Authentication API configuration

Updated December 10, 2019

The SecureAuth Authentication API embeds the SecureAuth IdP functionality into a custom application, enabling flexible workflow configurations and user interfaces.

Using a RESTful API encrypted over Secure Sockets Layer (SSL), SecureAuth IdP can:

- validate
  - user IDs
  - passwords
  - PINs
  - soft tokens
  - knowledge-based answers
  - Push-to-Accept responses; note the following
    - Symbol-to-Accept or Push-to-Accept and biometric and are available in SecureAuth® Identity Platform version 19.07 or later.
    - Depending on how the administrator configures Identity Platform, either Push-to-Accept or Symbol-to-Accept is available. For example, an organization can have Push-to-Accept and biometric or Symbol-to-Accept and biometric, depending on the Identity Platform configuration.
    - Teams must download the Authenticate app on their mobile device because both Push-to-Accept and Symbol-to-Accept are initiated through the Authenticate app.
- generate One-time Passwords (OTPs) delivered by
  - phone call
  - SMS message
  - email message
  - help desk
  - Push / Push-to-Accept Notification
- analyze a user's access attempt through the SecureAuth
  - Device / Browser Fingerprinting
  - Adaptive Authentication
  - Behavioral Biometric profile
- evaluate IP address risk through threat intelligence data
- prevent end users from logging on a realm

Each SecureAuth IdP realm can host its own uniquely configured Authentication API, so admins can enable various workflows and registration methods.

By integrating an application with the SecureAuth Authentication API, enabling Multi-Factor Authentication mechanisms, and configuring Adaptive Authentication, admins can securely direct users through unique logins and interfaces without leaving the application.

This document is for SecureAuth IdP v9.3.

Information about using Identity Management API tools is in the Identity Management API Guide.

Information about configuring the Login for Windows API endpoint is in the Login for Windows v19.06 configuration guide.

The SecureAuth public GitHub link is https://github.com/SecureAuthCorp/. Use the link to access sample API SDK files for your applications.

Prerequisites

- SecureAuth IdP v9.3
- Symbol-to-Accept or Push-to-Accept and biometric and are available in SecureAuth® Identity Platform version 19.07 or later; end users must download the Authenticate app on their mobile device
- Access to the application code
- On-premises directory that SecureAuth IdP can integrate with
- New realm or access to existing realm on SecureAuth IdP v9.1 or later where the Authentication API will be enabled

  The API can be included in any realm with any post-authentication event as long as the appropriate directory is integrated and the necessary features are configured, based on the endpoints being used.

- Configure the Data Tab in the SecureAuth IdP Web Admin.

  A directory integration is required for SecureAuth IdP to pull user profile information during the login process.

The Behavioral Biometrics feature only supports LDAP directory integrations, while other Authentication API features support most directory type integrations.

OPTIONAL: Prevent Threat Service license – to use the machine learning user risk score analysis feature in advanced adaptive capability powered by machine learning – contact SecureAuth Support to upgrade the SecureAuth IdP license.

On the version 9.3 New Experience user interface, you can configure an Active Directory integration or SQL Server integration and apply it to applications made from App onboarding library templates. Configure the remaining components – for example, Workflow, Multi-Factor Methods, and Adaptive Authentication tabs – on the Classic Experience user interface.

Configure SecureAuth IdP Web Admin

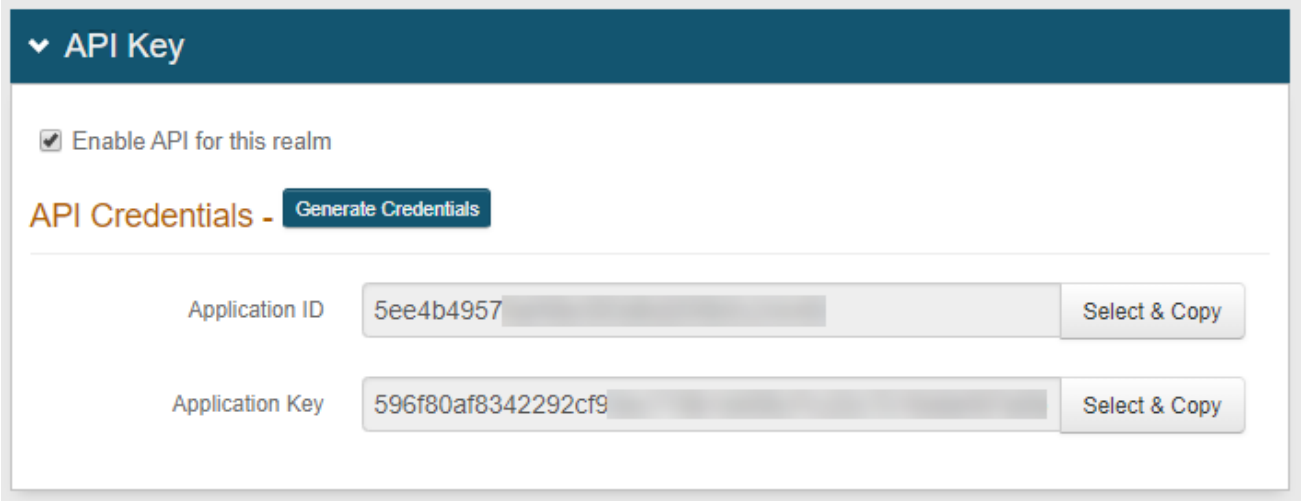
Only API steps are required; all other Web Admin steps are optional and should be performed based on the features you want to implement.

Make the changes to the following sections in the **API** tab.

API Key section

1. In the **API** tab, go to the **API Key** section.

2. Check **Enable API for this realm**.



2. Click **Generate Credentials** to create a new **Application ID** and **Application Key.**

   The **Application ID** and **Application Key** are unique for each realm.

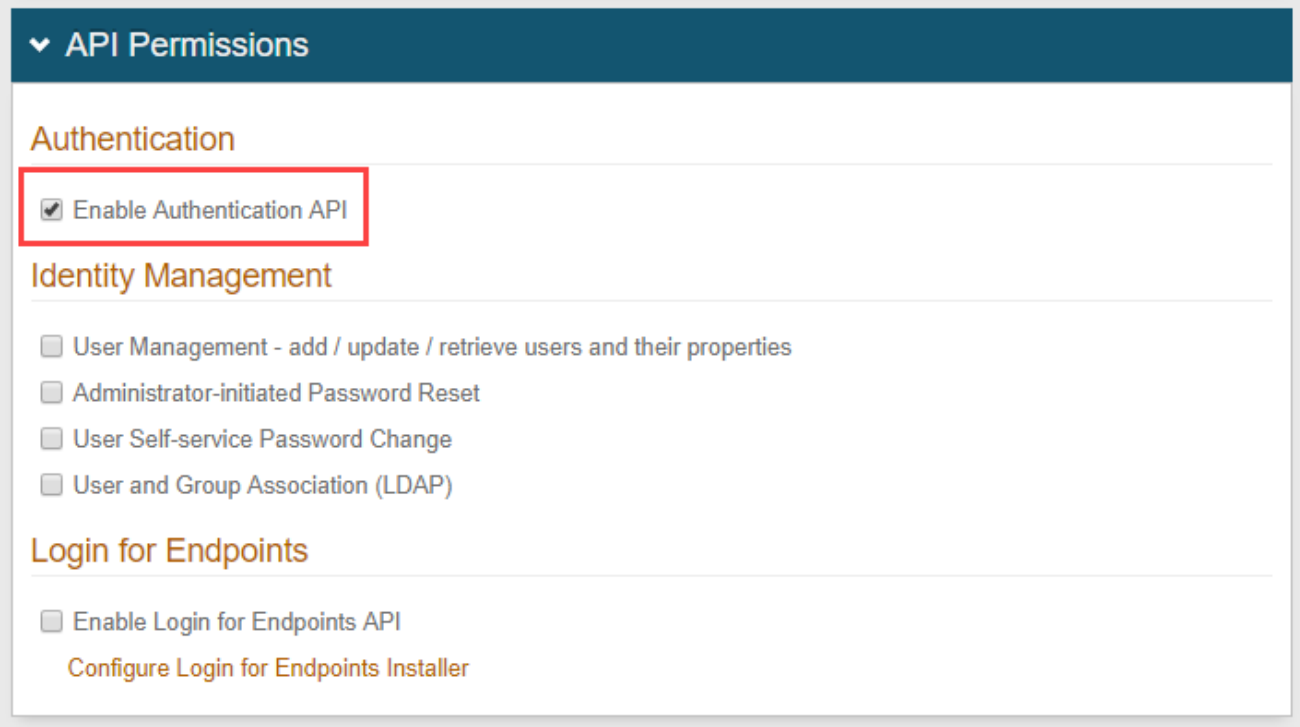
   The API key looks like it comprises **64** random characters, but it actually comprises **32** two-character base-16 hexadecimal values.

   This is important to note when using the API key to produce the HMAC hash.

3. Click **Select & Copy** to copy the contents from the fields

   These values are required in the Header configuration steps below.

4. In the **API Permissions** section, check **Enable Authentication API.**



5. If your team is using SecureAuth RADIUS 2.4.15 or later, you must check **User Management**. This setting enables the SecureAuth IdP API to connect to User properties.

6. Save the configuration.

Configure Request Header

Authentication against an API requires an HTTP basic authorization header and Content-Type header.

1. Add a **Content-Type** header with a value of **application/json**.

2. Create an Authorization Header for all GET and POST requests by following the steps below. See the code examples for SecureAuth IdP v9.1 or SecureAuth IdP v9.2 or later.

Authorization Header

Set up the authorization header for GET and POST endpoints.

GET endpoint

a. Build a string based on the request.

- METHOD (GET)
- DATE/TIME

Header types	IdP version	String requirements
Date	v9.1+	second-precision timestamp
X-SA-Date (custom)	v9.1+	second-precision timestamp
X-SA-Ext-Date (custom)	v9.2+	millisecond-precision timestamp

- APPLICATION ID (from SecureAuth IdP Web Admin)
- PATH (API endpoint) – e.g. /secureauth2/api/v1/users/{userID}

b. Create an HMAC SHA256 hash of step 1 using the **Application Key** (from SecureAuth IdP Web Admin).

This step is executed by calling the HMAC and producing the hash value.

c. Encode the HMAC SHA256 hash from step 2 in Base64.

d. Concatenate the **"Application ID"**, **":"**, and the **"Base64 encoded HMAC SHA256 hash"** from step 3.

ApplicationID:Base64EncodedHMACSHA256Hash

e. Encode the value from step 4 in Base64.

f. Concatenate **"Basic"** and the **"Value of Step 5"**.

```
Basic Step5Value
```

GET Request Example for SecureAuth IdP v9.1

```
Step 1
    GET
    Wed, 08 Apr 2015 21:37:33 GMT
    1b700d2e7b7b4abfa1950c865e23e81a
    /secureauth2/api/v1/users/jsmith/factors

End Result: "GET\nWed, 08 Apr 2015 21:37:33 GMT\n1b700d2e7b7b4abfa1950c865e23e81a\n/secureauth2/api/v1/users
/jsmith/factors"

Step 2
        Non-printable bytes are produced in this step

Step 3
    F5yqdLDJddUYOlrpBlOJBh/YCUIMVCsWejuhiCrqMmw=

Step 4
    1b700d2e7b7b4abfa1950c865e23e81a:F5yqdLDJddUYOlrpBlOJBh/YCUIMVCsWejuhiCrqMmw=

Step 5

MWI3MDBkMmUtN2I3Yi00YWJmLWExOTUtMGM4NjVlMjNlODFhOnorVGNYNG4vbFlsTmNvNjRpQkRENVJKaHFiZ0h0UGYwaEQ4d1d4bTgvWVk9

Step 6
    Basic
MWI3MDBkMmUtN2I3Yi00YWJmLWExOTUtMGM4NjVlMjNlODFhOnorVGNYNG4vbFlsTmNvNjRpQkRENVJKaHFiZ0h0UGYwaEQ4d1d4bTgvWVk9

End Result:
    Method: GET,
    RequestUri: 'https://secureauth.company.com/secureauth2/api/v1/users/jsmith/factors',
    Version: 1.1,
    Headers: {
        Connection: Keep-Alive
        Date: Wed, 08 Apr 2015 21:37:33 GMT
        Authorization: Basic
MWI3MDBkMmUtN2I3Yi00YWJmLWExOTUtMGM4NjVlMjNlODFhOkY1eXFkTERKZGRVWU9scnBCbE9KQmgvWUNVSU1WQ3NXZWp1aGlDcnFNbXc9

        Host: secureauth.company.com
        Content-Length: 0
    }
```

GET Request Example for SecureAuth IdP v9.2+

```
Step 1
    GET
    Wed, 08 Apr 2015 21:37:33.123 GMT
    1b700d2e7b7b4abfa1950c865e23e81a
    /secureauth2/api/v1/users/jsmith/factors

End Result: "GET\nWed, 08 Apr 2015 21:37:33.123 GMT\n1b700d2e7b7b4abfa1950c865e23e81a\n/secureauth2/api/v1
/users/jsmith/factors"

Step 2
        Non-printable bytes are produced in this step

Step 3
    F5yqdLDJddUYOlrpBlOJBh/YCUIMVCsWejuhiCrqMmw=

Step 4
    1b700d2e7b7b4abfa1950c865e23e81a:F5yqdLDJddUYOlrpBlOJBh/YCUIMVCsWejuhiCrqMmw=

Step 5

MWI3MDBkMmUtN2I3Yi00YWJmLWExOTUtMGM4NjVlMjNlODFhOnorVGNYNG4vbFlsTmNvNjRpQkRENVJKaHFiZ0h0UGYwaEQ4d1d4bTgvWVk9

Step 6
    Basic
MWI3MDBkMmUtN2I3Yi00YWJmLWExOTUtMGM4NjVlMjNlODFhOnorVGNYNG4vbFlsTmNvNjRpQkRENVJKaHFiZ0h0UGYwaEQ4d1d4bTgvWVk9

End Result:
    Method: GET,
    RequestUri: 'https://secureauth.company.com/secureauth2/api/v1/users/jsmith/factors',
    Version: 1.1,
    Headers: {
        Connection: Keep-Alive
        X-SA-Ext-Date: Wed, 08 Apr 2015 21:37:33.123 GMT
        Authorization: Basic
MWI3MDBkMmUtN2I3Yi00YWJmLWExOTUtMGM4NjVlMjNlODFhOkY1eXFkTERKZGRVWU9scnBCbE9KQmgvWUNVSU1WQ3NXZWp1aGlDcnFNbXc9

        Host: secureauth.company.com
        Content-Length: 0
    }
```

POST endpoint

g. Build a string based on the request.

- METHOD (POST)
- DATE/TIME

Header types	IdP version	String requirements
Date	v9.1+	second-precision timestamp
X-SA-Date (custom)	v9.1+	second-precision timestamp
X-SA-Ext-Date (custom)	v9.2+	millisecond-precision timestamp

- APPLICATION ID (from SecureAuth IdP Web Admin)
- PATH (API endpoint) – e.g. /secureauth2/api/v1/users/{userID}
- CONTENT (JSON parameters)

h. Create an HMAC SHA256 hash of step 1 using the **Application Key** (from SecureAuth IdP Web Admin).

This step is executed by calling the HMAC and producing the hash value.

i. Encode the HMAC SHA256 hash from step 2 in Base64.

j. Concatenate the **"Application ID"**, **":"**, and the **"Base64 encoded HMAC SHA256 hash"** from step 3.

ApplicationID:Base64EncodedHMAC256Hash

k. Encode the value from step 4 in Base64.

I. Concatenate **"Basic "** and the **"Value of Step 5"**.

```
Basic Step5Value
```

POST Request Example for SecureAuth IdP v9.1

```
Step 1
    POST
    Wed, 08 Apr 2015 21:27:30 GMT
    1b700d2e7b7b4abfa1950c865e23e81a
    /secureauth2/api/v1/auth
    {"user_id":"jsmith","type":"user_id"}

End Result: "POST\nWed, 08 Apr 2015 21:27:30 GMT\n1b700d2e7b7b4abfa1950c865e23e81a\n/secureauth2/api/v1
/auth\n{"user_id":"jsmith","type":"user_id"}"

Step 2
        Non-printable bytes are produced in this step

Step 3
    D6nkepAEtk/M+cpkyWQ/hZMXZxPJ32L++5ZZa6+pB8U=

Step 4
    1b700d2e7b7b4abfa1950c865e23e81a:D6nkepAEtk/M+cpkyWQ/hZMXZxPJ32L++5ZZa6+pB8U=

Step 5

MWI3MDBkMmUtN2I3Yi00YWJmLWExOTUtMGM4NjVlMjNlODFhOkQ2bmtlcEFFdGsvTStjcGt5V1EvaFpNWFp4UEozMkwrKzVaWmE2K3BCOFU9

Step 6
    Basic
MWI3MDBkMmUtN2I3Yi00YWJmLWExOTUtMGM4NjVlMjNlODFhOkQ2bmtlcEFFdGsvTStjcGt5V1EvaFpNWFp4UEozMkwrKzVaWmE2K3BCOFU9

End Result:
    Method: POST
    RequestUri: 'https://secureauth.company.com/secureauth2/api/v1/auth'
    Version: 1.1
    Headers: {
        Connection: Keep-Alive
        Date: Wed, 08 Apr 2015 21:27:30 GMT
        Authorization: Basic
MWI3MDBkMmUtN2I3Yi00YWJmLWExOTUtMGM4NjVlMjNlODFhOkQ2bmtlcEFFdGsvTStjcGt5V1EvaFpNWFp4UEozMkwrKzVaWmE2K3BCOFU9

        Expect: 100-continue
        Host: secureauth.company.com
        Content-Length: 36
        Content-Type: application/json; charset=utf-8
    }
```

POST Request Example for SecureAuth IdP v9.2+

```
Step 1
    POST
    Wed, 08 Apr 2015 21:27:30.123 GMT
    1b700d2e7b7b4abfa1950c865e23e81a
    /secureauth2/api/v1/auth
    {"user_id":"jsmith","type":"user_id"}

End Result: "POST\nWed, 08 Apr 2015 21:27:30.123 GMT\n1b700d2e7b7b4abfa1950c865e23e81a\n/secureauth2/api/v1
/auth\n{"user_id":"jsmith","type":"user_id"}"

Step 2
        Non-printable bytes are produced in this step

Step 3
    D6nkepAEtk/M+cpkyWQ/hZMXZxPJ32L++5ZZa6+pB8U=

Step 4
    1b700d2e7b7b4abfa1950c865e23e81a:D6nkepAEtk/M+cpkyWQ/hZMXZxPJ32L++5ZZa6+pB8U=

Step 5

MWI3MDBkMmUtN2I3Yi00YWJmLWExOTUtMGM4NjVlMjNlODFhOkQ2bmtlcEFFdGsvTStjcGt5V1EvaFpNWFp4UEozMkwrKzVaWmE2K3BCOFU9

Step 6
    Basic
MWI3MDBkMmUtN2I3Yi00YWJmLWExOTUtMGM4NjVlMjNlODFhOkQ2bmtlcEFFdGsvTStjcGt5V1EvaFpNWFp4UEozMkwrKzVaWmE2K3BCOFU9

End Result:
    Method: POST
    RequestUri: 'https://secureauth.company.com/secureauth2/api/v1/auth'
    Version: 1.1
    Headers: {
        Connection: Keep-Alive
        X-SA-Ext-Date: Wed, 08 Apr 2015 21:27:30.123 GMT
        Authorization: Basic
MWI3MDBkMmUtN2I3Yi00YWJmLWExOTUtMGM4NjVlMjNlODFhOkQ2bmtlcEFFdGsvTStjcGt5V1EvaFpNWFp4UEozMkwrKzVaWmE2K3BCOFU9

        Expect: 100-continue
        Host: secureauth.company.com
        Content-Length: 36
        Content-Type: application/json; charset=utf-8
    }
```

Authorization Header non-validation responses...

When an Authorization Header cannot be validated, one of the following responses is returned:

`{ "status": "invalid", "message": "Missing authentication header.", }`	`{ "status": "invalid", "message": "Unknown authentication scheme.", }`
`{ "status": "invalid", "message": "Clock skew of message is outside threshold.", }`	`{ "status": "invalid", "message": "AppId is unknown.", }`
`{ "status": "invalid", "message": "Authentication header value is empty.", }`	`{ "status": "invalid", "message": "Authentication header has been seen before.", }`
`{ "status": "invalid", "message": "Authentication header value's format should be 'appId:hash'.", }`	`{ "status": "invalid", "message": "Invalid credentials.", }`

3. Optional: If using the **Email** two-factor authentication method and a language different than **US English**, create an **Accept-Language** header to generate the Email OTP messages in the preferred language.

    If no **Accept-Language** header is present, the Email OTP messages default to **US English**.

Configure Response Header

SecureAuth's API includes a security hashing enhancement that ensures the integrity of the information being sent in all of the endpoints' responses from the appliance to the application.

Through a hashing algorithm, SecureAuth IdP delivers a signature that can be validated by the application to ensure that no data manipulation has occurred prior to the application consuming the data.

Before sending the response to the application (initiated by the endpoint request), SecureAuth IdP creates the signature and includes it in the Response Header (prepended by X-SA-SIGNATURE:). The application can then validate the response by hashing the date / time and content from the consumed response and the Application ID with the Application Key and compare the new hashed value with the X-SA-SIGNATURE value. The Application ID and Application Key are generated in SecureAuth IdP and connect the appliance with the application for each endpoint transaction.

Application Response Header

In the application's code, the following is required to validate the response header's signature.

1. Build a string based on the request.

    a. **X-SA-DATE** for a second-precision timestamp (from the SecureAuth IdP v.1+ response)

    b. **APPLICATION ID** (from SecureAuth IdP Web Admin)

    c. **CONTENT** (JSON Parameters from the SecureAuth IdP response)

2. Create an HMAC SHA256 hash of step 1 using the **Application Key** (from SecureAuth IdP Web Admin).

    This step is executed by calling the HMAC and producing the hash value.

3. Encode the HMAC SHA256 hash from step 2 in Base64.

4. Compare the HMAC SHA256 hash from step 3 to the **X-SA-SIGNATURE** value in the SecureAuth Response Header.

5. Consume the response based on the comparison result.

NOTE: If, after hashing the data, the value exactly matches the signature provided in the SecureAuth IdP response header, then the data has not been compromised; if the value does not match the response signature, then the data has been modified.

Optional: Configure X-SA-Ext-Date Header

The string section for **DATE/TIME** can be configured to use either the second-precision UTC time or the millisecond-precision format DateTime.

If using the millisecond-precision, the date string must be included in the X-SA-Ext-Date header.

The following is a sample of X-SA-Ext-Date code:

```
var dateMillis = request.Headers.Date.Value.UtcDateTime.ToString("ddd, dd MMM yyyy HH:mm:ss.fff G\\MT");
                request.Headers.Add("X-SA-Ext-Date", dateMillis);
                request.Headers.Remove("Date");
                var httpMethod = request.Method.Method;
                string uri = request.RequestUri.AbsolutePath;
                string content = null;
                if (request.Content != null)
                {
                    content = request.Content.ReadAsStringAsync().Result;
                }
                result = (string.IsNullOrEmpty(content)) ?
                    string.Join("\n", httpMethod, dateMillis, appId, uri) :
                    string.Join("\n", httpMethod, dateMillis, appId, uri, content);
```

Configure Endpoints

Configure the endpoints for the selected feature.

Feature	Endpoint	Methods	Functions
Profile Validation	/auth	POST	Validate end user information and generate OTPs for authentication
	/auth/{REF_ID}	GET	Check status of Push-to-Accept responses
Multi-Factor Authentication	/users/{username} /factors	GET	Access end user profile and respond with Multi-Factor Authentication selections
Multi-Factor Throttling Authentication	/users/{username} /throttle	GET	Retrieve current count of Multi-Factor Authentication attempts for given username
	/users/{username} /throttle	PUT	Reset count of Multi-Factor Authentication attempts to 0
Phone Profiling Service Authentication	/numberprofile	POST	Retrieve phone profile record from data provider / directory which includes carrier information
	/numberprofile	PUT	Update the directory to reflect change in phone profile record
Device Recognition Authentication	/dfp/js	GET	Retrieve JavaScript reference required to generate fingerprints
	/dfp/validate	POST	Compare presented fingerprint with those stored in end user profile
	/dfp/confirm	POST	Create / update fingerprint in end user profile in directory
Adaptive Authentication	/adaptauth	POST	Analyze end user profile, group, IP address, country, geo-velocity, and risks detected by threat intelligence data
	/accesshistory	POST	Create end user access history for geo-velocity calculations
IP Evaluation	/ipeval	POST	Evaluate an IP address for risk factors based on threat intelligence data
Behavioral Biometrics Authentication	/behavebio/js	GET	Retrieve JavaScript reference required to gather and analyze end user behavioral biometric profile
	/behavebio	POST	Collect and create end user behavioral biometric profile for analyzing profile account info subsequently posted
	/behavebio	PUT	Reset end user profile to enable retraining

Related documentation

Symbol-to-Accept API endpoints

Biometric API endpoints

API configuration guide

Authentication API guide

Multi-factor Authentication API guide