key (server side code):

In this step you would need to encapsulate the session-key that was returned in the previous step into a QR code or a deep-link url so that it can be read by the mobile application and processed for further signature and validation.

3.6 Step 3: Check the authentication status of the session key (server side code):

Call Safewalk to check periodically if the transaction for the session code was accepted

API Reference:

API to check if the transaction for the session code was accepted.

Method: *GET*

API Path: *api/v1/auth/session_key/<challenge>*

Below is an example of such a query using the *curl* command:

```
curl --insecure -X GET -H "Authorization: Bearer ACCESS_KEY" https://SERVER:PORT/api/v1/auth/session_key/
↪SESSION_KEY/
```

The parameters values that are provided in the API call are:

ACCESS_KEY
: The access-key of the application account.

SERVER_ADDRESS
: The server address that will receive the authentication request.

Note: If you are using the authentication API service on the Safewalk Gateway you will need to provide the Safewalk Gateway address here.

If you are querying the authentication API directly on Safewalk you will need to provide the Safewalk server address here.

PORT
: The port the authentication api is listening on (443 for the Safewalk Gateway and 8445 for the Safewalk server).

SESSION_KEY
: The session-key that was returned in step 1.

Below is sample valid query using the *curl* command:

```
curl --insecure -X GET -H "Authorization: Bearer c02ec58b3f1b5ae872cc0f67e811490e0fa0efab"
https://192.168.0.5:8445/api/v1/auth/session_key/B157DD60289EA6481D289EEA54...28C6265F1B/
```

API response

When the call to the API returns with `HTTP CODE : 200 OK` and the transaction was successfully completed, you will get the following output:

```
{
  "code": "ACCESS_ALLOWED",
  "username": "FULL_USERNAME",
  "auth-method": "AUTHENTICATION_METHOD",
  "username_stripped": "STRIPPED_USERNAME",
  "transaction-id": "TRANSACTION_ID"
}
```

For example:

```
{"transaction-id": "37dc87e056cc49d392ab5a4359f0a433", "username": "test2@directaccess.domain", "code":
↪"ACCESS_ALLOWED"}
```

Note: Calling the api again with the same parameters will not result in the same response since the session code will be deleted from the system after it has been used and claimed.

If there was no reply yet and the session timeout didn't pass, you will get the following output:

```
{"code": "ACCESS_PENDING"}
```

If the session timeout has passed but the session key was not yet deleted, you will get the following output:

```
{"code": "NO_RESPONSE"}
```

If the timeout has passed and the session key has expired you will get an error 404:

```
HTTP/1.1 404 NOT FOUND
```

code (ACCESS_ALLOWED | ACCESS_PENDING | NO_RESPONSE)
The value that is set in this parameter defines the authentication result:

Returned value	Description
ACCESS_ALLOWED	The authentication completed successfully.
ACCESS_PENDING	There was no reply yet and the session timeout didn't pass.
NO_RESPONSE	The session timeout has passed but the session key was not yet deleted.

username (FULL_USERNAME)
The value that is set in this parameter contains the full username of the user who has authenticated in the form of `user@domain`.

auth-method (AUTHENTICATION_METHOD)
The value that is set in this parameter contains the authentication method that was used by the user.

Note: The expected value when using this api will typically be `FAST:AUTH:QR`.

---

username_stripped (STRIPPED_USERNAME)
The value that is set in this parameter contains the simple username form (excluding the domain part).

transaction-id (TRANSACTION_ID)
The transaction id inside the Safewalk server that can be used to trace the transaction at a later time in Safewalk's transaction log.