Developer Guide

Advanced Authentication

Version 5.3

Legal Notices

Contents

About NetIQ Corporation

We are a global, enterprise software company, with a focus on the three persistent challenges in your environment: Change, complexity and risk—and how we can help you control them.

Our Viewpoint

Adapting to change and managing complexity and risk are nothing new

In fact, of all the challenges you face, these are perhaps the most prominent variables that deny you the control you need to securely measure, monitor, and manage your physical, virtual, and cloud computing environments.

Enabling critical business services, better and faster

We believe that providing as much control as possible to IT organizations is the only way to enable timelier and cost effective delivery of services. Persistent pressures like change and complexity will only continue to increase as organizations continue to change and the technologies needed to manage them become inherently more complex.

Our Philosophy

Selling intelligent solutions, not just software

In order to provide reliable control, we first make sure we understand the real-world scenarios in which IT organizations like yours operate—day in and day out. That's the only way we can develop practical, intelligent IT solutions that successfully yield proven, measurable results. And that's so much more rewarding than simply selling software.

Driving your success is our passion

We place your success at the heart of how we do business. From product inception to deployment, we understand that you need IT solutions that work well and integrate seamlessly with your existing investments; you need ongoing support and training post-deployment; and you need someone that is truly easy to work with—for a change. Ultimately, when you succeed, we all succeed.

Our Solutions

- Identity & Access Governance
- Access Management
- Security Management
- Systems & Application Management
- Workload Management
- Service Management

Contacting Sales Support

For questions about products, pricing, and capabilities, contact your local partner. If you cannot contact your partner, contact our Sales Support team.

Worldwide:	www.netiq.com/about_netiq/officelocations.asp
United States and Canada:	1-888-323-6768
Email:	info@netiq.com
Web Site:	www.netiq.com

Contacting Technical Support

For specific product issues, contact our Technical Support team.

Worldwide:	www.netiq.com/support/contactinfo.asp
North and South America:	1-713-418-5555
Europe, Middle East, and Africa:	+353 (0) 91-782 677
Email:	support@netiq.com
Web Site:	www.netiq.com/support

Contacting Documentation Support

Our goal is to provide documentation that meets your needs. The documentation for this product is available on the NetIQ Web site in HTML and PDF formats on a page that does not require you to log in. If you have suggestions for documentation improvements, click **Add Comment** at the bottom of any page in the HTML version of the documentation posted at www.netiq.com/documentation. You can also email Documentation-Feedback@netiq.com. We value your input and look forward to hearing from you.

Contacting the Online User Community

NetIQ Communities, the NetIQ online community, is a collaborative network connecting you to your peers and NetIQ experts. By providing more immediate information, useful links to helpful resources, and access to NetIQ experts, NetIQ Communities helps ensure you are mastering the knowledge you need to realize the full potential of IT investments upon which you rely. For more information, visit community.netiq.com.

About this Book

This document describes the HTTP REST API for Advanced Authentication Server v5.2. The document is intended for developers and contains information on how to integrate strong authentication into the applications.

Intended Audience

This book provides information for individuals responsible for understanding administration concepts and implementing a secure, distributed administration model.

1 Advanced Authentication Overview

In this chapter:

1.1 About Advanced Authentication

Advanced Authentication^{TM} is a software solution that enhances the standard user authentication process by providing an opportunity to logon with various types of authenticators.

Why choose Advanced Authentication^{TM}?

Advanced Authentication^{TM}...

- ...makes the authentication process easy and secure (no complex passwords, “secret words”, etc.)
- ...prevents unauthorized use of your computer
- ...protects you from fraud, phishing and similar illegal actions online
- ...can be used to provide secure access to your office

1.2 Advanced Authentication Server Appliance Functionality

Benefits of using Advanced Authentication Server appliance are evident. Advanced Authentication Server appliance...

- ...is cross-platform
- ...contains an inbuilt RADIUS server
- ...supports integration with Advanced Authentication Access Manager
- ...does not require scheme extending
- ...provides administrators with a capability of editing the configured settings through web-based Advanced Authentication Administrative Portal

1.3 Architecture

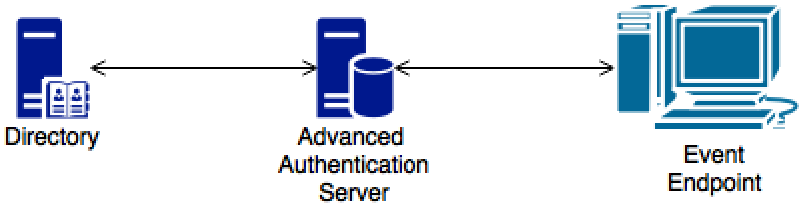
In this chapter:

- Enterprise Architecture
- Enterprise Architecture with Load Balancer

1.3.1 Basic Architecture

The basic architecture of the Advanced Authentication is simple and requires only one Advanced Authentication Server. You can use it for testing and proof of concepts.

Advanced Authentication Server is connected to a Directory that can be an Active Directory Domain Services, NetIQ eDirectory, Active Directory Lightweight Directory Service or other compliant LDAP directories. An Event Endpoint can be Windows, Linux or Mac OS X machine, NetIQ Access Manager, NetIQ CloudAccess, or RADIUS Client to authenticate through the RADIUS Server that is built-in the Advanced Authentication Server. For a complete list of supported events, see“Configuring Events” in the *Advanced Authentication Server Administration Guide*.



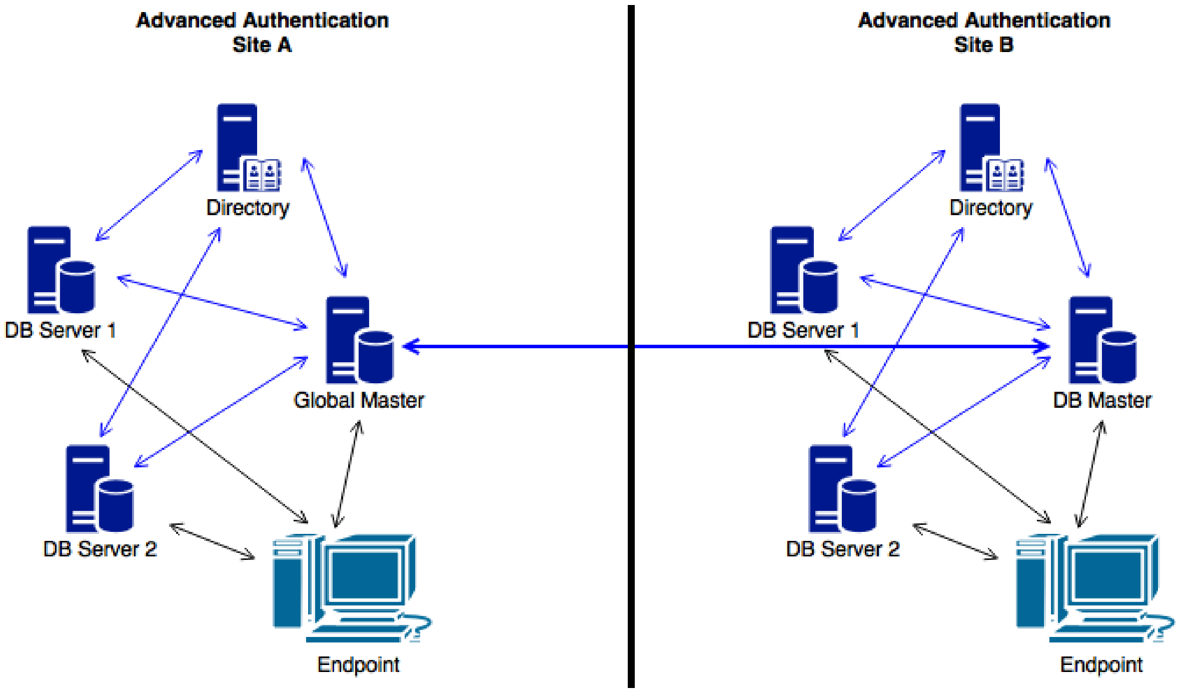
1.3.2 Enterprise Architecture

The Enterprise architecture of the Advanced Authentication contains sites that can be created for different geographical locations. For example, the following illustration displays two Advanced Authentication sites. Site A is the first site created for headquarters in New York. Site A’s first Advanced Authentication Server contains the **Global Master** and **Registrar** roles. This server contains a master database and it can be used to register new sites and servers.

Site B is created for the office in London and it contains the identical structure. The master server in another site has **DB Master** role. DB Masters interacts with the Global Master.

DB Server provides a DB Slave database that is used for backup and fail-over. You can create a maximum of two DB Slave Servers per site that can be DB Server 1 and DB Server 2. When the DB Master is unavailable, the DB Slave node responds to the database requests. When the DB Master becomes available again, the DB Slave node synchronizes with the DB Master and the DB Master becomes the primary point of contact for database requests again.

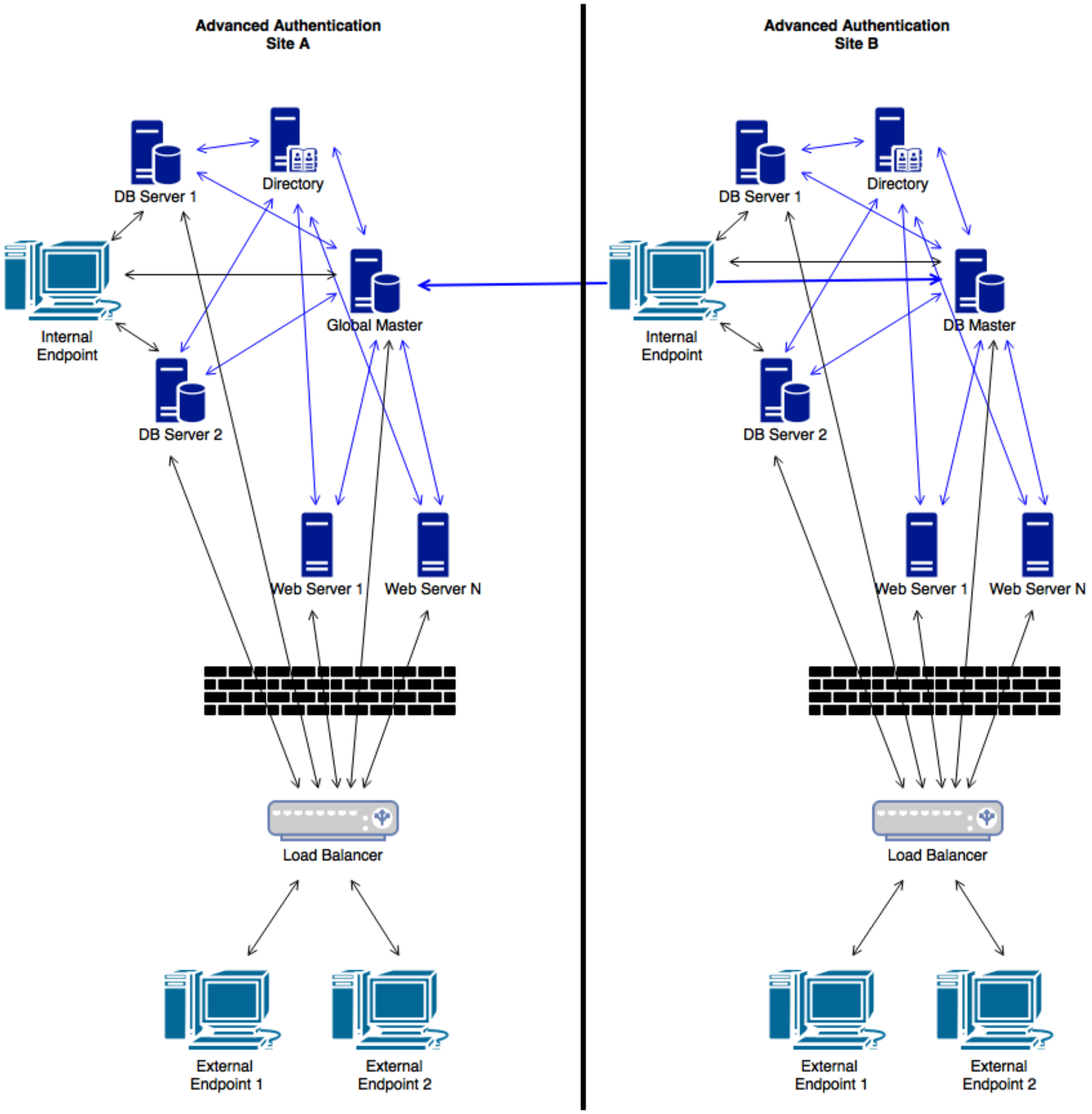
Endpoints can interact with every server that contain a database.



1.3.3 Enterprise Architecture with Load Balancer

The Enterprise architecture with Load balancer contains a more complicated architecture in comparison with the Enterprise Architecture. The architecture contains the following components:

- **Web Servers**: Web Server does not contain a database. It responds to the authentication requests and connects to the DB Master database. You need more Web Servers to serve more workload. There is no limitation for Web Servers.
- **Load Balancer**: It provides an ability to serve authentication requests from the **External Endpoints**. Load Balancer is a third-party component. It is located in DMZ and can be configured to interact with all the Advanced Authentication Servers.



1.3.4 How to Configure Load Balancer for Advanced Authentication Cluster

Load balancer can be installed and configured via third party software. Below is an example of how to install and configure nginx as load balancer on Ubuntu 14.

Target configuration:

	Hostname	IP address	Role	Operation System
Domain controller	win-dc	192.168.1.42	AD DS, DNS	Windows Server 2008 R2
NAAF 5.1 master	naafmaster	192.168.1.43	NAAF Master server	NAAF 5.1.2

	Hostname	IP address	Role	Operation System
NAAF 5.1 slave	naafslave	192.168.1.41	NAAF Slave server	NAAF 5.1.2
Load balancer	loadbalancer	192.168.1.40	Nginx load balancer	Ubuntu 14

Before starting the configuration, please make sure that the following requirements are fulfilled:

- Repository is configured in Advanced Authentication appliance.
- Both Advanced Authentication servers are installed and configured as Master and Slave.
- Appropriate entries are added to DNS.
- Ubuntu 14 is installed.

To configure Load Balancer for Advanced Authentication cluster, it is required to install nginx on Ubuntu 14 and configure it.

Installing nginx on Ubuntu 14

To install nginx on Ubuntu 14, follow the steps:

1. Open the following source list:
   - sudo nano /etc/apt/sources.list
2. Add necessary entries:
   - `deb http://nginx.org/packages/ubuntu/ trusty nginx`
   - `deb-src http://nginx.org/packages/ubuntu/ trusty nginx`
3. Update repository and install nginx:
   - apt-get update
   - `apt-get install nginx`
4. Start nginx and make sure that web server is working:
   - `sudo service nginx restart`
5. Open your browser and go to web server http://192.168.1.40 or http://loadbalancer.

Configuring nginx

The following load balancing mechanisms/methods are supported in nginx:

- **round-robin** - requests to the application servers that are distributed in a round-robin fashion
- **least-connected** - next request assigned to the server with the least number of active connections
- **ip-hash** - a hash-function that is used to determine what server should be selected for the next request (based on the client’s IP address)

This article describes only round-robin configuration. To configure nginx, follow the steps:

1. Backup original configuration file: `sudo cp /etc/nginx/nginx.conf /etc/nginx/nginx.conf_original`.
2. Open the *nginx.conf* file and replace with following:

```
user nginx;
error_log /var/log/nginx/error.log warn; # error log location
pid /var/run/nginx.pid; # process id file
# limit number of open sockets. Debian default max is 1024, ensure nginx not
open all the sockets.
worker_processes 1;
events {
worker_connections 900; # 512 is default
}
# worker_processes auto; # ssl needs CPU
http {
include /etc/nginx/mime.types;
default_type application/octet-stream;
log_format main '$remote_addr - $remote_user [$time_local] "$request" '
'$status $body_bytes_sent "$http_referer" '
'"$http_user_agent" "$http_x_forwarded_for"';
access_log /var/log/nginx/access.log main; # access log location
sendfile on;
# keepalive default is 75
# keepalive_timeout 10;
gzip on;
gzip_static on;
gzip_comp_level 5;
gzip_disable msie6;
gzip_min_length 1000;
gzip_proxied expired no-cache no-store private auth;
gzip_vary on;
gzip_types text/plain text/css application/json application/javascript
text/xml application/xml application/rss+xml application/atom+xml;
ssl_certificate /etc/nginx/cert.pem;
ssl_certificate_key /etc/nginx/cert.pem;
ssl_session_cache shared:SSL:2m; # 1m stores 4000 sessions, default expire 5
min
ssl_protocols TLSv1 TLSv1.1 TLSv1.2; # disable TLSv3 - POODLE vulnerability
resolver 192.168.1.42 valid=300s ipv6=off; # ip address of DNS
resolver_timeout 10s;
upstream web {
#server naafmaster.company.local:443 resolve;
#server naafslave.company.local:443 resolve;
server 192.168.1.43:443;
server 192.168.1.41:443;
}
server {
#listen 80;
listen 443 ssl;
location / {
proxy_pass https://web;
proxy_set_header HOST $host;
proxy_set_header X-Forwarded-Proto $scheme;
proxy_set_header X-Real-IP $remote_addr;
proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
}
}
}
```

3. Copy certificate from any Advanced Authentication server in cluster from the directory */etc/nginx/cert.pem* to the same directory on load balancer.

4. Go to https://loadbalancer/admin page and make sure that connection was redirected to Advanced Authentication cluster.

IMPORTANT: Nginx can be installed and configured on any Linux supported by nginx.

Additional information on nginx configuration can be found at http://nginx.org/en/docs/.

1.4 Terms

In this chapter:

- Authentication Method
- Authentication Chain
- Authentication Event

1.4.1 Authentication Method

Authentication Method verifies the identity of someone who wants to access data, resources, or applications. Validating that identity establishes a trust relationship for further interactions.

1.4.2 Authentication Chain

Authentication Chain is a combination of authentication methods. User needs to pass all methods in order to be successfully authenticated. E.g., if you create a chain which has LDAP Password and SMS in it, the user will first need to enter his/her LDAP Password. If the password is correct, the system will send SMS with an One-Time-Password to the mobile of the user. The user needs to enter the correct OTP in order to be authenticated.

It is possible to create any chain. So for high secure environments it is possible to assign multiple methods to one chain to achieve better security.

Authentication can consist of 3 different factors. These are:

- Something you know: password, PIN, security questions
- Something you have: smartcard, token, telephone
- Something you are: biometrics like fingerprint or iris

Multi-Factor or Strong Authentication is when 2 out of the 3 factors are used. A password with a token, or a smartcard with a fingerprint are considered to be multi-factor authentication. A password and a PIN is not consideed to be multi-factor as they are in the same area.

Authentication chains are linked to user groups in your repositories. So only a certain group can be allowed to use the specific authentication chain.

1.4.3 Authentication Event

Authentication Event is triggered by an external device or application which needs to perform authentication. It can be triggered by a RADIUS Client (Citrix Netscaler, Cisco VPN, Juniper VPN, etc) or API request. Each event can be configured with one or more authentication chains which will provide user with a capability to authenticate.

Within the Advanced Authentication framework, an authentication event is configured in the Events section. It is possible to enable or disable an event, and to add method-chains to the event. With specific events it is possible to assign clients to the event.

2 About the API

The current API version is 1.0. Since API is based on REST principles, it is easy to write and test applications. The browser can be used to access URLs. Any HTTP client in any programming language can be also used to interact with the API.

API accepts and responds with JSON objects.

Base URL

The Base URL of the REST API is

```
https://authserver.example.com/api/v1/
```

Replace authserver.example.com by hostname of your appliance.

Use the HTTPS protocol for more security with interaction with server's REST API. The authentication server can return security information, e.g. Domain passwords. Note that some network libraries can block self-signed SSL certificate.

Main Processes

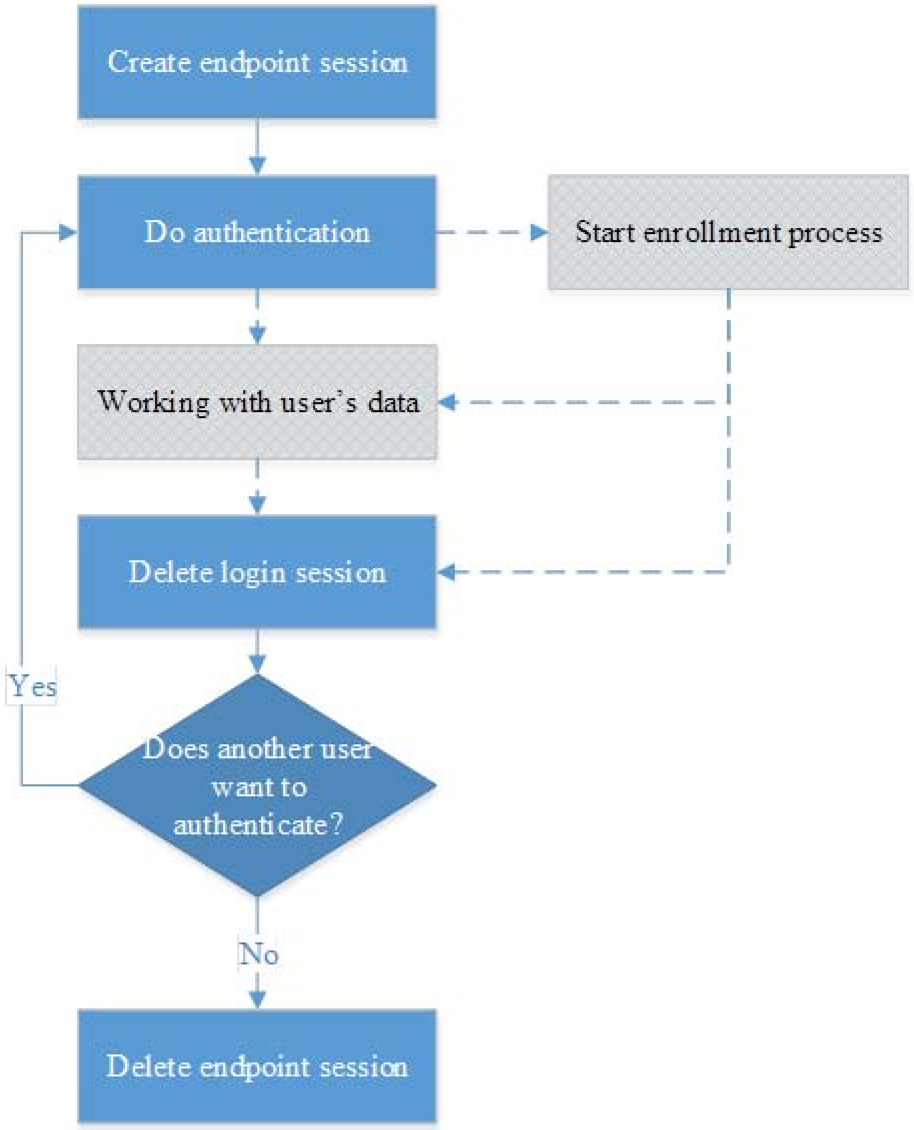
The Authentication Server has two main processes:

- Logon Process
- Enrollment Process.

2.1 Logon Process

The logon process provides authentication and authorization, the enrollment process stores authentication information for new authentication methods. For enrollment, you should be authenticated before. The following picture describes the logon process.

Figure 2-1 Picture 1 - Logon process



The logon process includes several steps:

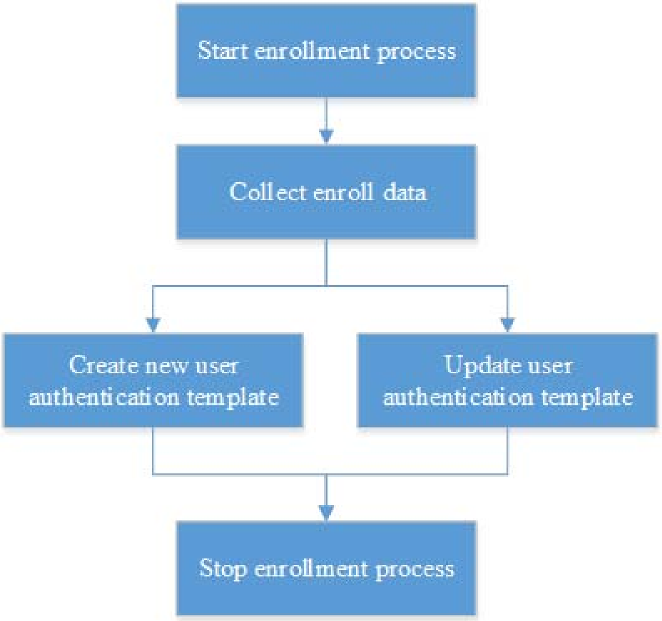
1. **Create endpoint session** – Endpoints are devices which connect to the server. A valid endpoint session is needed in order to communicate with the server. This is a sub process. For more information, check the Working With Endpoint Sessions chapter.
2. **Do authentication** – The user can perform the actual authentication after creating the endpoint session. Authentication is a sub process. For more information, check the Provide Authentication chapter.
3. **Working with user's data** – Information about a user like password or any other credential can be stored in the user data container and can be accessed after a successful authentication. For more information, check the Working With User's Data chapter. It is also possible to start an enrollment after successfully authenticating the user.
4. **Delete login session** – After authentication the login session should be deleted. For more information, check the Delete Login Sessions chapter.

5. **Does another user want to authenticate?** - after the login session is deleted, the system will wait for another user. If a new user wants to login, the process will be started from the second step because the endpoint is already created.
6. **Delete endpoint session** - the logon process will delete endpoint session at the end. For more information, check the Delete Endpoint Session chapter.

2.2 Enrollment Process

The following chart describes the enrollment process.

Figure 2-2 *Picture 2. Enrollment process.*


Start enrollment process
Collect enroll data
Create new user authentication template
Update user authentication template
Stop enrollment process

Enrollment process contains several steps.

1. **Start enrollment process** - this step is the primary one in the enrollment process. For more information, check the Start Enroll Process chapter.
2. **Collect enroll data** - when the enrollment process is started, the user’s credentials will be collected. For more information, check the Providing Data Into Enroll Process chapter.
3. If this is a new enrollment the enrollment data will be saved. If the user already has an enrollment for this specific method the enrollment data will be updated:
   - **Create new user authentication template** - For more information, check the Create User's Templates From Enroll Session chapter.
   - **Update user authentication template** - For more information, check the Updating User's Template chapter.
4. **Stop enrollment process** - after collecting enrollment data, enrollment process should be stopped. For more information, check the Delete Enroll Process chapter.

The Working With Enrollment chapter contains information of the enrollment process.

The Working With User's Templates chapter describes users’ templates .

3 API Documentation

3.1 Localization

The appliance is supported in different languages and help users with their troubles and problems.

If you want to use the localized messages, specify locale in you requests. For this, use a special parameter.

Parameter Name	Parameter Description
`_LOCALE_`	The locale for messages in responses from server, you could use it in different formats: languages with territory, for example, en_US, nl_NL, or ru-RU, fr-FR, or just languages, for example, ru, nl, fr.

For `GET` request add this parameter as a URL parameter, for `POST`, `PUT` and `PATCH` add this parameter as a `JSON` parameter in request body. If for some reasons, you couldn't change a request URI or body, you could specify a cookie with the parameter name and value and server will automatically read it and change locale, or you could add the header "Accept-Language" with locale value in your request.

There are the list of supported locales

Locale	Locale Value	Language

If you could not find your locale in table above, you could use a messages identifier for your custom localization. In responses you could find this parameter with identifier of message.

Parameter Name	Parameter Description
msgid	The identifier of the message for message customization.

This parameter could be used for additional messaging too, but do not use this parameter for checking status of an authentication process or other statuses, for this kind of checking API has special parameters.

On requests and response in examples below all messages in default English locale, message identifier was removed.

3.2 Working With Endpoints

3.2.1 About Endpoints

The appliance provide user's authentication in different places, ex. Microsoft Windows, Apple MacOS X or other custom applications and systems. The final destination of user's authentication is endpoint. The endpoint could be a physical workstation or an application. The endpoints are combined in the events, the event is a logical separation for the endpoints. The Windows logon and Mac OS logon are different events, each event could had some set of the endpoints, and endpoints are Mac or Windows workstations. System has the following standard events.

Table 3-1 *Supported Events List*

Event name	Event identifier	Description
Windows logon	WINDOWS	Event for log on in Microsoft Windows, used for authentication on Windows based operation system.
Template management	TEMPLATES	Template (authenticator) management event, this event is used for the enrollment process, collecting authentication data and updating user's template.
Endpoint management	ENDPOINTS	Endpoint management event, for management endpoints use this event.
NAM	NAM	Event for Advanced Authentication Access Manager.
NCA	NCA	Event for Advanced Authentication Cloud Access.
Admin user interface	ADMIN	The event for logon into Advanced Authentication Administrative Portal.
Radius Server	RADIUS	The event for logon by RADIUS server.
Helpdesk	HELPDESK	The event for logon into Advanced Authentication Helpdesk Portal.
MacOS logon	MACOS	The event for log on in Apple MacOS.

3.2.2 Create Endpoint

For creating endpoint, you could use the following resource with URI:

```
/endpoints
```

Resource provided by HTTP POST and has these parameters as JSON-object.

Table 3-2 *URI Parameters for creating endpoint.*

URI parameter name	Description
name	A name for the new endpoint.
typ	A type identifier of the new endpoint.
desc	A description of the new endpoint

The appliance supports this list of the endpoints types:

Table 3-3 *Supported endpoint types*

Type value	Type identifier	Description
TYP_UNKNOWN	1	The unknown type for the endpoint, use this value if you couldn't use other identifier.
TYP_WINDOWS_CLIENT	2	The identifier for the Windows Clients.
TYP_NAM	3	The identifier for the Advanced Authentication Access Manager.
TYP_MACOS_CLIENT	4	The identifier for the Mac OS Clients.
TYP_LINUX_CLIENT	5	The identifier for the Linux Clients.
TYP_NCA	6	The identifier for the Advanced Authentication CloudAccess.
TYP_RADIUS	7	The identifier for the RADIUS client.

The resource returns a JSON-object, which contain the identifier and the secret of the created endpoint.

Table 3-4 *Object with created endpoint.*

Parameter name	Description
id	The identifier of the created endpoint.
secret	The secret of the created endpoint.

Example

The following example demonstrates how to create a new endpoint.

```
HTTP POST
https://authserver.example.com/api/v1/endpoints
equest
{
"name":"nam.company.local",
"typ":3,
"desc":"NAM endpoint"
}
Response
{
"id":"885f71c85b8711e5b507000c2951aca4",
"secret":"Mxy4NJLHEGYicviRqrPqdaxRo0RAQtYn"
}
```

3.2.3 Delete Endpoint

For deleting endpoint, you could use the following resource with URI:

```
/endpoints/{endpoint_id}?secret={endpoint_secret}
```

Resource is provided by HTTP DELETE. It has the following parameters.

URI Parameter Name	Parameter Description
`endpoint_id`	Endpoint identifier. Identifier of endpoint for deleting.
`secret`	The secret of the endpoint for deleting

Resource does not return any data. If the deletion is successful, the HTTP 200 status will be returned.

Example

Endpoint with identifier “42424242424242424242424242424242” and secret “1234567890” will be deleted in the following example.

```
HTTP DELETE
```

```
https://authserver.example.com/api/v1/endpoints/
42424242424242424242424242424242?secret=1234567890
```

```
Response
```

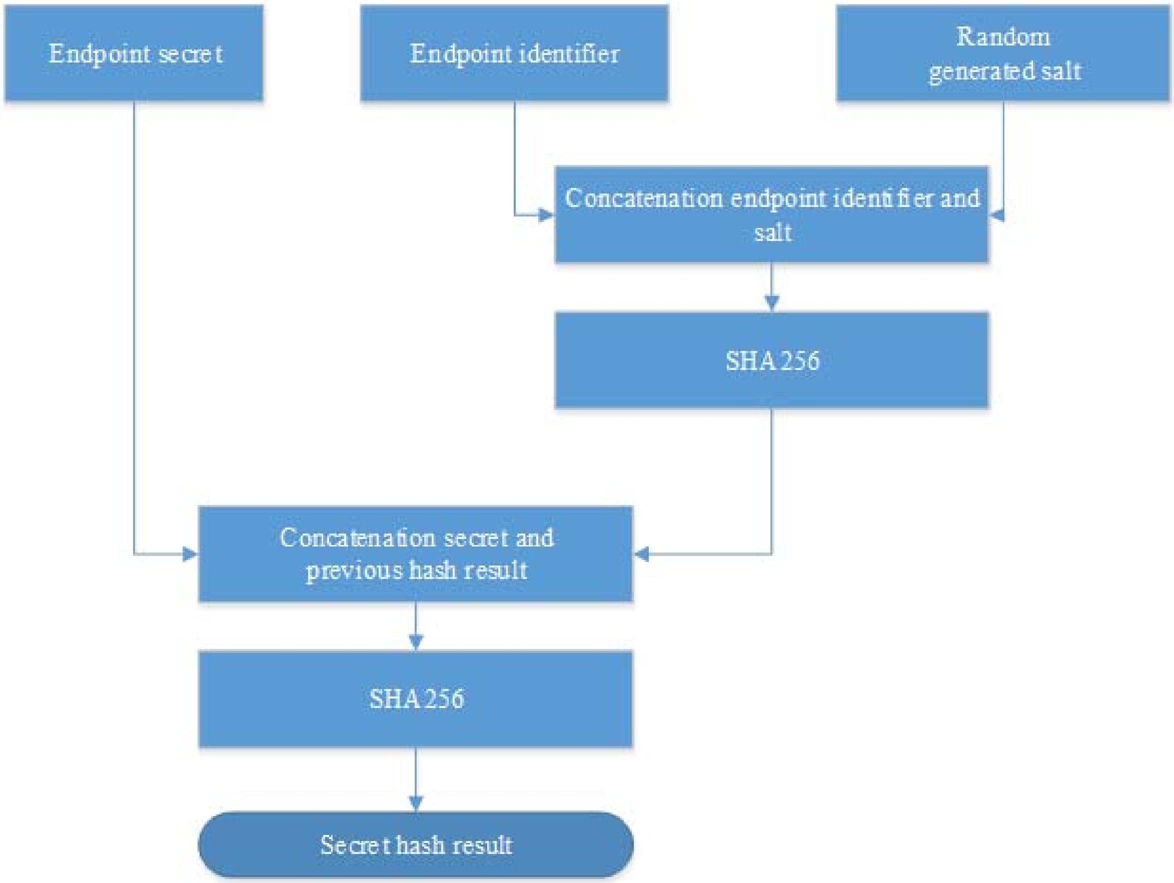
```
HTTP 200
```

3.3 Working With Endpoint Sessions

3.3.1 About Endpoint Sessions

Endpoint is any logical or physical unit which interacts with the authentication server. E.g., client computer, tablet device, smartphone, any software or system is an endpoint. Endpoint should create endpoint session on the server to start working. The appliance use endpoints’ sessions for checking what PC wants to get access, the appliance supports endpoint’s white lists for events, also the appliance can check the owner for an endpoint. The whitelist for events can restrict access to the endpoints, for example, if a workstation (as endpoint) is not added to the whitelist for the Windows or Mac OS event, the workstation cannot use advanced authentication. Each endpoint has an identifier and secret. Secret is a security value that is used for generating endpoint security hash. Security hash is used to start endpoint session. The algorithm for generating the secret hash is represented on the following picture.

Figure 3-1 Algorithm for generating the secret hash


Endpoint secret
Endpoint identifier
Random generated salt
Concatenation endpoint identifier and salt
SHA256
Concatenation secret and previous hash result
SHA256
Secret hash result

After SHA-256 function, you should transform result to a HEX-value string. All parameters are string. All concatenation functions are string concatenation functions. The salt should be generated on the client side. It is advised to use a strong algorithm for generating the salt.

The following example shows how to generate secret hash.

```
Endpoint id = 42424242424242424242424242424242
Salt = e26eaecba7cbe186c08469f6ddbf6f6c0321651b53f80d8eb2c3b0d4e1c19c4c
Endpoint secret = 12345678
Middle result = e53e75ec3d48171abff1e4d2f0a5066e64fdbe11a4d3fd7c45810d7d93c26956 =
SHA256(endpoint id + salt)
Result 3b5dac383282df6936f9350a01ad079096f777f5c44eda8e0c2e66bfc443ee26 =
SHA256(endpoint secret + middle result)
```

The endpoint identifier and endpoint secret need to be very secure. This information should be kept in a secure place and parameters should be changed every time the information is compromised.

Endpoints can work with the server and provide user authentication after the endpoint session creation. The endpoint session has a lifetime, after this, the endpoint session will be deleted and the session will need to be renewed. All users can work with one endpoint session on one endpoint.

3.3.2 Create Endpoint Session

To create the endpoint session, use the following resource with URI:

```
/endpoints/{endpoint_id}/sessions
```

Resource is provided by HTTP POST. It has the following parameter.

Table 3-5 *URI parameters for creating endpoint session.*

URI Parameter	Description
endpoint_id	Endpoint identifier. Identifier of endpoint that creates endpoint session.

Resource accept JSON-object with these parameters.

Table 3-6 *Parameters for creating endpoint session.*

Parameter name	Description
salt	Client generated salt. This salt is used in secret hash generated algorithm.
endpoint_secret_hash	Generated secret hash by algorithm.
session_data	Any session data. This is a JSON-object with any parameters and structure.

The resource returns JSON-object with the following endpoint session identifier.

Table 3-7 *JSON-object with created endpoint session.*

Parameter name	Parameter description
endpoint_session_id	Endpoint session identifier. This is an identifier of the created endpoint session. It should be used for other methods.

Example

Endpoint with identifier “42424242424242424242424242424242” creates endpoint session. Endpoint has already generated salt and endpoint secret hash.

```
HTTP POST
https://authserver.example.com/api/v1/endpoints/42424242424242424242424242424242/
sessions
Request
{
"salt":"2615c070937935246c6a91df70a8eb672b21d842a225621c9797a83bedf00a7b",
"endpoint_secret_hash":"38d55fb7a899dcef6cbec053df8f7673cb05068b9ee9d6a23ee759232b
25cf4e",
"session_data":{}
}
Response
{
"endpoint_session_id":"IRx7UXwenytMn5B7fgRSH4k1s6PAAs0I"
}
```

3.3.3 Read Information About Endpoint Sessions

To read information about endpoint session, use the following resource with URI:

```
/endpoints/{endpoint_id}/sessions/
{session_id}?salt={salt}&endpoint_secret_hash={endpoint_secret_hash}
```

Resource is provided by HTTP GET. It has the following parameters.

Table 3-8 *URI parameters for reading information about endpoint session.*

URI parameter name	Description
endpoint_id	Endpoint identifier. Identifier of endpoint which session will be read.
session_id	Session identifier. Identifier of endpoint session for reading.
salt Salt	It is used for generated endpoint secret hash
endpoint_secret_hash	Endpoint secret hash.

Resource returns JSON-object which contains the following information about endpoint session.

Table 3-9 *JSON-object with information about endpoint session.*

Parameter name	Description
sid	Endpoint session identifier, identifier of endpoint session
endpoint_id	Endpoint identifier
session_data	Session data, this data can be added on creating session, this is any JSON-object with any structure.

Example

We read the endpoint session information for endpoint with identifier “42424242424242424242424242424242”. Session identifier is ”T2Fv3FOvWojyBOZBlsik1d9RBDjDr9Rq”. Endpoint has already prepared secret hash and salt.

```
HTTP GET
https://authserver.example.com/api/v1/endpoints/42424242424242424242424242424242/
sessions/
T2Fv3FOvWojyBOZBlsik1d9RBDjDr9Rq?salt=279bc0da2a9da8afcfda50c2bc3f418218354020d73c
00a6a088dea704ece3f9&endpoint_secret_hash=bfe7b1e1cad57525a9695f39eb5cff4d4dd7a42e
039122c732adb151d097835c
Response
{
"sid":"T2Fv3FOvWojyBOZBlsik1d9RBDjDr9Rq",
"endpoint_id":"42424242424242424242424242424242",
"session_data":null
}
```

3.3.4 Delete Endpoint Session

To delete an endpoint session, use the following resource with URI:

```
/endpoints/{endpoint_id}/sessions/
{session_id}?salt={salt}&endpoint_secret_hash={endpoint_secret_hash}
```

Resource is provided by HTTP DELETE. It has the following parameters.

***Table 3-10** URI parameters for deleting endpoint session.*

URI parameter name	Description
endpoint_id	Endpoint identifier. Identifier of endpoint which session will be deleted.
session_id	Session identifier. Identifier of endpoint session to be deteled.
salt Salt	It is used for generated endpoint secret hash.
endpoint_secret_hash	Endpoint secret hash.

Resource does not return any data. If the deletion is successful, the HTTP 200 status will be returned.

Example

Endpoint with identifier “42424242424242424242424242424242” deletes session with identifier “9Ml1ByqdIsCr3re5C0aivXR6ap20fNW8”.

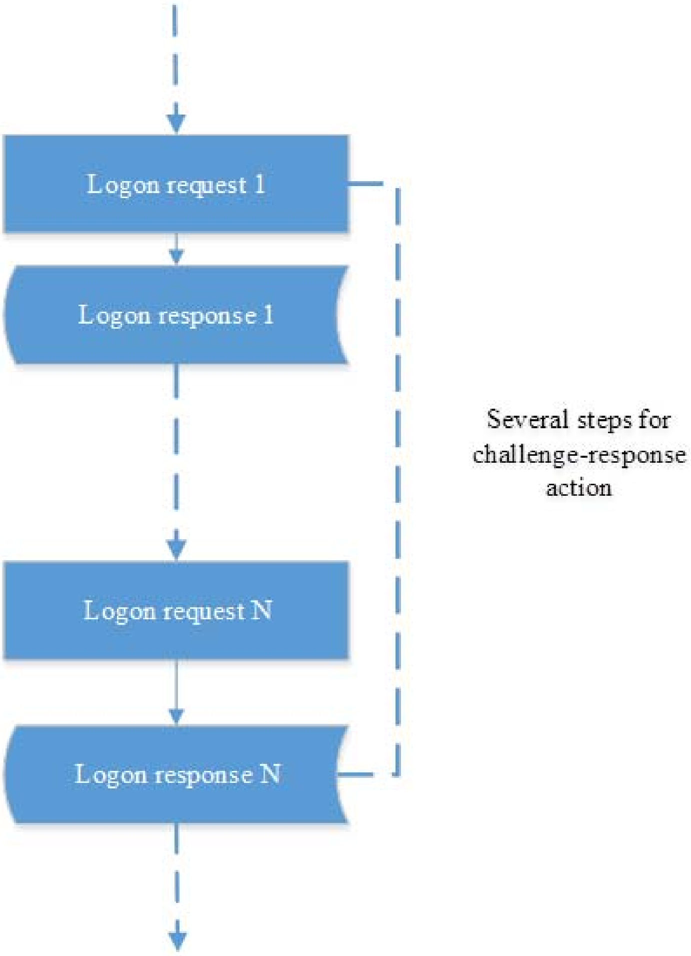
```
HTTP
DELETE
https://authserver.example.com/api/v1/endpoints/42424242424242424242424242424242/
sessions/
9Ml1ByqdIsCr3re5C0aivXR6ap20fNW8?salt=4b470b4ee96630ced049a78d6488ee127b9d419f4635
30c63f3e1ee27226f721&endpoint_secret_hash=c2bf7b8288aa55de22baa9d077150d83b3460d59
51a57d6e7ccf2a01f61bc9ec
Response
HTTP 200
```

3.4 Provide Authentication

3.4.1 About Login Process

The server provides strong user authentication by using the chain-login concept. Each chain is a challenge-response login. The following chart describes chain logic. To get a successful authentication, the entire chain should be completed. A chain can consist of one or several authentication method(s).

Figure 3-2 *Chain logic.*



The user should start the logon process and pass all authentication methods. If all of them are successful, the user will login. If not, he/she will fail and should start the logon process once again.

API has the following standard responses for authentication status description.

Table 3-11 *Response statuses.*

Status name	Status value	Description
OK	OK	This status is used to describe success result of logon process. Server has created the login session.
More data	MORE_DATA	This status is used to indicate that authentication method requests for additional data for authentication. Each authentication method requests different data.
Next	NEXT	This status indicate that current method finished, you should check completed_methods parameter from server response to be sure that current method was really finished.
Failed	FAILED	This status is used for unsuccessful result of logon process description. It indicates that the logon process should be started once again.

For additional information the server also returns the authentication reasons, the server has standard reasons for all authentication methods and specific reasons for each authentication method. Please check authentication method description for more information about supported authentication reasons.

Table 3-12 *Common authentication reasons*

Reason value	Description
METHOD_COMPLETED	The authentication method was completed
METHOD_NOT_NEEDED	The authentication method not needed. The event does not work with this method
METHOD_RETRY	The authentication method was retried
ENDPOINT_DISABLED	The endpoint is disabled
ENDPOINT_NO_ACCESS	The endpoint has no access
CHAIN_DISABLED	The authentication chain is disabled
CHAIN_COMPLETED	The authentication chain was completed
PROCESS_STARTED	The authentication process was started
PROCESS_NOT_FOUND_OR_EXPIRED	The authentication process was not found or process is expired, you should start a new logon process
USER_LOCKED	User is locked

Once the chain is completed, user's data associated with event can be accessed. Event is the logical final destination for login process. Each event is a security point of the system and the login process provides authentication and authorization to this event. Organizations can create their own login chains with different authentication method. Currently the following authentication methods are supported.

Table 3-13 *Authentication methods.*

Authentication method name	Authentication method identifier	Description
LDAP password	LDAP_PASSWORD:1	Authentication by LDAP password, system uses different LDAP users repository and provide authentication by LDAP password, check the LDAP Password Authentication Method chapter
One-time password based on hash algorithm	HOTP:1	Authentication by OTP with hash algorithm, check the HOTP Authentication Method chapter
One-time password based on time algorithm	TOTP:1	Authentication by OTP with time based algorithm, check the TOTP Authentication Method chapter
One-time password sending by e-mail	EMAIL_OTP:1	Authentication by OTP sending by e-mail, check the Email Authentication Method chapter
One time password sending by SMS	SMS_OTP:1	Authentication by OTP sending by SMS, check the SMS Authentication Method chapter
RADIUS password	RADIUS:1	Authentication by RADIUS server, check the RADIUS Authentication Method chapter

Authentication method name	Authentication method identifier	Description
Security question	SECQUEST:1	Authentication by security question, check the Security Questions Authentication Method chapter
Smartphone authentication	SMARTPHONE:1	Authentication by smartphone application, check the Smartphone Authentication Method chapter
Virtual password	PASSWORD:1	Authentication by password assign to user, check the Password Authentication Method chapter
Voice call	VOICE:1	Authentication by voice call, check the Voice Call Authentication Method chapter
Cards	CARD:1	Authentication by cards, check the Card Authentication Method chapter
FIDO U2F	U2F:1	Authentication by FIDO U2F tokens, check the FIDO U2F Authentication Method chapter
Emergency password	EMERG_PASSWORD:1	Authentication by emergency password, check the Emergency Password Authentication Method chapter
NotarisID	NOTARIS_ID:1	Authentication by NorarisID, see the “NotarisID authentication method” capter.
PKI	PKI:1	The authentication by PKI (Public key infrastructure), see the “PKI authentication method” chapter.

Combining different authentication methods into authentication chains provides strong protection for different applications.

Before you start the logon process, you should create an endpoint session. The endpoint is the final destination for the login process. E.g., a client PC is an endpoint. One endpoint session can provide many logon processes. Endpoint session should be created once and used for all logon processes. For more information on endpoint session, check the Create Endpoint Session chapter.

3.4.2 Provide Simple Authentication Using One Method In A Chain

API has two resources for providing login:

- for starting the login session
- for providing data to the authentication provider

This chapter describes how to provide a simple logon with one authentication method in the chain.

To start the logon process, the request should be sent to the following resource with URI:

```
/logon
```

Resource is provided by HTTP POST. It accepts JSON-object which describes the new logon process.

Table 3-14 Parameters for staring new login process.

Parameter name	Description
endpoint_session_id	Endpoint session identifier. Identifier of endpoint which creates logon session.
method_id	Authentication method identifier. Identifier of method from authentication methods list that is supported by the server. This parameter defines which method will be used for authentication. To view the list of the supported methods, check the Authentication methods..
user_name	User name for login. User name should be provided in format repo_name/user_name, where repo_name is user's repository name from server, user_name is user name.
event	Event identifier. Event from list of events supported by server.
is_1N	The boolean flag identify is authentication will be 1:N or not, if true, that authentication will be 1:N, if false – not 1:N
unit_id	The unit identifier, use this value with is_1N flag, if you use this parameter you should not provide user name for the authentication

Table 3-15 Created login session.

Parameter name	Description
chains	JSON-object, which describe available chains for login.
completed_methods	JSON-array, which contain list of completed authentication methods supported by the server.
current_method	Current authentication method, which is used for authentication.
msg	Message about process, contains more information about authentication process or about errors.
logon_process_id	Logon process identifier, identifier of current logon process, this identifier is constant in the authentication process and is used in next steps.
plugins	JSON-array, that contain list of available plugins.
status	Current status of logon process, check the Response statuses. for more information.
event_name	The event name for started logon process
reason	The authentication reasons, please check the Common authentication reasons
event_data_id	The event data identifier for started logon process

Chains JSON-object has these parameters.

Table 3-16 *Chain JSON-object.*

Parameter name	Description
methods	JSON-array of authentication methods represented into chain. All of these methods should be successful for login.
is_trusted	Boolean flag which determinate is it trusted chain or not.
name	Name of authentication chain.
image_name	The image name for this chain
apply_for_ep_owner	This is a boolean flag, if true – this is an authentication chain for the endpoints owner and this chain will have priority over other chains
position	The position of this chain in an authentication chains list
is_enabled	The boolean flag, if it true chain is enabled, if false – disabled
short_name	The short name of the chain

After starting the logon process, authentication data for the current authentication method in the chain should be sent. To send authentication data, use the following resource with URI:

```
/logon/{logon_process_id}/do_logon
```

Resource is provided by HTTP POST. It has the following parameters.

Table 3-17 *URI parameters for sending authentication data.*

URI parameter name	Description
logon_process_id	Logon process identifier. Identifier of started logon process.

Resource accepts data as JSON-object with these parameters.

Table 3-18 *Parameters for providing authentication data.*

Parameter name	Description
response	JSON-object with authentication data for method. This object is different for all authentication methods.
endpoint_session_id	Endpoint session identifier. Identifier of endpoint, which works with logon session.

Resource return JSON-object with data of authentication result.

Table 3-19 *JSON-object with authentication result.*

Parameter name	Description
chains	JSON-object, which describes available chains for login, check the Table 3-16. Chain JSON-object for more information.
current_method	Current authentication method, which is used for authentication.

Parameter name	Description
logon_process_id	Logon process identifier, identifier of current logon process, this identifier is constant in the authentication process and is used in next steps.
completed_methods	JSON-array, which contains list of completed authentication methods supported by server.
status	Current status of logon process. Check the About Login Process for more information.
repo_id	Repository identifier, server identifier of users' repository.
user_name	User name.
user_id	User server identifier.
msg	Message about process, contains more information about authentication process or about errors.
login_session_id	Login session identifier, identifier of success authentication session.
plugins	JSON-array, that contain list of available plugins.
reason	The authentication reasons, please check the Common authentication reasons
event_name	The event name
repo_obj_id	The identifier of the repository object
user_sid	The user's SID
user_name	The user name, e.g. COMPANY\JSmith
user_mobile_phone	The user's mobile phone if it presents.
user_dn	The user FQDN-name
event_data_id	The event data identifier
user_sid_hex	The user's SID as hex-string
user_email	The user's email if it presents
user_cn	The user canonical domain name
data_id	The data identifier
user_name_netbios	The user's NETBIOS name, if user came from Active Directory
user_upn	The user principal name, if user came from Active Directory

Each authentication method has a different number of challenge-response steps, check method description for more information.

Example

On the following example, user “JSmith” from repository “COMPANY” tries to login to the NAM endpoint with session identifier “46kGFB3MUUebkcqosO9t4pVAVURsCMyz”.

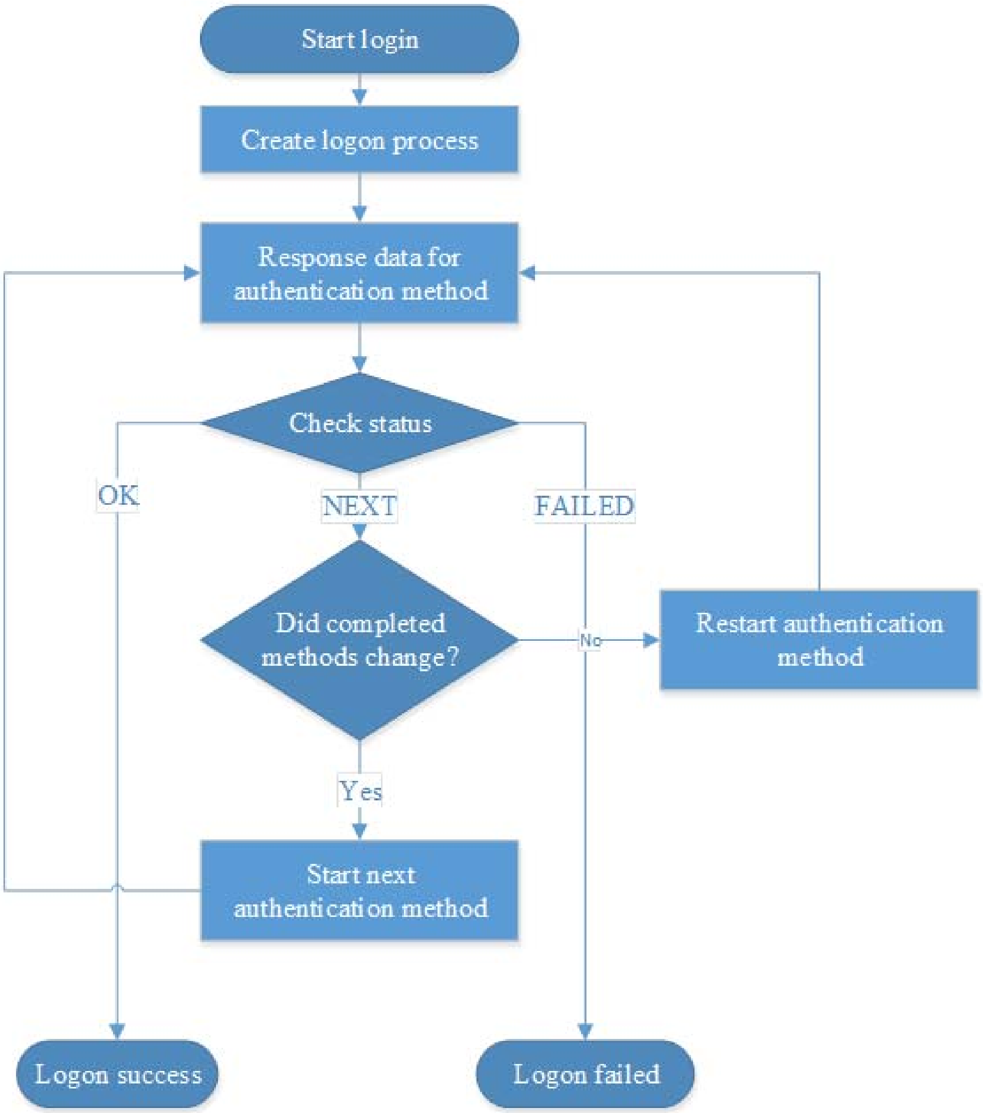
```
HTTP POST
https://authserver.example.com/api/v1/logon
Request
{
"method_id":"LDAP_PASSWORD:1",
"user_name":"COMPANY\\JSmith",
"event":"NAM",
"endpoint_session_id":"46kGFB3MUUebkcqosO9t4pVAVURsCMyz"
}
Response
{
"plugins": [],
"event_name":"NAM",
"msg":"Process started",
"status":"MORE_DATA",
"reason":"PROCESS_STARTED",
"current_method":"LDAP_PASSWORD:1",
"completed_methods":[],
"chains":[
{
"is_enabled":true,
"apply_for_ep_owner":false,
"is_trusted":null,
"image_name":"default",
"name":"LDAP password",
"position":0,
"methods":["LDAP_PASSWORD:1"],
"short_name": ""
}],
"logon_process_id":"zGjcx3Kbh3scIbTXXCefo1lR86BZ5V0k",
"event_data_id":"OSLogon"
}
HTTP POST
https://authserver.example.com/api/v1/logon/zGjcx3Kbh3scIbTXXCefo1lR86BZ5V0k/
do_logon
Request
{
"response":
{
"answer":"123"
},
"endpoint_session_id":"46kGFB3MUUebkcqosO9t4pVAVURsCMyz"
}
Response
{
"msg":"Welcome!",
"user_mobile_phone":"+16086783619",
"event_name":"NAM",
"repo_id":"d3fba00652d211e5a19a000c2951aca4",
"user_email":"jsmith@company.com",
"user_name": "COMPANY\\JSmith",
"user_dn":"CN=JSmith,CN=Users,DC=company,DC=com",
"completed_methods":["LDAP_PASSWORD:1"],
"chains":[
{
"is_enabled":true,
"apply_for_ep_owner":false,
"is_trusted":null,
"image_name":"default",
"name":"LDAP password",
"position":0,
"methods":["LDAP_PASSWORD:1"],
"short_name": ""
}],
"user_sid":"S-1-5-21-4279545561-3293806183-1755797738-500",
"user_name_netbios":"COMPANY\\JSmith",
"user_cn":"JSmith",
"user_id":"ab2b845652d311e5a19a000c2951aca4",
"login_session_id":"BF7xjPVyO9wSqFj9UY0II1qzsemfcVqD",
"plugins":["LdapRules"],
"data_id":"OSLogon",
"status":"OK",
"reason":"CHAIN_COMPLETED",
"current_method":"LDAP_PASSWORD:1",
"user_sid_hex":"01050000000000515000000d9ae14ff677e53c4ea58a768f40100",
"logon_process_id":"zGjcx3Kbh3scIbTXXCefo1lR86BZ5V0k",
"event_data_id":"OSLogon",
"repo_obj_id":"2d3c89ccb3ea7b4dacbdfda13e26f450"
}
```

3.4.3 Provide Chained Authentication

Chained authentication is like a simple authentication, it uses from the same logic as the simple authentication process and has the same JSON-object and other parameters. The differences are in step count, chained authentication combines many authentication methods and all of them should be passed successfully.

The logic of the chained authentication is displayed on the following chart. In this login, you should start next authentication method until you get successful or unsuccessful result. When the logon process is started, it will be required to choose authentication method and provide data for this authentication method at first iteration of logon process.

Figure 3-3 Picture 5. Chained authentication.


Start login
Create logon process
Response data for authentication method
Check status
OK
NEXT
FAILED
Did completed methods change?
No
Restart authentication method
Yes
Start next authentication method
Logon success
Logon failed

API for chained login has an additional resource for starting next authentication method, this resource has the following URI:

```
/logon/{logon_process_id}/next
```

Resource is provided by HTTP POST. It has the following URI parameter.

Table 3-20 URI parameters for starting next authentication method.

URI Parameter name	Description
logon_process_id	Logon process identifier, identifier of started logon process.

Resource accepts JSON-object which contains information about the next authentication method.

Table 3-21 Parameters for starting next authentication method.

Parameter name	Description
endpoint_session_id	Endpoint session identifier. Identifier of endpoint, which will create logon session, check the Create Endpoint Session chapter for more information.
method_id	Authentication method identifier. Identifier of next method from authentication methods list that is supported by server. This parameter defines which method will be used for authentication, check Authentication methods. for more information.

Resource return JSON-object equal to JSON-object returned form resource for starting logon process, check the Created login session. for more information. Other requests to resources used for chained login described in the same chapter.

Example

In the following example, user with username “JSmith” from users’ repository “COMPANY”, tries to logon from endpoint with session identifier “LTcnApseCzyFQCnDbxdid3rEvkgWk2f2”, user use chain with LDAP password and counter based one-time password, user authenticates to the NAM event.

```
HTTP POST
https://authserver.example.com/api/v1/logon
Request
{
"method_id":"LDAP_PASSWORD:1",
"user_name":"COMPANY\\JSmith",
"event":"NAM",
"endpoint_session_id":"LTcnApseCzyFQCnDbxdid3rEvkgWk2f2"
}
Response
{
"event_name":"NAM",
"reason":"PROCESS_STARTED",
"completed_methods":[],
"chains":[
{
"position":0,
"methods":["LDAP_PASSWORD:1", "HOTP:1"],
"is_enabled":true,
"name":"Password & HOTP",
"image_name":"default",
"apply_for_ep_owner":false,
"short_name":"",
"is_trusted":null
}],
"current_method":"LDAP_PASSWORD:1",
"status":"MORE_DATA",
"event_data_id":"OSLogon",
"plugins":[],
"msg":"Process started",
"logon_process_id":"cdlsXhjaEHOaPiwTlOck5p0xGaqgkNJB"
}
HTTP POST
https://authserver.example.com/api/v1/logon/cdlsXhjaEHOaPiwTlOck5p0xGaqgkNJB/
do_logon
Request
{
"response":
```

```
{
"answer":"P@$sW0rd"
},
"endpoint_session_id":"LTcnApseCzyFQCnDbxdid3rEvkgWk2f2"
}
Response
{
"event_name":"NAM",
4fa243fe48d6a13b9b50555fd1dcba273327398e9f665a945a73559"user_dn":"CN=John
Smith,CN=Users,DC=company,DC=local",
"user_sid":"S-1-5-21-4279545561-3293806183-1755797738-1602",
"user_upn":"JSmith@company.local",
"user_id":"395e14fc65ec11e58f04000c2951aca4",
"plugins":["LdapRules"],
"repo_id":"cd2ce1fa65eb11e58f04000c2951aca4",
"completed_methods":["LDAP_PASSWORD:1"],
"msg":"Continue with next login method",
"user_name_netbios":"COMPANY\\JSmith",
"reason":"METHOD_COMPLETED",
"repo_obj_id":"373bf00f9421394ab36c69f9d82283e4",
"chains":[
{
"position":0,
"methods":["LDAP_PASSWORD:1", "HOTP:1"],
"is_enabled":true,
"name":"Password & HOTP",
"image_name":"default",
"apply_for_ep_owner":false,
"short_name":"",
"is_trusted":null
}],
"current_method":"LDAP_PASSWORD:1",
"status":"NEXT",
"user_sid_hex":"010500000000000515000000d9ae14ff677e53c4ea58a768420600",
"event_data_id":"OSLogon",
"user_cn":"John Smith",
"logon_process_id":"cdlsXhjaEHOaPiwTlOck5p0xGaqgkNJB",
"user_name":"COMPANY\\John Smith"
}
HTTP POST
https://authserver.example.com/api/v1/logon/cdlsXhjaEHOaPiwTlOck5p0xGaqgkNJB/next
Request
{
"method_id":"HOTP:1",
"endpoint_session_id":"LTcnApseCzyFQCnDbxdid3rEvkgWk2f2"
}
Response
{
"event_name":"NAM",
"reason":"PROCESS_STARTED",
"completed_methods":["LDAP_PASSWORD:1"],
"chains":[
{
"position":0,
"methods":["LDAP_PASSWORD:1", "HOTP:1"],
"is_enabled":true,
"name":"Password & HOTP",
"image_name":"default",
"apply_for_ep_owner":false,
"short_name": "",
"is_trusted":null
}],
"current_method":"HOTP:1",
"status":"MORE_DATA",
"event_data_id":"OSLogon",
"plugins":[],
"msg":"Process started",
"logon_process_id":"cdlsXhjaEHOaPiwTlOck5p0xGaqgkNJB"
}
HTTP POST
https://authserver.example.com/api/v1/logon/cdlsXhjaEHOaPiwTlOck5p0xGaqgkNJB/
do_logon
Request
{
"response":
{
"answer":"150522"
},
"endpoint_session_id":"LTcnApseCzyFQCnDbxdid3rEvkgWk2f2"
}
Response
{
"event_name":"NAM",
"user_email":"jsmith@company.com",
"user_dn":"CN=John Smith,CN=Users,DC=company,DC=local",
"plugins":["LdapRules"],
"repo_id":"cd2ce1fa65eb11e58f04000c2951aca4",
"msg":"Welcome!",
"logon_process_id":"cdlsXhjaEHOaPiwTlOck5p0xGaqgkNJB",
"reason":"CHAIN_COMPLETED",
"repo_obj_id":"373bf00f9421394ab36c69f9d82283e4",
"user_cn":"John Smith",
"user_sid":"S-1-5-21-4279545561-3293806183-1755797738-1602",
"event_data_id":"OSLogon",
"user_id":"395e14fc65ec11e58f04000c2951aca4",
"user_upn":"JSmith@company.local",
"login_session_id":"NjHg8gVLMwYI84BDzlgFxNn5HI53ioI2",
"user_name_netbios":"COMPANY\\JSmith",
"completed_methods":["LDAP_PASSWORD:1", "HOTP:1"],
"chains":[
{
"position":0,
"methods":["LDAP_PASSWORD:1", "HOTP:1"],
"is_enabled":true,
"name":"Password & HOTP",
"image_name":"default",
"apply_for_ep_owner":false,
"short_name":"",
"is_trusted":null
}],
"current_method":"HOTP:1",
"status":"OK",
"user_sid_hex":"010500000000000515000000d9ae14ff677e53c4ea58a768420600",
"data_id":"OSLogon",
"user_name":"COMPANY\\John Smith"
}
```

3.4.4 1:N Authentication

The 1:N authentication could contain one or more chains and it equals non 1:N authentication. For starting 1:N authentication you should at first provide the unit identifier and set the 1:N flag to true and you should exclude user name from first request. The 1:N logon process is absolutely the same as non 1:N logon process, provide authentication data to the method and work with methods as you work with non 1:N logon process.

3.4.5 Read Available Chains

Before you start the logon process, you should get available authentication chains for this endpoint and user. The API has the following resource for this with URI.

```
/logon/
chains?user_name={user_name}&event={event}&endpoint_session_id={endpoint_session_i
d}
```

Resource provided by HTTP GET and has these parameters.

Table 3-22 *Parameters for getting available chains.*

URI parameter name	Description
user_name	User name, this parameter is optional, if you use this parameter, server return chain only for this user, if not – chains for all users.
event	Event identifier from server supported events list. Check “Supported events list” for getting events list.
endpoint_session_id	Endpoint session identifier, identifier of endpoint session, which want to get the chains. Check the Create Endpoint Session chapter for more information.

Resource return JSON-object, which contain list of available chains.

Table 3-23 *Object with available chains.*

Parameter name	Description
chains	Chain, this parameter using for wrapping of chains JSON-object array, chain object described in Chain JSON-object., check for more information about chain object.

Example

On the following example, endpoint with session “2NC69uuu0r63fYpqamMSJfYEVjIkSsbv” tries to get all trusted chains available for NAM event.

```
HTTP GET
https://authserver.example.com/api/v1/logon/
chains?event=NAM&endpoint_session_id=2NC69uuu0r63fYpqamMSJfYEVjIkSsbv&is_trusted=t
rue
Response
{
"chains":[
{
"methods":["LDAP_PASSWORD:1", "RADIUS:1"],
"is_trusted":true,
"name":"RADIUS and LDAP",
"position":0,
"is_enabled":true,
"image_name":"default",
"apply_for_ep_owner":false,
"short_name":""
},
{
"methods":["LDAP_PASSWORD:1", "TOTP:1"],
"is_trusted":true,
"name":"TOTP and LDAP",
"position":1,
"is_enabled":true,
"image_name":"default",
"apply_for_ep_owner":false,
"short_name":""
},
{
"methods":["LDAP_PASSWORD:1", "HOTP:1"],
"is_trusted":true,
"name":"HOTP and LDAP",
"position":2,
"is_enabled":true,
"image_name":"default",
"apply_for_ep_owner":false,
"short_name":""
}]
}
```

3.4.6 Priority of the Authentication Chains

The administrator of the Advanced Authentication appliance can manage the authentication chains priority and appliance will return chains in defined priority. You should use chain as it returned from the appliance. The chain which is highest on the priority list should be used first. For example, event has chain with LDAP authentication and chain with LDAP authentication and authentication by smartphone, and chain with LDAP authentication is higher, in this case for success authentication user should do LDAP authentication, it will be enough, but user also could add smartphone authentication after LDAP as additional authentication, appliance will allow this. If user not enroll method appliance will not return chains with this method. Same logic appears if the user could not use some of the method, for example, user have not phone number in user’s repository.

3.4.7 Delete Logon Process

You can delete the logon process, you can use the following resource with URI

```
/logon/{logon_process_id}?endpoint_session_id={endpoint_session_id}
```

Resource provided by HTTP DELETE and has these URI parameters.

Table 3-24 URI parameters for deleting logon process.

URI parameter name	Description
logon_process_id	Logon process identifier, identifier of process for deleting.
endpoint_session_id	Endpoint session identifier, identifier of endpoint session, which wants to delete the login process. Check the Create Endpoint Session chapter for more information.

The resource does not return any data, if deleting was successful the method return HTTP 200 status.

Example

On the following example, endpoint with session identifier “Sm7hACXcyvfl4ApoQ17g5QntE0q1ZW1K” deletes logon process with identifier “wLlxiBUCDDiTRJtL0xJPOeZpWfNcWfoH”.

```
HTTP DELETE
https://authserver.example.com/api/v1/logon/
wLlxiBUCDDiTRJtL0xJPOeZpWfNcWfoH?endpoint_session_id=Sm7hACXcyvfl4ApoQ17g5QntE0q1Z
W1K
Response
HTTP 200
```

3.5 Working With Users

3.5.1 About Users

Appliance can provide information about stored users; you can get information about all users from all repository. You can not get this information only about one user, not as users’ list. This information available only for full administrators.

3.5.2 Getting Information About User

To get user information use the resource with URI:

```
/users?user_name={user_name}&login_session_id={login_session_id}
```

The resource has URI parameters

Table 3-25 The parameter for get information about user

URI parameter	Description
user_name	The user name, appliance will return information about this user
login_session_id	The login session identifier, the identifier of login session for user with administrative role

The resource will return JSON object with user information.

Table 3-26 The object with information about user

Parameter name	Description
id	The user identifier
repo_id	The repository identifier
obj_id	The user identifier in repository
repo_name	The repository name
loginame	The user name in repository
user_name	The user name with repository, repo\user_name

Example

On the following example, administrator with login session identifier "cdlsXhjaEHOaPiwTlOck5p0xGaqgkNJB" will try to get information about user JSmith from COMPANY repository.

```
HTTP GET
https://authserver.example.com/api/v1/
users?user_name=COMPANY\JSmith&login_session_id=cdlsXhjaEHOaPiwTlOck5p0xGaqgkNJB
Response
{
"repo_id":"cd2ce1fa65eb11e58f04000c2951aca4",
"obj_id":"373bf00f9421394ab36c69f9d82283e4",
"id":"395e14fc65ec11e58f04000c2951aca4",
"user_name":"COMPANY\\John Smith",
"loginame":"JSmith",
"repo_name":"COMPANY",
}
```

3.6 Working With User's Data

3.6.1 About User's Data

Users' data – data that is stored for a specific event for a user. After logon, you can get access to the user's data. Users' data is a JSON-object with a custom structure and parameter names. Each event has different users' data, check event's description for more information.

Appliance supports this list of users' data storage.

Table 3-27 The list of users' data storage

Data storage identifier	Description
OSLogon	The data storage for operation systems logon processes and information
PasswordStore	The data storage for user's passwords

3.6.2 Read User's Data

For reading user's data use resource with URI

```
/users/{user_id}/data/{data_id}/
{data_parameter}?login_session_id={login_session_id}
```

Resource provided by HTTP GET and has these parameters.

Table 3-28 *URI parameters for reading user's data.*

URI parameter name	Description
user_id	User identifier, identifier of user, which wants to get assigned application data.
data_id	The data storage identifier, identifier of the storage which data will be read.
data_parameter	User's data parameter, optional parameter. If you use this parameter, resource return only this parameter from user's data.
login_session_id	Login session identifier, identifier of user login session. Check the Provide Authentication chapter for more information.

Resource return user's data as JSON-object, each event has different data format, check description for more information.

Table 3-29 *JSON-object for user's data.*

Parameter name	Description
data	Container for application data JSON-object, if method uses application data parameter, then this container will contain only application data parameter.

Example

On the following examples, user with identifier "6f4db9228c2711e4bb4100155d62a8b3" with login session identifier "cHIIvvAS4KteAG6MnVzJXx1I7TjVtnxP" gets data for storage "OSLogon" and then get only "test1" parameter from application data.

```
HTTP GET
https://authserver.example.com/api/v1/users/6f4db9228c2711e4bb4100155d62a8b3/data/
OSLogon?login_session_id=cHIIvvAS4KteAG6MnVzJXx1I7TjVtnxP
Response
{
"data":
{
"test1":"value 1",
"test3": "value 3",
"test2": "value 2"
}
}
HTTP GET
https://authserver.example.com/api/v1/users/6f4db9228c2711e4bb4100155d62a8b3/data/
OSLogon/test1?login_session_id=cHIIvvAS4KteAG6MnVzJXx1I7TjVtnxP
Response
{
"data":
{
"test1":"value 1"
}
}
```

3.6.3 Modifying User's Data

For modifying user's data use resource with URI.

`/users/{user_id}/data/{data_id}`

Resource provided by HTTP PATCH and has these parameters.

Table 3-30 *URI parameters for modifying user's data.*

URI parameter name	Description
user_id	User identifier, method will add application data to this user.
data_id	The data storage identifier, resource will add data for this storage, user should be logged on to this event.

Resource accept JSON-object with user's data.

Table 3-31 *Parameters for modifying user's data.*

Parameter name	Description
data	User's data container, this parameter contains all user's data, each event has different data format, check description for more information.
login_session_id	Login session identifier, identifier of user login session.

Resource does not returns any data, if the method is successful then it will return HTTP 200 status.

This resource has a flexible behavior, resource can update or modify data, for example, if user has no data, then resource add data for user for this event, if user has some data, then resource update parameters represented in request, add new parameters, if these parameters are not represented into current user's data, and delete parameters, which set to null into request.

Example

On the following example, user with identifier "6f4db9228c2711e4bb4100155d62a8b3" and login session with identifier "BFldNh3rLx39gjmun65gwJpLETjGF5fO" add user's data to storage "OSLogon".

```
HTTP PATCH
https://authserver.example.com/api/v1/users/6f4db9228c2711e4bb4100155d62a8b3/data/
OSLogon
Request
{
"data":
{
"test1":"value 1",
"test2":"value 2",
"test3":"value 3"
},
"login_session_id":"BFldNh3rLx39gjmun65gwJpLETjGF5fO"
}
Response
HTTP 200
```

3.6.4 Delete User's Data

For deleting user’s data use resource with URI

```
/users/{user_id}/data/{data_id}/
{data_parameter}?login_session_id={login_session_id}
```

Resource provided by HTTP DELETE and has these parameters.

Table 3-32 *URI parameters for deleting user’s data.*

URI parameter name	Description
user_id	User identifier, identifier of user, which wants to delete application data.
data_id	The data storage identifier, identifier of storage where data will be deleted.
data_parameter	User’s data parameter, optional parameter. If you use this parameter, resource deletes only this parameter from user’s data. If you do not use this parameter, resource deletes all application data.
login_session_id	Login session identifier, identifier of user login session.

Resource does not return data, if method was success, then return HTTP 200 status.

Example

On the following example, user with identifier “6f4db9228c2711e4bb4100155d62a8b3” with login session identifier “H0u1NPOpPs6foZonfAxoRRDxgTsSYuVM” deletes data parameter “test1” for storage “OSLogon” and then deletes all data for “OSLogon”.

```
HTTP
DELETE
https://authserver.example.com/api/v1/users/6f4db9228c2711e4bb4100155d62a8b3/data/
OSLogon/test1?login_session_id=H0u1NPOpPs6foZonfAxoRRDxgTsSYuVM
Response
HTTP 200
HTTP DELETE
https://authserver.example.com/api/v1/users/6f4db9228c2711e4bb4100155d62a8b3/data/
OSLogon?login_session_id=H0u1NPOpPs6foZonfAxoRRDxgTsSYuVM
Response
HTTP 200
```

3.7 Working With Login Sessions

3.7.1 About Login Sessions

After a user logon, the system creates a login session, to access protected information from the server use this session. For example, reading event data requires a login session identifier. The API allows reading the login session information and deleting the session.

3.7.2 Read Information About Login Sessions

For reading information about login session use resource with URI

```
/logon/sessions/{logon_session_id}?endpoint_session_id={endpoint_session_id}
```

Resource provided by HTTP GET and has these parameters.

Table 3-33 *URI parameters for reading information about login session.*

URI parameter name	Description
logon_session_id	Logon session identifier, identifier of session for reading.
endpoint_session_id	Endpoint session identifier, identifier of endpoint session.

Resource returns JSON-object, which contain information about login session.

Table 3-34 *JSON-object with information about login session.*

Parameter name	Description
event_name	Event name, name of event, which linked to this logon session.
repo_id	Users' repository identifier, identifier of users' repository which used for user logon.
user_id	User identifier, identifier of user which create this logon session.
repo_obj_id	For later usage.
sid	Logon session identifier.
user_name	User name, user name who create logon session.
data_id	Event data's identifier, identifier of event which creates this session.

Example

On the following example, we try to get information about the session with identifier "9hGUd3xuKE6VX0I8bVMWNeX1zNm0QfNd" from endpoint with session identifier "Wpeg2ek8IvsNF1hFZTXqYjPYvhCdUsZd".

```
HTTP GET
https://authserver.example.com/api/v1/logon/sessions/
9hGUd3xuKE6VX0I8bVMWNeX1zNm0QfNd?endpoint_session_id=Wpeg2ek8IvsNF1hFZTXqYjPYvhCdU
sZd
Response
{
"event_name":"NAM",
"repo_id":"6e0b696e8c2711e4bd9600155d62a8b3",
"user_id":"6f4db9228c2711e4bb4100155d62a8b3",
"repo_obj_id":"2d3c89ccb3ea7b4dacbdfda13e26f450",
"sid":"9hGUd3xuKE6VX0I8bVMWNeX1zNm0QfNd",
"user_name":"COMPANY\\JSmith",
"data_id":"OSLogon"
}
```

3.7.3 Delete Login Sessions

For deleting login session use resource with URI

```
/logon/sessions/{logon_session_id}?endpoint_session_id={endpoint_session_id}
```

Resource provided by HTTP DELETE and has these parameters.

Table 3-35 *URI parameters for deleting login session.*

URI parameter name	Description
logon_session_id	Logon session identifier, identifier of session for deleting.
endpoint_session_id	Endpoint session identifier, identifier of endpoint session.

Method does not return any data, if the method was success method return HTTP 200 status.

Example

On the following example we will delete the logon session with identifier “4Keuv7THWdSMR1H7mPc34O4mGoxTP0TP” with endpoint session identifier “Ex3hqCoF5A2ARWGv221AmPtEkWQDUi0U”.

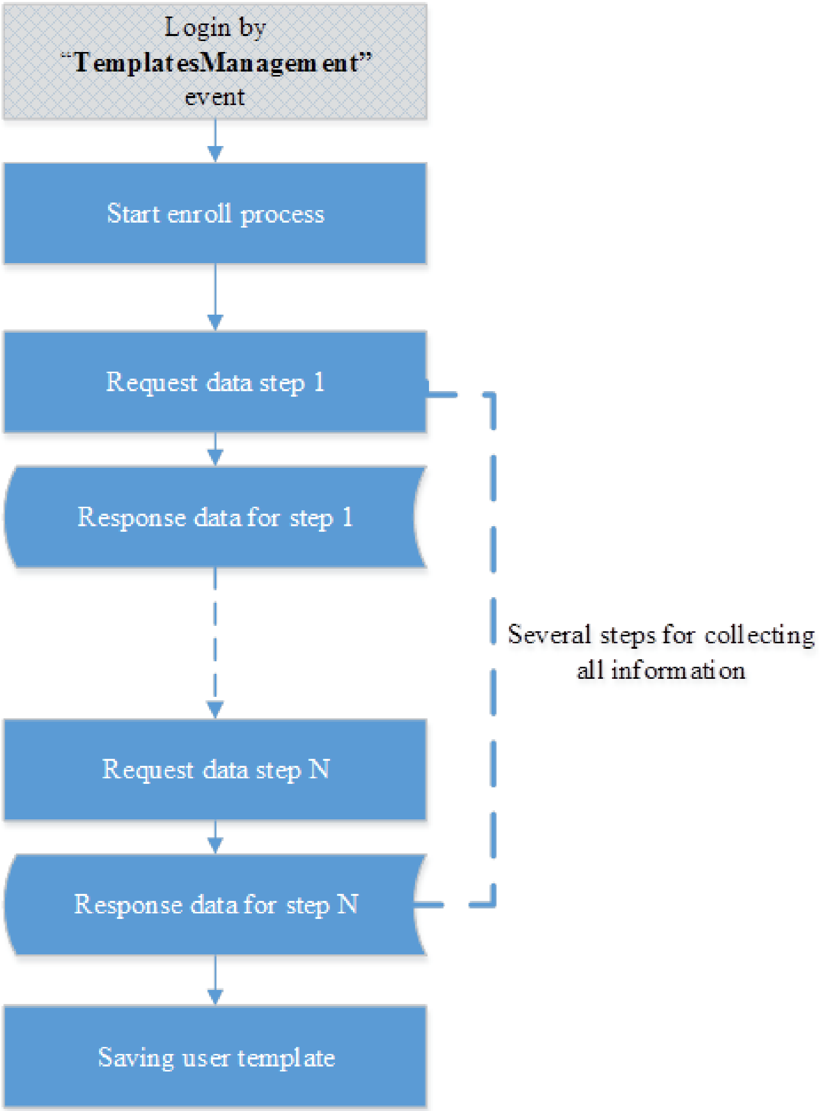
```
HTTP
DELETE
https://authserver.example.com/api/v1/logon/sessions/
4Keuv7THWdSMR1H7mPc34O4mGoxTP0TP?endpoint_session_id=Ex3hqCoF5A2ARWGv221AmPtEkWQDU
i0U
Response
HTTP 200
```

3.8 Working With Enrollment

3.8.1 About Enroll Process

The enrollment process collects information for creating of user templates. User templates can be created by several steps with the enroll process. The enroll process is wizard to user templates. Each user can create user templates into enroll process, administrator can assign enroll process results to another users. For starting the enrollment process the user should be authenticated in the “TemplatesManagement” event. Enroll process can be described by the following chart; it has several steps for collection information from user and saving the user template.

Figure 3-4 Detailed enrollment process chart.



3.8.2 Start Enroll Process

For starting enroll process you should make request to create process to resource with URI:

```
/enroll
```

Resource provided by HTTP POST and accepted JSON object with this parameters:

Table 3-36 Parameters for starting enroll process.

Parameter name	Description
login_session_id	Login session identifier, user should be logged to "TemplatesManagement" for creating enroll process.

Parameter name	Description
method_id	Authentication method identifier. Identifier of method form authentication methods list that supports by server.

Resource returns enroll process identifier, it is represented as JSON object with this parameter.

Table 3-37 JSON-object for started enroll process.

Parameter name	Description
enroll_process_id	Enroll process identifier, this identifier will be used in next enroll process step.

Example

On the following example user with login session identifier “Iz4awDMiRYcZh55SYt8awBz3Fcl1vikJ” starts the enroll process for the security question authentication method.

```
HTTP POST
https://authserver.example.com/api/v1/enroll
Request
{
"method_id":"SECQUEST:1",
"login_session_id":"Iz4awDMiRYcZh55SYt8awBz3Fcl1vikJ"
}
Response
{
"enroll_process_id":"WqQd4TwxPCz7q1tAKrWGzCtajg7Iav14"
}
```

3.8.3 Providing Data Into Enroll Process

You should provide data into enroll process after this process is started. Resource for providing data has URI:

```
enroll/{enroll_process_id}/do_enroll
```

Resource provided by HTTP POST and has this parameter.

Table 3-38 URI parameter for providing data into enroll process.

URI parameter	Description
enroll_process_id	Enroll process identifier, identifier of current enroll process. Start enroll process to get this identifier.

Resource accepts JSON object.

Table 3-39 *Parameter for providing data into enroll process.*

Parameter name	Description
login_session_id	Login session identifier, user should be logged to “TemplatesManagement” for providing enroll data.
response	Object with specific data for method. Data varies for each authentication method. Check method description for more information.

Resource returns the result for accepting the enroll data. Result can be different for each authentication method, for more information check description for authentication methods, but all responses contain some parameters which are the same for all authentication methods.

Table 3-40 *JSON-object with result for accepting the enroll data.*

Parameter name	Description
method_id	Authentication method identifier. Identifier of method from authentication methods list that is supported by the server. This parameter shows which authentication method currently works with enroll data.
status	Current status of enroll process, see status description for more information.
msg	Message from enroll process, this parameter is used for providing more information about enroll process.
reason	The detailed status of the enrollment process, please check authentication method description for more information.

Enroll process has the following status

Table 3-41 *Enroll process status.*

Status name	Description
OK	This status indicates that the enroll process has collected all necessary data and you can create or update the user template.
MORE_DATA	This status indicates that enroll process is waiting for more data from user, you should provide specific data to enroll process, for more information about data see description of authentication methods.
FAILED	This status indicate that the enroll process failed, for more information check message parameter from response. With this status you should start enroll process again.

Each enroll process is one or more steps for collecting all necessary data for the authentication method, each authentication method has a different number of steps and data, for more information see authentication method description.

Example

On the following example we provide data for security question authentication method, this method has two steps: first step is getting security question list from server, second step is providing answers for each security question. On example we have enroll process with identifier

“WqQd4TwxPCz7q1tAKrWGzCtajg7Iav14”, user logged to “TemplatesManagement” application with session identifier “7Ge4BCGDLKPyG5b6Mp7PBcKUsQouhpdX”. On the first step, we will get security question list.

```
HTTP POST
https://authserver.example.com/api/v1/enroll/WqQd4TwxPCz7q1tAKrWGzCtajg7Iav14/
do_enroll
Request
{
"response":{},
"login_session_id":"7Ge4BCGDLKPyG5b6Mp7PBcKUsQouhpdX"
}
Response
{
"questions":
{
"1":"What is your dog name?",
"0": "What is your favorite song?"
},
"method_id":"SECQUEST:1",
"msg":"Waiting for answers...",
"status":"MORE_DATA",
"reason":"SECQUEST_WAITING_ANSWERS"
}
```

We got question and servers change status to “MORE_DATA”, this means that the server is waiting for the answers to the questions. Now we will send answers, it will be the second step.

```
HTTP POST
https://authserver.example.com/api/v1/enroll/WqQd4TwxPCz7q1tAKrWGzCtajg7Iav14/
do_enroll
Request
{
"response":
{
"answers":
{
"1":"Spotty",
"0":"Yesterday"
}
},
"login_session_id":"7Ge4BCGDLKPyG5b6Mp7PBcKUsQouhpdX"
}
Response
{
"method_id":"SECQUEST:1",
"status":"OK",
"msg":"",
"reason":""
}
```

After the second step server accepted our data and return status “OK” it means that enrollment was successful.

3.8.4 Delete Enroll Process

You can stop enroll process by resource with URI:

```
/enroll/{enroll_process_id}?login_session_id={login_session_id}
```

Resource provided by HTTP DELETE and has these URI parameters.

Table 3-42 *URI parameters for stop enroll process.*

URI parameter name	Description
enroll_process_id	Enroll process identifier, identifier of enroll process, which will be stop.
login_session_id	Login session identifier, user should be logged to “TemplatesManagement” for deleting enroll process.

This resource does not return any data, if deleting enroll session was successful method return HTTP 200 status.

Example

On the following example a user with login session, identifier “4T1WEpCKumaIonjDBhoXbWkHhGg5Tk7Z” deletes enroll process with identifier “AfAXRqxrI1K2jgeJzrarD0NxllhpY9UW”.

```
HTTP DELETE
https://authserver.example.com/api/v1/enroll/
AfAXRqxrI1K2jgeJzrarD0NxllhpY9UW?login_session_id=4T1WEpCKumaIonjDBhoXbWkHhGg5Tk7Z
Response
HTTP 200
```

3.9 Working With User's Templates

3.9.1 About User's Templates

The user's templates contain authentication information associated with users. Each template is linked to a user and to an authentication method. When users try to logon using a specific authentication method, the server finds the associated user template and provide authentication. Users cannot use authentication methods without associated user templates. The Enrollment process creates user templates, check the Working With Enrollment chapter for more information. Each authentication method stores different information in the templates. The Authentication method can update a user’s template in the authentication process, also users or administrator can change templates manually, a user can edit only his own templates.

The API provide resources for working with user’s templates: creating, updating, reading and deleting.

3.9.2 Get User's Templates

This resource provide reading user templates. Resource has URI:

```
/users/{user_id}/templates?login_session_id={login_session_id}
```

Resource provided by HTTP GET.

Table 3-43 *URI parameters for get user’s template.*

URI parameter name	Description
user_id	User identifier, this identifier will be used for getting associated user templates. Administrator can use any user identifier, user should use only his own identifier.
login_session_id	Login session identifier, user’s login session identifier used for checking user access. User should be logged to “TemplatesManagement” for getting access to user template.

Resource return list of user templates. List present as array of JSON object.

Table 3-44 *JSON-object with user templates.*

Parameter name	Description
templates	This parameter is used for wrapping array of JSON object with user templates.
id	Identifier of user template.
method_id	Identifier of authentication method
is_enrolled	Boolean flag, when flag is true, then the template is enrolled and can be used for authentication, if flag is false, than the template is not enrolled and cannot be used for authentication.
method_title	Authentication method title
comment	Comment for user template

Example

On the following example a user with identifier “958ca11a7fa511e4b6ab00155d62a8b3” and session identifier “TBsifFiE4UJCyMnIyTkmY21kFctSxdwe” gets his own list of two user templates:

```
HTTP GET
https://authserver.example.com/api/v1/users/958ca11a7fa511e4b6ab00155d62a8b3/
templates?login_session_id=TBsifFiE4UJCyMnIyTkmY21kFctSxdwe
Response
{
"templates":[
{
"id":"958ca11b7fa511e4a32200155d62a8b3",
"method_id":"LDAP_PASSWORD:1",
"is_enrolled":true,
"method_title":"LDAP password",
"comment": "LDAP password template"
},
{
"id":"959165a47fa511e4bd1500155d62a8b3",
"method_id":"HOTP:1",
"is_enrolled":true,
"method_title":"HOTP",
"comment": "OATH HTOP template"}
]
}
```

3.9.3 Create User's Templates From Enroll Session

After a successful enroll process, you can assign enrolled data to the user template. This is provided by resource with URI:

```
/users/{user_id}/templates
```

Resource provided by HTTP POST and has this parameter:

Table 3-45 *URI parameter for assigning enrolled data.*

URI parameter name	Description
user_id	User identifier, identifier user who will be assign to new template.

An administrator can assign enrolled data and create user templates for any user.

Resource accept JSON-object with these parameters:

Table 3-46 *Parameters for assigning enrolled data.*

URI parameter	Description
enroll_process_id	Enroll process identifier, identifier of success enroll process, method will use data from this enroll process.
login_session_id	Login session identifier, user should be logged to “TemplatesManagement” for creating user template.
comment	Comment for created user template.

Resource return JSON-object with identifier of new user template.

Table 3-47 *JSON-object for new user template.*

Parameter name	Description
auth_t_id	Authentication template identifier, identifier of created user template

Example

On the following example, a user with identifier “e08c6b48810611e4b79300155d62a8b3” and login session identifier “33ebFAQBW1e1kTkKVzIRz5rf5dhv4OoE” creates a user template with enroll process identifier “nrXwfyy0l4QcXxYZNebzjs33rqS9UkG5”.

```
HTTP POST
https://authserver.example.com/api/v1/users/e08c6b48810611e4b79300155d62a8b3/
templates
Request
{
"enroll_process_id":"nrXwfyy0l4QcXxYZNebzjs33rqS9UkG5",
"comment":"Authentication template comment",
"login_session_id":"33ebFAQBW1e1kTkKVzIRz5rf5dhv4OoE"
}
Response
{
"auth_t_id":"9df84602842f11e4817e00155d62a8b3"
}
```

3.9.4 Assign User's Template To Another User

You can assign created user template to another user. This way a user can impersonate another user using his own authentication method. An example would be a user having 2 accounts, an admin account and a normal user account. He will only enroll authentication methods for his normal user account but will link his methods to his admin account. Now he can logon using the username of the admin account and the method of the normal account and this will be logged in the audit log.

Another use case would be group accounts where multiple users are linked too.

To link a template to another user use the resource with URI:

```
/users/{user_id}/templates
```

Resource provided by HTTP POST and has this parameter:

Table 3-48 *URI parameter for assign user template to another user.*

URI parameter name	Description
user_id	User identifier, identifier user who will be assign to template.

Resource accept JSON-object with these parameters:

Table 3-49 *Parameters for assign user template to another user.*

Parameter name	Description
auth_t_id	Authentication user template identifier, identifier of user template which will be assign to another user.
login_session_id	Login session identifier, user should be logged to “TemplatesManagement” for creating user template.

Resource return JSON-object with identifier of user’s template.

Table 3-50 *JSON-object for assigned template.*

Parameter name	Description
auth_t_id	Authentication template identifier, identifier of user template

This resource just links the user template to another user, new user can't modify template, user can only read assigned template.

Example

On the following example, user with login session “sC7PfOjvHt7OMgIccEHjK7b9XGlGNSoj” assign template with identifier “ba3cf01e845311e4841900155d62a8b3” to user with identifier “76cc2a62845411e4bd6c00155d62a8b3”.

```
HTTP POST
https://authserver.example.com/api/v1/users/76cc2a62845411e4bd6c00155d62a8b3/
templates
Request
{
"auth_t_id":"ba3cf01e845311e4841900155d62a8b3",
"login_session_id":"sC7PfOjvHt7OMgIccEHjK7b9XGlGNSoj"}
Response
{ "auth_t_id":"ba3cf01e845311e4841900155d62a8b3" }
```

3.9.5 Updating User's Template

You can update existing user template, this is provided by resource with URI:

`/users/{user_id}/templates/{auth_template_id}`

Resource provided by HTTP PUT and has this parameter:

Table 3-51 URI parameter for updating user template.

URI parameter name	Description
user_id	User identifier, identifier user which template will be update.
auth_template_id	Authentication template identifier, identifier of template which will be update.

Administrator can update user template for all users, other users can update own templates.

Resource accept JSON-object with these parameters:

Table 3-52 Parameters for updating user template.

Parameter name	Description
enroll_process_id	Enroll process identifier, identifier of success enroll process, method will use data from this enroll process for updating data in current user template.
login_session_id	Login session identifier, user should be logged to “TemplatesManagement” for changing user template.
comment	New comment for user template.

You can use one or both parameters for method, if you use enroll process identifier for method you will change user's template authentication data. When an template is updated all linked users will need to use the updated template

Resource does not return data, if operation was success method returns HTTP 200 status.

Example

On the following example, user with identifier “e08c6b48810611e4b79300155d62a8b3” update user template with identifier “ba3cf01e845311e4841900155d62a8b3” with new enrolled data form enroll process with identifier “l4rh0HvzJxNFBaO9d7PC4Uyq9YRc8EMY”.

```
HTTP PUT
https://authserver.example.com/api/v1/users/e08c6b48810611e4b79300155d62a8b3/
templates/ba3cf01e845311e4841900155d62a8b3
Request
{
"enroll_process_id":"l4rh0HvzJxNFBaO9d7PC4Uyq9YRc8EMY",
"comment":"Updated comment",
"login_session_id":"86oe9ebDvJ08UJvlG0O14ZARm3wIYNTB"
}
Response
HTTP 200
```

3.9.6 Delete User's Template

This resource provide deleting of user templates. Resource has URI

`/users/{user_id}/templates/{template_id}?login_session_id={login_session_id}`

Resource providing by HTTP DELETE and has this parameters:

Table 3-53 *URI parameters for deleting user template.*

URI parameter name	Description
user_id	User identifier, identifier of user which template will be deleted.
template_id	Template identifier, identifier of template for deleting.
login_session_id	Login session identifier, user’s login session identifier used for checking user access. User should be logged to “TemplatesManagement” for getting access to user template deleting.

Administrator can delete any user templates, other users can delete only their own templates.

This method doesn’t return any data, if deleting was successful method returns HTTP 200 status, if an error occurs, method returns error description.

Example

On the following example, user with the user identifier “e08c6b48810611e4b79300155d62a8b3” and login session identifier “GsnGHDvqOiEdPR3KUhKqy2kZq7l4RCcC” tries to delete his own template with identifier “9df84602842f11e4817e00155d62a8b3”.

```
HTTP DELETE
https://authserver.example.com/api/v1/users/e08c6b48810611e4b79300155d62a8b3/
templates/
9df84602842f11e4817e00155d62a8b3?login_session_id=GsnGHDvqOiEdPR3KUhKqy2kZq7l4RCcC
Response
HTTP 200
```

3.10 Working With Authentication Methods

The chapter will describe the authentication and the enrollment processes for each authentication method provided by appliance.

Before authentication, you should start a logon process and then provide data required by the authentication method. The data differs from method to method. For getting more information about authentication process, please check the Provide Authentication chapter.

For the enrollment, you should be authenticated at the appliance and have a login session identifier, then you should start enrollment process and provide the enrollment data, the enrollment data is different from method to method. If a method has optional parameter and default value was not present in parameter's description, that means – appliance will use default values from the authentication method's policy. For more information about enrollment process, please check the Working With Enrollment chapter.

3.10.1 Card Authentication Method

The card authentication method provide users' authentication by card. At the moment the method could work with the contactless cards (UID) and the certificate-based (PKI) cards. The card authentication method could be used as a single method in the authentication chain or as a part of the authentication chain.

Authentication

For the authentication, you should create a logon process with the card authentication method or continue with the card authentication method in case, when the authentication chain has more than one authentication methods. You should provide a JSON container with card's data by POST request. This authentication method supports 1:N logon and you could start a logon process without user name, you can define the card UID in unit_id parameter on a logon process creation. If you will use a chain with many authentication methods, you should get card identifier before you start 1:N logon process, and you should provide a card identifier at second time if your chain has the card authentication method. This method also work in non 1:N logon.

Table 3-54 *The authentication data for the card authentication method*

Parameter name	Parameter value
card_uid	The card's UID
card_cert	The hex-string with certificate in DER format, optional parameter used for PKI-based smartcards

Resource will return a JSON-object with information about current state of the authentication.

Example

On the following example, the user JSmith from the COMPANY repository will try to authenticate with this own PKI-based smartcard, user already has an endpoint session with the identifier "P7p3JJuenqo0SnyJ4HnbRbbJIqDhtt0u".

```
HTTP POST
https://authserver.example.com/api/v1/logon
Request
{
"method_id":"CARD:1",
"user_name":"COMPANY\\JSmith",
"event":"NAM",
"endpoint_session_id":"P7p3JJuenqo0SnyJ4HnbRbbJIqDhtt0u"
}
Response
{
"current_method":"CARD:1",
"chains":[
{
"name":"Smartcard",
"methods":["CARD:1"],
"image_name":"default",
"apply_for_ep_owner":false,
"position": 0,
"is_enabled":true,
"is_trusted":true,
"short_name": ""
}],
"completed_methods":[],
"msg": "Process started",
"logon_process_id":"t9HZES2DFD6vDjhA3wNtizyjAuqMTrwi",
"plugins":[],
"event_data_id":"OSLogon",
"event_name":"NAM",
"status":"MORE_DATA",
"reason":"PROCESS_STARTED"
}
HTTP POST
https://authserver.example.com/api/v1/logon/t9HZES2DFD6vDjhA3wNtizyjAuqMTrwi/
do_logon
Request
{
"response":
{
"card_uid":"0e00000000005c9e1e3a2d7532c30e00000046",
"card_cert":"3082070d308204f5a0030201020213460000000ec332752d3a1e9e5c00000000000e3
00d06092a864886f70d01010d0500304531133011060a0992268993f22c6401191603636f6d3118301
6060a0992268993f22c640119160861757468617361733114301206035504031308b617574686173617
32d6361301e170d3135303930383135353432365a170d3136303930373135353432365a30818131133
011060a0992268993f22c6401191603636f6d31183016060a0992268993f22c640119160861757468 6
17361733111300f060355040b130841414155736572733117301506035504031 30e446d69747279204
76f6c7562657631243022 06092a864886f70d010901161564676f6c75626576406175746861736173 2
e636f6d30819f300d06092a864886f70d010101050003818d0030818902818100df8ebdf44a7e6dd1e
4c64c2b2a96865c42410f73504f90b73836d31f4942beb44957af89d503f09c5c0fc9e3b44e96b3bd2
dbdd7235e63090e23051f657c575c18c22a3edebb3264b6d7cabc28eebc7227156c1b5c795c959587a
5a82afb22849125af3f2699030de7c2fe44cbb75097c2d123561360d9bc993073088db1b03d0203010
001a382033b30820337301706092b0601040182371402040a1e0800550073006500723 01d0603551d0
e041604144d18c67db47d1e4cd23f273864d8412d6f0493a9300e0603551d0f0101ff0404030205a03
01f0603551d2304183016801 48876d728949259f5462a224099af4cd41bc973523081fb0603551d1f0
481f33081f03081eda081eaa081e78681b26c6461703a2f2f2f434e3d6175746861736173 2d63612c4
34e3d4141415244532c434e3d4344502c434e3d5075626c69632532304b6579253230536572766963 6
5732c434e3d53657276696365732c434e3d436f6e66696775726174696f6e2c44433d6175746861736
1732c44433d636f6d3f63657274696669636174655265766f636174696f6e4c6973743f626173653f6
f626a656374436c6173733d63524c446973747269627574696f6e506f696e7486306874747 03a2f2f6
465762e6175746861736173 2e636f6d2f43657274446174612f6175746861736173 2d63612e63726c3
082011206082b0601050507010104820104308201003081ab06082b0601050507300286819e6c64617
03a2f2f2f434e3d61757468617361732d63612c434e3d4149412c434e3d5075626c69632532304b657
92532305365727669636573c434e3d5365727669636573c434e3d436f6e6669677572617469f6e2
c44433d61757468617361732c44433d636f6d3f63414365727469666963617465f626173653f6f626
a656374436c6173733d63657274696669636174696f6e417574686f72697479305006082b060105050
7300286446874747073a2f2f6465762e61757468617361732e636f6d2f436572744461746612f4141415
244532e61757468617361732e636f6d5f61757468617361732d63612e6372743029060355 1d2504223
020060a2b0601040182370a030406082b0601050507030406082b0601050507030230470603551d110
440303ea025060a2b060104018237140203a0170c1564676f6c7562657640617574686173617 32e636
f6d811564676f6c756265764061757468617361732e636f6d304406092a864886f70d01090f0437303
5300e06082a864886f70d030202020080300e06082a864886f70d030402020080300706052b0e03020
7300a06082a864886f70d0307300d06092a864886f70d01010d0500038202010056fd6f97f650c150f
3116a838f1ccf6856532e4dfe8d60a662804d120fe6ae116ed8ce76445da7375f75b6ee7827a6d5f33
011401edbda3262b6689fdda820ee66b5c3c0584424b249716682c45a4ffd6ff34e650f52061e59527
ea44fe624e86057a384f70d7393af19ec282435d3859fbe60c2ed4114df963a92e4df97e906da74ff6
082edfca0ae18b6610915890c4983d84f49ddc899ed2e46f007841d88f775236dc3b7ff0d0de3ddc98
0ebc0cf2cb5cf80b59f8d409d4a3d161704db4a2a6a69a15ef953564c24fea005b68a59056abf5dd33
4b50375e17c84f9bf8f6e81428d50f6b63c0d101f433412871142de893376103c277a0f1b4c68c3ae0
87fcbff3c1feed85289f156c6c092176a885fce5320d14cee1b696a1d65fc6b0a84b4bb068fc93388a
173824f2c6f9af0fb509670405d56e4ecfbace31f4095bc63d0c13df1487eb60ebd3276014374e35d6
be4b0e8149e7495cefb8a9b0e1425226211e676882b48893aec8b24895e6b2f96f8bf34ca715df0435
e94a3562940c08cfedf2ac2a966cdd104197fc37391d0fa9d5cacd39931d1b55641d58e2dde8029b36
3c7d494cb9d88c4d55ee408954752f112095a3f04adc6cfd9fdb7fa23e9d33d3d1c84e539b93fa3939
daa696a70da203307de5923156380c9a827c1d63ca4ca9d0114fa243fe48d6a13b9b50555fd1dcba27
3327398e9f665a945a73559"
},
"endpoint_session_id":"P7p3JJuenqo0SnyJ4HnbRbbJIqDhtt0u"
}
Response
{
"event_data_id":"OSLogon",
"user_mobile_phone":"+16086783619",
"msg":"Welcome!",
"user_dn":"CN=JSmith,CN=Users,DC=company,DC=com",
"data_id":"OSLogon",
"user_email":"jsmith@company.com",
"repo_id":"d3fba00652d211e5a19a000c2951aca4",
"chains":[
{
"name":"Smartcard",
"methods":["CARD:1"],
"image_name":"default",
"apply_for_ep_owner":false,
"position": 0,
"is_enabled":true,
"is_trusted":true,
"short_name": ""
}],
"user_sid":"S-1-5-21-4279545561-3293806183-1755797738-500",
"user_cn":"JSmith",
"completed_methods":["CARD:1"],
"user_sid_hex":"010500000000000515000000d9ae14ff677e53c4ea58a768f40100",
"logon_process_id":"t9HZES2DFD6vDjhA3wNtizyjAuqMTrwi ",
"current_method":"CARD:1",
"user_name":"COMPANY\\JSmith",
"repo_obj_id":"2d3c89ccb3ea7b4dacbdfda13e26f450",
"login_session_id":"DirIO8s41TbT1lm7Dh5BNfK6gRTHhXTl",
"event_name":"NAM",
"status":"OK",
"user_name_netbios":"COMPANY\\JSmith",
"plugins":["LdapRules"],
"user_id":"ab2b845652d311e5a19a000c2951aca4",
"reason": "CHAIN_COMPLETED"
}
```

Enrollment

For the enrollment, the card authentication method you should start an enrollment process and provide an enrollment data as a JSON container.

Table 3-55 *The enrollment data for the card authentication method*

Parameter name	Parameter value
card_uid	The card's UID
card_cert	The hex-string with certificate in DER format, optional parameter used for PKI-based smartcards

Resource will return a JSON object with status of the enrollment process.

Example

On the following example, the user JSmith from the COMPANY repository will try to enroll the smartcard authentication method for PKI-based smartcard, user has a login session with the identifier “mDclxFuBrCNAHddXazdAeu06bsxlyfqY”.

```
HTTP POST
https://authserver.example.com/api/v1/enroll
Request
{
"method_id":"CARD:1",
"login_session_id":"mDclxFuBrCNAHddXazdAeu06bsxlyfqY"
}
Response
{
"enroll_process_id":"23AEHu4uoISJBZ3KEjUo3Y7s2MN7MQe2"
}
HTTP POST
https://authserver.example.com/api/v1/enroll/23AEHu4uoISJBZ3KEjUo3Y7s2MN7MQe2 /
do_enroll
Request
{
"response":
{
"card_uid":"0e00000000005c9e1e3a2d7532c30e00000046",
"card_cert":"3082070d308204f5a0030201020213460000000ec332752d3a1e9e5c00000000000e3
00d06092a864886f70d01010d0500304531133011060a0992268993f22c6401191603636f6d3118301
6060a0992268993f22c64011916086175746861736173311430120603550403130b617574686173617
32d6361301e170d3135303930383135353432365a170d3136303930373135353432365a30818131133
011060a0992268993f22c6401191603636f6d31183016060a0992268993f22c6401191608617574686
17361733111300f060355040b130841414155736572733117301506035504031 30e446d69747279204
76f6c7562657631243022060 92a864886f70d010901161564676f6c7562657640617574686173617 32
e636f6d30819f300d06092a864886f70d010101050003818d0030818902818100df8ebdf44a7e6dd1e
4c64c2b2a96865c42410f73504f90b73836d31f4942beb44957af89d503f09c5c0fc9e3b44e96b3bd2
dbdd7235e63090e23051f657c575c18c22a3edebb3264b6d7cabc28eebc7227156c1b5c795c959587a
5a82afb22849125af3f2699030de7c2fe44cbb75097c2d123561360d9bc993073088db1b03d0203010
001a382033b308203373017060 92b06010401823714020 40a1e08005500730065007230 1d0603551d0
e041604144d18c67db47d1e4cd23f273864d8412d6f0493a9300e0603551d0f0101ff0404030205a03
01f0603551d23041830168014 8876d728949259f5462a224099af4cd41bc973523081fb0603551d1f0
481f33081f03081eda081eaa081e78681b26c6461703a2f2f2f434e3d617574686173617 32d63612c4
34e3d41414152445 32c434e3d4344502c434e3d5075626c69632532304b657925323053657276696 36
5732c434e3d53657276696365732c434e3d436f6e66696775726174696f6e2c44433d617574686173 6
1732c44433d636f6d3f636572746966696361746552657 66f636174696f6e4c6973743f626173653f6
f626a656374436c6173733d63524c446973747269627574696f6e506f696e74863068747470 3a2f2f6
465762e617574686173617 32e636f6d2f43657274446174612f617574686173617 32d63612e63726c3
082011206082b0601050507010104820104308201003081ab06082b060105050730028681 9e6c64617
03a2f2f2f434e3d617574686173617 32d63612c434e3d4149412c434e3d5075626c69632532304b657
925323053657276696365732c434e3d53657276696365732c434e3d436f6e66696775726174696f6e2
c44433d617574686173617 32c44433d636f6d3f634143657274696669636174653f626173653f6f626
a656374436c6173733d636572746966696361746 96f6e417574686f726974793050 06082b060105050
730028644687474703a2f2f6465762e617574686173617 32e636f6d2f43657274446174612f4141415
244532e617574686173617 32e636f6d5f617574686173617 32d63612e63727430290603551d2504223
020060a2b0601040182370a030406082b0601050507030406082b06010505070302304706 03551d110
440303ea025060a2b060104018237140203a0170c1564676f6c7562657640617574686173617 32e636
f6d811564676f6c7562657640617574686173617 32e636f6d304406092a864886f70d01090f0437303
5300e06082a864886f70d030202020080300e06082a864886f70d030402020080300706052b0e03020
7300a06082a864886f70d0307300d06092a864886f70d01010d05000382020100 56fd6f97f650c150f
3116a838f1ccf6856532e4dfe8d60a662804d120fe6ae116ed8ce76445da7375f75b6ee7827a6d5f33
011401edbda3262b6689fdda820ee66b5c3c0584424b249716682c45a4ffd6ff34e650f52061e59527
ea44fe624e86057a384f70d7393af19ec282435d3859fbe60c2ed4114df963a92e4df97e906da74ff6
082edfca0ae18b6610915890c4983d84f49ddc899ed2e46f007841d88f775236dc3b7ff0d0de3ddc98
0ebc0cf2cb5cf80b59f8d409d4a3d161704db4a2a6a69a15ef953564c24fea005b68a59056abf5dd33
4b50375e17c84f9bf8f6e81428d50f6b63c0d101f433412871142de893376103c277a0f1b4c68c3ae0
87fcbff3c1feed85289f156c6c092176a885fce5320d14cee1b696a1d65fc6b0a84b4bb068fc93388a
173824f2c6f9af0fb509670405d56e4ecfbace31f4095bc63d0c13df1487eb60ebd3276014374e35d6
be4b0e8149e7495cefb8a9b0e1425226211e676882b48893aec8b24895e6b2f96f8bf34ca715df0435
e94a3562940c08cfedf2ac2a966cdd104197fc37391d0fa9d5cacd39931d1b55641d58e2dde8029b36
3c7d494cb9d88c4d55ee408954752f112095a3f04adc6cfd9fdb7fa23e9d33d3d1c84e539b93fa3939
daa696a70da203307de5923156380c9a827c1d63ca4ca9d0114fa243fe48d6a13b9b50555fd1dcba27
3327398e9f665a945a73559"
},
"login_session_id":"mDclxFuBrCNAHddXazdAeu06bsxlyfqY"
}
```

Response

```
{
"reason":"",
"msg":"",
"status":"OK",
"method_id":"CARD:1"
}
```

3.10.2 Email Authentication Method

The email one-time password authentication method provide authentication by one-time password that will send to user's email. The appliance use an email address from LDAP repository, if user have not email address in repository, he could not use this authentication method. The email one-time password authentication method could be used as a single method in the authentication chain or as a part of the authentication chain.

Authentication

For the authentication, you should create a logon process with the email one-time password authentication method or continue with the email one-time password authentication method in case, when the authentication chain has more than one authentication methods. You should provide a JSON container with user's one-time password by POST request. This method have two step for authentication, on first step you should send empty data for sending email with one-time password to user, on second step you should provide one-time password.

Table 3-56 *The authentication data for the email authentication method*

Parameter name	Description
answer	This parameter contain user's one-time password

Resource will return a JSON-object with information about current state of the authentication.

The email one-time password authentication method supports the list of the authentication reasons.

Table 3-57 *The email method's authentication reasons*

Reason value	Description
OTP_CANNOT_SEND	The appliance cannot send by email one-time password
OTP_TOO_MANY_SENT	The appliance was sent too many one-time passwords
OTP_WAITING_PASSWORD	The authentication method waiting for one-time password
OTP_NO_PASSWORD	The password provided for the authentication was empty
OTP_PASSWORD_EXPIRED	The one-time password was expired
OTP_WRONG_PASSWORD	The one-time password was wrong
OTP_TOO_MANY_REQUESTS	The appliance got too many requests

Example

On the following example, the user JSmith from the COMPANY repository will try to authenticate by the email one-time password authentication method, user already has an endpoint session with the identifier "P1npGvMizbs4HiFsiLX5h3FquTB5tfJj".

```
HTTP POST
https://authserver.example.com/api/v1/logon
Request
{
"method_id":"EMAIL_OTP:1",
"user_name":"COMPANY\\JSmith",
"event":"NAM",
"endpoint_session_id":"P1npGvMizbs4HiFsiLX5h3FquTB5tfJj"
}
Response
{
"reason":"PROCESS_STARTED",
"current_method":"EMAIL_OTP:1",
"msg":"Process started",
"chains":[
{
"is_trusted":true,
"is_enabled":true,
"short_name":"",
"position":0,
"methods":["EMAIL_OTP:1"],
"name":"Email",
"apply_for_ep_owner":false,
"image_name":"default"
}],
"event_data_id":"OSLogon",
"status":"MORE_DATA",
"plugins":[],
"logon_process_id":"Imyr6xEyKTsVCuvj0s2dRz6j6rtCqfFK",
"completed_methods":[],
"event_name":"NAM"
}
HTTP POST
https://authserver.example.com/api/v1/logon/Imyr6xEyKTsVCuvj0s2dRz6j6rtCqfFK/
do_logon
Request
{
"response":{},
"endpoint_session_id":"P1npGvMizbs4HiFsiLX5h3FquTB5tfJj"
}
Response
{
"reason":"OTP_WAITING_PASSWORD",
"current_method":"EMAIL_OTP:1",
"msg":"OTP password sent, please enter",
"chains":[
{
"is_trusted":true,
"is_enabled":true,
"short_name":"",
"position":0,
"methods":["EMAIL_OTP:1"],
"name":"Email",
"apply_for_ep_owner":false,
"image_name":"default"
}],
"event_data_id":"OSLogon",
"status":"MORE_DATA",
"plugins":[],
"logon_process_id":"Imyr6xEyKTsVCuvj0s2dRz6j6rtCqfFK",
"completed_methods":[],
"event_name": "NAM"
}
HTTP POST
https://authserver.example.com/api/v1/logon/Imyr6xEyKTsVCuvj0s2dRz6j6rtCqfFK/
do_logon
Request
{
"response":
{
"answer":"12348765"
},
"endpoint_session_id":"P1npGvMizbs4HiFsiLX5h3FquTB5tfJj"
}
Response
{
"reason":"CHAIN_COMPLETED",
"user_id":"ab2b845652d311e5a19a000c2951aca4",
"event_name":"NAM",
"chains":[
{
"is_trusted":true,
"is_enabled":true,
"short_name":"",
"position":0,
"methods":["EMAIL_OTP:1"],
"name":"Email",
"apply_for_ep_owner":false,
"image_name":"default"
}],
"data_id":"OSLogon",
"repo_obj_id":"2d3c89ccb3ea7b4dacbdfda13e26f450",
"msg":"Welcome!",
"plugins":["LdapRules"],
"user_name":"COMPANY\\JSmith",
"user_email":"jsmith@company.com",
"current_method":"EMAIL_PASSWORD:1",
"user_sid":"S-1-5-21-4279545561-3293806183-1755797738-500",
"status":"OK",
"logon_process_id":"Imyr6xEyKTsVCuvj0s2dRz6j6rtCqfFK",
"user_dn":"CN=JSmith,CN=Users,DC=company,DC=com",
"user_sid_hex":"010500000000000515000000d9ae14ff677e53c4ea58a768f40100",
"login_session_id":"ugn5m5Hm9ow5ejuvBUz7SQi3SWazoHS8",
"user_mobile_phone":"+16086783619",
"event_data_id":"OSLogon",
"completed_methods":["EMAIL_PASSWORD:1"],
"user_name_netbios":"COMPANY\\JSmith",
"repo_id":"d3fba00652d211e5a19a000c2951aca4",
"user_cn":"JSmith"
}
```

Enrollment

The email one-time password authentication method could not be enrolled, this method based on the user’s LDAP attribute and all users from repository with email addresses could authenticate by this method.

3.10.3 Emergency Password Authentication Method

The emergency password authentication method is a method for the authentication by password with lifetime and usage counter. This method created for cases, when user should be authorized, but user could not use other method (smartphone or SMS authentication). The emergency password authentication method could be used as a single method in the authentication chain or as a part of the authentication chain.

Authentication

For the authentication, you should create a logon process with the emergency password authentication method or continue with the emergency password authentication method in case, when the authentication chain has more than one authentication methods. You should provide a JSON container with user’s emergency password by POST request.

Table 3-58 *The authentication data for the emergency password authentication method*

Parameter name	Description
answer	This parameter contain user’s emergency password

Resource will return a JSON-object with information about current state of the authentication.

The password authentication method supports the list of the authentication reasons.

Table 3-59 *The emergency password method’s authentication reasons.*

Reason value	Description
EMERG_PASSWORD_EXPIRED	The emergency password was expired
EMERG_PASSWORD_INEFFECTIVE	The emergency password was ineffective
EMERG_PASSWORD_EXHAUSTED	The maximum login count was exhausted

Example

On the following example, the user JSmith form the COMPANY repository will try to authenticate by the emergency password, user has an endpoint session with the identifier “zQ9YQ1Txpax09iRBBTQJN71tSDgsMiuA”.

```
HTTP POST
https://authserver.example.com/api/v1/logon
Request
{
"method_id":"EMERG_PASSWORD:1",
"user_name":"COMPANY\\JSmith",
"event":"NAM",
"endpoint_session_id":"zQ9YQ1Txpax09iRBBTQJN71tSDgsMiuA"
}
Response
{
"reason":"PROCESS_STARTED",
"current_method":"EMERG_PASSWORD:1",
"msg":"Process started",
"chains":[
{
"is_trusted":null,
"is_enabled":true,
"short_name":"",
"position": 0,
"methods":["EMERG_PASSWORD:1"],
"name": "Emergency password",
"apply_for_ep_owner":false,
"image_name":"default"
}],
"event_data_id":"OSLogon",
"status":"MORE_DATA",
"plugins":[],
"logon_process_id":"ULMzchEGWNPnutRROYDM18p24tTvQh2g",
"completed_methods":[],
"event_name":"NAM"
}
HTTP POST
https://authserver.example.com/api/v1/logon/ULMzchEGWNPnutRROYDM18p24tTvQh2g/
do_logon
Request
{
"response":
{
"answer":"emgP@ssw0rD"
},
"endpoint_session_id":"zQ9YQ1Txpax09iRBBTQJN71tSDgsMiuA"
}
Response
{
"reason":"CHAIN_COMPLETED",
"user_id":"ab2b845652d311e5a19a000c2951aca4",
"event_name":"NAM",
"chains":[
{
"is_trusted":null,
"is_enabled":true,
"short_name":"",
"position": 0,
"methods":["EMERG_PASSWORD:1"],
"name": "Emergency password",
"apply_for_ep_owner":false,
"image_name":"default"
}],
"data_id":"OSLogon",
"repo_obj_id":"2d3c89ccb3ea7b4dacbdfda13e26f450",
"msg":"Welcome (CHAP)!",
"plugins":["LdapRules"],
"user_name":"COMPANY\\JSmith",
"user_email":"jsmith@company.com",
"current_method":"EMERG_PASSWORD:1",
"user_sid":"S-1-5-21-4279545561-3293806183-1755797738-500",
"status":"OK",
"logon_process_id":"ULMzchEGWNPnutRROYDM18p24tTvQh2g",
"user_dn":"CN=JSmith,CN=Users,DC=company,DC=com",
"user_sid_hex":"010500000000000515000000d9ae14ff677e53c4ea58a768f40100",
"login_session_id":"6XFwPdePJ6y0pq9dBWFIcyerTVH2yMRJ",
"user_mobile_phone":"+16086783619",
"event_data_id":"OSLogon",
"completed_methods":["EMERG_PASSWORD:1"],
"user_name_netbios":"COMPANY\\JSmith",
"repo_id":"d3fba00652d211e5a19a000c2951aca4",
"user_cn":"JSmith"
}
```

Enrollment

For the enrollment, the emergency password authentication method you should start an enrollment process and provide an enrollment data as JSON container.

Table 3-60 *The enrollment data for the emergency password method*

Parameter name	Description
password	The new emergency password value
confirmation	The confirmation of the new emergency password value, if password will not equals to the confirmation – enrollment will fail.
max_logon_count	The maximum login count, method will be disabled, when maximum count will reach.
start_date	The start date, from that date emergency password authentication method will begin work. Value should be a string with date. This is an optional parameter.
end_date	The end date, from that date emergency password authentication method will stop work. Value should be a string with date. This is an optional parameter.

Resource will return a JSON object with status of the enrollment process.

The emergency password authentication method supports this set of the enrollment reasons.

Table 3-61 *The emergency password method's enrollment reasons*

Reason value	Description
PASSWORD_BAD_CONFIRMATION	The emergency password and the confirmation is not equals
PASSWORD_EMPTY	The provided emergency password or confirmation was empty
PASSWORD_UNCHANGED	The provided emergency password is equals to the current password
PASSWORD_TOO_SHORT	The provided emergency password is too small, please check password length in the policy values
PASSWORD_TOO_SIMPLE	The provided emergency password is too simple, user should use a stronger password

Example

On the following example, the user JSmith from the COMPANY repository will enroll the password authentication method, user has a login session with the identifier “wQabBzcnJTBqOqTdClHJTtrkpHFUzg40”.

```
HTTP POST
https://authserver.example.com/api/v1/enroll
Request
{
"method_id":"EMERG_PASSWORD:1",
"login_session_id":"wQabBzcnJTBqOqTdClHJTtrkpHFUzg40"
}
Response
{
"enroll_process_id":"j7wpbJRTJ3LHIhSFSn2UWAEnTA15ldTK"
}
HTTP POST
https://authserver.example.com/api/v1/enroll/j7wpbJRTJ3LHIhSFSn2UWAEnTA15ldTK/
do_enroll
Request
{
"response":
{
"password":"new_EmgP@$sw0rd",
"confirmation":"new_EmgP@$sw0rd",
"max_logon_count":5
},
"login_session_id":"wQabBzcnJTBqOqTdClHJTtrkpHFUzg40"
}
Response
{
"reason":"",
"msg":"",
"status":"OK",
"method_id":"EMERG_PASSWORD:1"
}
```

3.10.4 FIDO U2F Authentication Method

The FIDO U2F authentication method provide authentication by FIDO U2F standard, please check this standard description. The FIDO U2F authentication method should always be used together with a second factor like a Password or PIN.

Authentication

For the authentication, you should create a logon process with the FIDO U2F authentication method or continue with the FIDO U2F authentication method in case, when the authentication chain has more than one authentication methods. The FIDO U2F authentication method has two-step authentication process: on first step you should provide the application identifier and get the sign request from the appliance, on second step you should generate the sign response by FIDO U2F device and send it to the appliance.

Table 3-62 The authentication data for the FIDO U2F authentication method

Parameter name	Description
appId	The application identifier for login
signResponse	The sign response from the FIDO U2F device, this is JSON container, check FIDO U2F description for more information

Resource will return a JSON-object with information about current state of the authentication and the sign request.

Table 3-63 The response for the FIDO U2F authentication method

Response parameter name	Description
signRequests	The JSON container for an array of the sign requests from the appliance. Each sign request is JSON container.

The FIDO U2F authentication method supports the list of the authentication reasons.

Table 3-64 The FIDO U2F method's authentication reasons

Reason value	Description
U2F_ALL_TOKENS_COMPROMISED	The all tokens assigned to user was compromised
U2F_NO_TOKENS	The uses has not any assigned tokens.
U2F_WAITING_AUTH_RESPONSE	The FIDO U2F authentication method wait for the authentication request with the sign response data

Example

On the following example, the user JSmith from the COMPANY repository will try to authentication with the FIDO U2F authentication method, user already has an endpoint session with the identifier "eSFakgjm0o7eCb0hlUTVf1jzf2uGs9Y9".

```
HTTP POST
https://authserver.example.com/api/v1/logon
Request
{
"method_id":"U2F:1",
"user_name":"COMPANY\\JSmith",
"event":"NAM",
"endpoint_session_id":"eSFakgjm0o7eCb0hlUTVf1jzf2uGs9Y9"
}
Response
{
"event_data_id":"OSLogon",
"current_method":"U2F:1",
"completed_methods":[],
"plugins":[],
"status":"MORE_DATA",
"logon_process_id":"kytfmodEy4QmcMKgEb9cuGpLpNv9ooYp",
"chains":[
{
```

```
"image_name":"default",
"position":0,
"name":"FIDO U2F",
"short_name":"",
"is_enabled":true,
"apply_for_ep_owner":false,
"is_trusted":true,
"methods":["U2F:1"]
}],
"reason":"PROCESS_STARTED",
"event_name":"NAM",
"msg":"Process started"
}
HTTP POST
https://authserver.example.com/api/v1/logon/kytfmodEy4QmcMKgEb9cuGpLpNv9ooYp/
do_logon
Request
{
"response":
{
"appId":"https://demo.yubico.com"
},
"endpoint_session_id":"eSFakgjm0o7eCb0hlUTVf1jzf2uGs9Y9"
}
Response
{
"event_data_id":"OSLogon",
"current_method":"U2F:1",
"completed_methods":[],
"plugins":[],
"status":"MORE_DATA",
"logon_process_id":"kytfmodEy4QmcMKgEb9cuGpLpNv9ooYp",
"chains":[
{
"image_name":"default",
"position":0,
"name":"FIDO U2F",
"short_name":"",
"is_enabled":true,
"apply_for_ep_owner":false,
"is_trusted":true,
"methods":["U2F:1"]
}],
"reason":"U2F_WAITING_AUTH_RESPONSE",
"event_name":"NAM",
"method_id":"U2F:1",
"msg":"Waiting authentication response",
"signRequests":[
{
"appId":"https://demo.yubico.com",
"challenge":"aU1A3-GQkcYDmgH78bGeO55dXN4fEyeTHGObhRr8GgI",
"keyHandle":"qF5EfdOTSEK4kP3hlEd_Q2SD4kOYs6K9qZJR5Ve9EloEs6ns36GHG-
jGUKFD4JLMwOqsLsRnsipa-XsBzfF6Ow",
"version":"U2F_V2"
}],
}
HTTP POST
https://authserver.example.com/api/v1/logon/kytfmodEy4QmcMKgEb9cuGpLpNv9ooYp/
do_logon
Request
```

```
{
"response":
{
"signResponse":
{
"clientData":"eyJ0eXAiOiJuYXZpZ2F0b3IuaWQuZ2V0QXNzZXJ0aW9uIiwiY2hhbGxlbmdlIjoiZXYw
d3dLR3l5bV9YamdZZkExd2tNT3Y4bGpsOHphS0F5WGdBV3gyUW91OCIsIm9yaWdpbiI6Imh0dHBzOi8vZG
Vtby55dWJpY28uY29tIiwiY2lkX3B1YmtleSI6IiJ9",
"keyHandle":"qF5EfdOTSEK4kP3hlEd_Q2SD4kOYs6K9qZJR5Ve9EloEs6ns36GHG-
jGUKFD4JLMwOqsLsRnsipa-XsBzfF6Ow",
"signatureData":"AQAAAKowRAIgFrmzsrUHiUw2ixt20cXLjBRDo7-
UoqWqZlNsFcXZL4ECIH45ALRE86ijsKPv_r3zmzhoE34N3NqzgvRJ1f49C1UA"
}
},
"endpoint_session_id":"eSFakgjm0o7eCb0hlUTVf1jzf2uGs9Y9"
}
Response
{
"user_name_netbios":"COMPANY\\JSmith",
"event_data_id":"OSLogon",
"login_session_id":"dkQsnIAMsCxQrcJP9AgIB2tSCkDDZQML",
"user_name":"COMPANY\\JSmith",
"repo_id":"d3fba00652d211e5a19a000c2951aca4",
"data_id":"OSLogon",
"user_cn":"JSmith",
"current_method":"U2F:1",
"status":"OK",
"chains":[
{
"image_name":"default",
"position":0,
"name":"FIDO U2F",
"short_name":"",
"is_enabled":true,
"apply_for_ep_owner":false,
"is_trusted":true,
"methods":["U2F:1"]
}],
"reason":"CHAIN_COMPLETED",
"user_sid":"S-1-5-21-4279545561-3293806183-1755797738-500",
"user_mobile_phone":"+16086783619",
"user_id":"ab2b845652d311e5a19a000c2951aca4",
"logon_process_id":"kytfmodEy4QmcMKgEb9cuGpLpNv9ooYp",
"event_name":"NAM",
"user_sid_hex":"010500000000000515000000d9ae14ff677e53c4ea58a768f40100",
"completed_methods":["U2F:1"],
"plugins":["LdapRules"],
"repo_obj_id":"2d3c89ccb3ea7b4dacbdfda13e26f450",
"user_email":"jsmith@company.com",
"user_dn":"CN=JSmith,CN=Users,DC=company,DC=com",
"method_id":"U2F:1",
"msg":"Welcome!"
}
```

Enrollment

For the enrollment, the FIDO U2F authentication method you should start enrollment process and provide enrollment data. The FIDO U2F authentication method has two enrollment step: on first step you should get the registration request from the appliance, on second step you should provide the registration response from the device.

Table 3-65 The enrollment data for the FIDO U2F authentication method on first step

Parameter name	Description
userId	The identifier of the enrolling user, you could skip this parameter if you will use “userName”.
userName	The user name of the enrolling user, you could skip this parameter if you will use “userId”.
appId	The application identifier, for more information about the application identifier please check FIDO U2F documentation.

Table 3-66 The enrollment data for the FIDO U2F authentication method on second step

Parameter name	Description
registerResponse	The registration response generated by device, this is JSON container. For more information about registration response, please check FIDO U2F documentation.

Resource will return a JSON object with status of the enrollment process.

Table 3-67 The enrollment response for the FIDO U2F authentication method

Enrollment parameter	Description
registerRequests	The registration request generated by the appliance, this is array of a JSON containers. The parameter will return only for new devices.
signRequests	The sign requests from the appliance, if user already has enrolled device the appliance will return the sign request and user could just validate it by device, this is array of a JSON containers.

For more information about registration request, please check FIDO U2F documentation.

The FIDO U2F authentication method supports this set of the enrollment reasons.

Table 3-68 The FIDO U2F method’s enrollment reasons

Reason value	Description
U2F_WAITING_REG_RESPONSE	The FIDO U2F authentication method waiting for the registration response
U2F_NOT_ATTESTED	The FIDO U2F authentication method was not attested, you should upload your token manufacturer attestation certificate.

Example

On the following example, the user JSmith from the COMPANY repository will try to enroll the FIDO U2F authentication method by his token, user already has a login session with the identifier “WXogCpZcHlsGsmmYiXkcxbg1qJMHJ4eD”.

```
HTTP POST
https://authserver.example.com/api/v1/enroll
Request
{
"method_id":"U2F:1",
"login_session_id":"WXogCpZcHlsGsmmYiXkcxbg1qJMHJ4eD"
}
Response
{
"enroll_process_id":"hn0k1pTx1T6MQwOIbhBPUlLIopROig4Q"
}
HTTP POST
https://authserver.example.com/api/v1/enroll/hn0k1pTx1T6MQwOIbhBPUlLIopROig4Q/
do_enroll
Request
{
"response":
{
"userName":"COMPANY\\JSmith",
"appId":"https://demo.yubico.com"
},
"login_session_id":"WXogCpZcHlsGsmmYiXkcxbg1qJMHJ4eD"
}
Response
{
"signRequests":[],
"registerRequests":[
{
"appId":"https://demo.yubico.com",
"challenge":"Ij3TTvLi3H-oWNA18BDeIp_p1zN5bkzPipjQeuIbntQ",
"version":"U2F_V2"
}],
"msg":"Waiting register response ",
"reason":"U2F_WAITING_REG_RESPONSE",
"method_id":"U2F:1",
"status":"MORE_DATA"
}
HTTP POST
https://authserver.example.com/api/v1/enroll/hn0k1pTx1T6MQwOIbhBPUlLIopROig4Q /
do_enroll
Request
{
"response":
{
"registerResponse":
{
"clientData":"eyJ0eXAiOiJuYXZpZ2F0b3IuaWQuZmluaXNoRW5yb2xsbWVudCIsImNoYWxsZW5nZSI6
IjFNX2hrbkxka0Ixb1hySFQxWUZqSHA5NDRpT2w4N2kwd1B5UUV0MW5DYzQiLCJvcmlnaW4iOiJodHRwcz
ovL2RlbW8ueXViaWNvLmNvbSIsImNpZF9wdWJrZXkiOiIifQ",
"registrationData":"BQQ_jNAqemWkTfI9uL4CVo3eQSNIMe66u-
pasr8BMuu9maA764C7jDPqXMUWhw8_4tDxeUbM72VJH5fssT6F2vT5QKheRH3Tk0hCuJD94ZRHf0Nkg-
JDmLOivamSUeVXvRJaBLOp7N-hhxvoxlChQ-
CSzMDqrC7EZ7IqWvl7Ac3xejswggIcMIIBBqADAgECAgRyWMLqMAsGCSqGSIb3DQEBCzAuMSwwKgYDVQQD
EyNZdWJpY28gVTJGIFJvb3QgQ0EgU2VyaWFsIDQ1NzIwMDYzMTAgFw0xNDA4MDEwMDAwMDBaGA8yMDUwMD
kwNDAwMDAwMFowKzEpMCcGA1UEAwwgWXViaWNvIFUyRiBFRSBTZXJpYWwgMTQ4MDMzMjE1NzgwWTATBgcq
hkjOPQIBBggqhkjOPQMBBwNCAASisDmTIlQxnUH6SFTVfKGN62nMmz5Nga45nzI-
gRZDme8qlRRnPRV87L-
18LzHiQhT7lXPPxogZvTVE5uTizELoxIwEDAOBgorBgEEAYLECgECBAAwCwYJKoZIhvcNAQELA4IBAQG8z
Br5C3uVeBjVVaQzcWpgFqztyzEyw0EPNmFkEGwj2SqwbF0cLLaSmtQhSKoqOvOuU4k6aqFAyukyZZMVPZK
qAP0Vh0sCMpRMzpDvEZjO3v6gh5Z8bIDmtQAJ5B2nnILyVpc7DA7taj3dUrZzNMD8v-
bYjKdTsZJ_QzQstsewIPkoFOIRRtqta0iwkEFiX_cwR11IF-
USGcQHKUBoMX65JP9nY6DzQ3XHplOD3bHUOHsCi2MqBZU-
1fKOrQJpNP0w8cBQpSk_hsVTm7UiGW_FGrxrIKXfpGfCGICKDxCMfuWKIshu0HjP0pEhowAX1Ls1pie2So
K3-VEhYtkOFRLqMEUCIFnX5ZcO8ZWOKyF6R-
tAse3rYdzwFBq4LZzEeXUwn6oMAiEAwNuC4mkjobmeyilaLM4twOAicm-R_x_YKzcQcXIx5cU"
}
},
"login_session_id":"WXogCpZcHlsGsmmYiXkcxbg1qJMHJ4eD"
}
Response
{
"reason":"",
"msg":"Enroll complete",
"status":"OK",
"method_id":"U2F:1"
}
```

3.10.5 Fingerprint Authentication Method

The fingerprint authentication method provide users’ authentication by a fingerprint. The fingerprint authentication method could be used as a single method in the authentication chain or as a part of the authentication chain.

Authentication

For the authentication, you should create a logon process with the fingerprint authentication method or continue with the fingerprint authentication method in case, when the authentication chain has more than one authentication methods. This method accepts fingerprint data from user.

Table 3-69 *The authentication data for the fingerprint authentication method*

Parameter name	Description
capture	The capture container with user’s finger data

Table 3-70 *The capture container*

Parameter name	Description
captureStatus	The one of the capture status
Width	The width of the fingerprint image in pixels
Height	The height of the fingerprint image in pixels
Dpi	The dot per inch of the fingerprint image
BitsPerPixel	The bits per pixel (usually 8 bits)
BytesPerLine	The bytes per one line in image (include align)
Image	The fingerprint image encoded using base-64 in gray scale
ISO	The data of fingerprint in ISO/IEC 19794-2:2005 format

If you will work with ISO/IEC 19794-2:2005 standard, response to the appliance should contain only "ISO" parameter without others. You should use only image or ISO format, not both. Better authentication for fingerprint provided by ISO format, image of fingerprint could contain errors for many reasons, ISO format could avoid this problems. For generating fingerprint in ISO format, check your hardware vendor documentation.

Table 3-71 *The capture statuses list*

Status value	Status description
Ok	The capture was successful
Timeout	The capture got timeout
Error	The capture got error
NoReader	There is no reader for the capture

Example

On the following example, the user JSmith for the COMPANY repository will try to authenticate by the fingerprint authentication method, user already has an endpoint session with the identifier "7XrFFQB28O6pod8Y5ElCs4K75TYBAffy".

```
HTTP POST
https://authserver.example.com/api/v1/logon
Request
{
"method_id":"FINGER:1",
"user_name":"COMPANY\\JSmith",
"event":"NAM",
"endpoint_session_id":"7XrFFQB28O6pod8Y5ElCs4K75TYBAffy"
}
Response
{
"current_method":"FINGER:1",
"chains":[
{
"name":"Fingerprint",
"methods":["FINGER:1"],
"image_name":"default",
"apply_for_ep_owner":false,
"position": 0,
"is_enabled":true,
"is_trusted":true,
"short_name": ""
}],
"completed_methods":[],
"msg": "Process started",
"logon_process_id":"wZFkJ6kTvrbohSH6o5X5swGvZXDMoCQW",
"plugins":[],
"event_data_id":"OSLogon",
"event_name":"NAM",
"status":"MORE_DATA",
"reason":"PROCESS_STARTED"
}
HTTP POST
https://authserver.example.com/api/v1/logon/wZFkJ6kTvrbohSH6o5X5swGvZXDMoCQW/
do_logon
```

```
Request
{
"response":
{
"capture":
{
"BitsPerPixel":8,
"BytesPerLine":300,
"Dpi":500,
"Height":300,
"Image":"The base64 encoded image",
"Width":300,
"captureStatus":"Ok"
}
},
"endpoint_session_id":"7XrFFQB28O6pod8Y5ElCs4K75TYBAffy"
}
Response
{
"event_data_id":"OSLogon",
"user_mobile_phone":"+16086783619",
"msg":"Welcome!",
"user_dn":"CN=JSmith,CN=Users,DC=company,DC=com",
"data_id":"OSLogon",
"user_email":"jsmith@company.com",
"repo_id":"d3fba00652d211e5a19a000c2951aca4",
"chains":[
{
"name":"Fingerprint",
"methods":["FINGER:1"],
"image_name":"default",
"apply_for_ep_owner":false,
"position": 0,
"is_enabled":true,
"is_trusted":true,
"short_name": ""
}],
"user_sid":"S-1-5-21-4279545561-3293806183-1755797738-500",
"user_cn":"JSmith",
"completed_methods":["FINGER:1"],
"user_sid_hex":"010500000000000515000000d9ae14ff677e53c4ea58a768f40100",
"logon_process_id":"wZFkJ6kTvrbohSH6o5X5swGvZXDMoCQW",
"current_method":"RADIUS:1",
"user_name":"COMPANY\\JSmith",
"repo_obj_id":"2d3c89ccb3ea7b4dacbdfda13e26f450",
"login_session_id":"LN8WNzoregyveYy69igAAKA77p3GC0RB",
"event_name":"NAM",
"status":"OK",
"user_name_netbios":"COMPANY\\JSmith",
"plugins":["LdapRules"],
"user_id":"ab2b845652d311e5a19a000c2951aca4",
"reason": "CHAIN_COMPLETED"
}
```

Example 2

The second example same as the first, but user will authenticate with ISO fingerprint data, this authentication method is better that image data. The ISO data reduced in example:

HTTP POST

https://authserver.example.com/api/v1/logon

```
Request
{
  "method_id":"FINGER:1",
  "user_name":"COMPANY\\JSmith",
  "event":"NAM",
  "endpoint_session_id":"7XrFFQB28O6pod8Y5ElCs4K75TYBAffy"
}
Response{
"current_method":"FINGER:1",
  "chains":[
  {
    "name":"Fingerprint",
    "methods":["FINGER:1"],
    "image_name":"default",
    "apply_for_ep_owner":false,
    "position": 0,
    "is_enabled":true,
    "is_trusted":true,
    "short_name": ""
  }],
  "completed_methods":[],
  "msg": "Process started",
  "logon_process_id":"wZFkJ6kTvrbohSH6o5X5swGvZXDMoCQW",
  "plugins":[],
  "event_data_id":"OSLogon",
  "event_name":"NAM",
  "status":"MORE_DATA",
  "reason":"PROCESS_STARTED"
}
```

HTTP POST

https://authserver.example.com/api/v1/logon/wZFkJ6kTvrbohSH6o5X5swGvZXDMoCQW/do_logon

```
Request
{
  "response":
```

```
  {
    "capture":
    {
      "ISO":"Rk...Y=",
      "captureStatus":"Ok"
    }
  },
  "endpoint_session_id":"7XrFFQB28O6pod8Y5ElCs4K75TYBAffy"
}
```

Response

```
{
  "event_data_id":"OSLogon",
  "user_mobile_phone":"+16086783619",
  "msg":"Welcome!",
  "user_dn":"CN=JSmith,CN=Users,DC=COMPANY,DC=com",
  "data_id":"OSLogon",
  "user_email":"jsmith@COMPANY.com",
  "repo_id":"d3fba00652d211e5a19a000c2951aca4",
  "chains":[
  {
    "name":"Fingerprint",
    "methods":["FINGER:1"],
    "image_name":"default",
    "apply_for_ep_owner":false,
    "position": 0,
    "is_enabled":true,
    "is_trusted":true,
    "short_name": ""
  }],
"user_sid":"S-1-5-21-4279545561-3293806183-1755797738-500",
"user_cn":"JSmith",
  "completed_methods":["FINGER:1"],
"user_sid_hex":"010500000000000515000000d9ae14ff677e53c4ea58a768f40100",
  "logon_process_id":"wZFkJ6kTvrbohSH6o5X5swGvZXDMoCQW",
"current_method":"RADIUS:1",
  "user_name":"COMPANY\\JSmith",
"repo_obj_id":"2d3c89ccb3ea7b4dacbdfda13e26f450",
  "login_session_id":"LN8WNzoregyveYy69igAAKA77p3GC0RB",
"event_name":"NAM",
  "status":"OK",
"user_name_netbios":"COMPANY\\JSmith",
  "plugins":["LdapRules"],
"user_id":"ab2b845652d311e5a19a000c2951aca4",
  "reason": "CHAIN_COMPLETED"
}
```

Enrollment

For the enrollment, the fingerprint authentication method you should start an enrollment process and provide the enrollment data as a JSON container. The enrollment data has same structure and names as authentication data

Example

On the following example, the user JSmith from the COMPANY repository will enroll the fingerprint authentication method, user has a login session with identifier the “wQabBzcnJTBqOqTdClHJTtrkpHFUzg40”.

```
HTTP POST

https://authserver.example.com/api/v1/logon

Request

{
  "method_id":"FINGER:1",
"user_name":"COMPANY\\JSmith",
  "event":"NAM",
"endpoint_session_id":"7XrFFQB28O6pod8Y5ElCs4K75TYBAffy"
}

Response
{
"current_method":"FINGER:1",
  "chains":[
{
    "name":"Fingerprint",
"methods":["FINGER:1"],
    "image_name":"default",
"apply_for_ep_owner":false,
    "position": 0,
"is_enabled":true,
    "is_trusted":true,
"short_name": ""
  }],
"completed_methods":[],
  "msg": "Process started",
"logon_process_id":"wZFkJ6kTvrbohSH6o5X5swGvZXDMoCQW",
  "plugins":[],
"event_data_id":"OSLogon",
  "event_name":"NAM",
"status":"MORE_DATA",
  "reason":"PROCESS_STARTED"
}
```

HTTP POST

https://authserver.example.com/api/v1/logon/wZFkJ6kTvrbohSH6o5X5swGvZXDMoCQW/do_logon

Request

```
{
"response":
  {
"capture":
    {
"ISO":"Rk...Y=",
      "captureStatus":"Ok"
}
  },
"endpoint_session_id":"7XrFFQB28O6pod8Y5ElCs4K75TYBAffy"
}
```

Response

```
{
"event_data_id":"OSLogon",
  "user_mobile_phone":"+16086783619",
"msg":"Welcome!",
  "user_dn":"CN=JSmith,CN=Users,DC=company,DC=com",
"data_id":"OSLogon",
  "user_email":"jsmith@company.com",
"repo_id":"d3fba00652d211e5a19a000c2951aca4",
  "chains":[
{
    "name":"Fingerprint",
"methods":["FINGER:1"],
    "image_name":"default",
"apply_for_ep_owner":false,
    "position": 0,
"is_enabled":true,
    "is_trusted":true,
"short_name": ""
  }],
"user_sid":"S-1-5-21-4279545561-3293806183-1755797738-500",
  "user_cn":"JSmith",
"completed_methods":["FINGER:1"],
  "user_sid_hex":"010500000000000515000000d9ae14ff677e53c4ea58a768f40100",
"logon_process_id":"wZFkJ6kTvrbohSH6o5X5swGvZXDMoCQW",
  "current_method":"RADIUS:1",
"user_name":"COMPANY\\JSmith",
  "repo_obj_id":"2d3c89ccb3ea7b4dacbdfda13e26f450",
"login_session_id":"LN8WNzoregyveYy69igAAKA77p3GC0RB",
  "event_name":"NAM",
"status":"OK",
  "user_name_netbios":"COMPANY\\JSmith",
"plugins":["LdapRules"],
  "user_id":"ab2b845652d311e5a19a000c2951aca4",
"reason": "CHAIN_COMPLETED"
}
```

3.10.6 HOTP Authentication Method

The OATH HOTP authentication method provide user authentication by counter based one-time password. This method can be used as a single method in the authentication chain or as a part of the authentication chain..

Authentication

For the authentication, you should create a logon process with the HOTP authentication method or continue with the HOTP authentication method as next authentication method in case, when the chain has more than one methods. You should provide a JSON object, which contain user’s counter based one-time password, by POST request.

Table 3-72 The authentication data for the HOTP authentication method

Parameter name	Description
answer	This parameter contain user’s counter based one-time password

Resource will return a JSON-object with information about current state of the authentication.

The HOTP authentication method supports the set of authentication reasons.

Table 3-73 The HOTP method’s authentication reasons.

Reason value	Description
HOTP_PASSWORD_WRONG	The counter based one-time password providing on authentication was wrong
HOTP_PASSWORD_UNDEFINED	The counter based one-time password is undefined for user.

Example

On the following example, we provide HOTP authentication for the user JSmith from the COMPANY repository, user already create endpoint session with the identifier “5IUw6BzZEZ30d5AurQJbp7R9KLWhcHoC”.

```
HTTP POST
https://authserver.example.com/api/v1/logon
Request
{
"method_id":"HOTP:1",
"user_name":"COMPANY\\JSmith",
"event":"NAM",
"endpoint_session_id":"5IUw6BzZEZ30d5AurQJbp7R9KLWhcHoC"
}
Response
{
"event_name":"NAM",
"completed_methods":[],
"reason":"PROCESS_STARTED",
"event_data_id":"OSLogon",
"msg":"Process started",
"logon_process_id":"xYKEgMYGntELKvbD3KvfmNfvI8EljNLR",
"plugins": [],
"status": "MORE_DATA",
"current_method":"HOTP:1",
"chains":[
{
"name":"Counter based one time password",
"apply_for_ep_owner":false,
"is_enabled":true,
"is_trusted":true,
"short_name":"",
"image_name":"default",
"position":0,
"methods": ["HOTP:1"]
}]
}
HTTP POST
https://authserver.example.com/api/v1/logon/xYKEgMYGntELKvbD3KvfmNfvI8EljNLR/
do_logon
Request
{
"response":
{
"answer":"573854"
},
"endpoint_session_id":"5IUw6BzZEZ30d5AurQJbp7R9KLWhcHoC"
}
Response
{
"reason":"CHAIN_COMPLETED",
"msg": "Welcome!",
"event_name":"NAM",
"repo_obj_id":"2d3c89ccb3ea7b4dacbdfda13e26f450",
"user_sid":"S-1-5-21-4279545561-3293806183-1755797738-500",
"user_name":"COMPANY\\JSmith",
"user_mobile_phone":"+16086783619",
"user_dn":"CN=JSmith,CN=Users,DC=company,DC=com",
"event_data_id":"OSLogon",
"user_name_netbios":"COMPANY\\Jsmith",
"repo_id":"d3fba00652d211e5a19a000c2951aca4",
"logon_process_id":"xYKEgMYGntELKvbD3KvfmNfvI8EljNLR",
"status":"OK",
"current_method":"HOTP:1",
"chains":[
{
"name":"Counter based one time password",
"apply_for_ep_owner":false,
"is_enabled":true,
"is_trusted":true,
"short_name":"",
"image_name":"default",
"position":0,
"methods": ["HOTP:1"]
}],
"user_sid_hex":"010500000000000515000000d9ae14ff677e53c4ea58a768f40100",
"user_email":"jsmith@company.com",
"completed_methods":["HOTP:1"],
"login_session_id":"1yljtKNxC37kZcMDTND7PrDPDjJ6Kv4D",
"user_cn":"JSmith",
"plugins":["LdapRules"],
"data_id":"OSLogon",
"user_id":"ab2b845652d311e5a19a000c2951aca4"
}
```

Enrollment

For the enrollment, the counter based one-time password authentication method you should start an enrollment process and provide the enrollment data as JSON container. You can enroll method with a counter or with a three consecutive generated passwords for case when user does not know a counter value.

Table 3-74 *The HOTP method enrollment's data.*

Parameter name	Description
secret	The secret for password generation, it should be a hex string with length more than 6 characters.
counter	The counter for password generation. This is optional parameter with default value 1.
otp_format	The password format from supporting password format list, this is optional parameter
hash	The name of hashing algorithm, this is optional parameter. List of the hashing algorithm provided in Python library hashlib.algorithms_guaranteed
token_public_id	The Yubikey token public identifier, this is optional parameter. You should use it when you work with Yubikey hardware tokens.
hotp1	The 1st counter based one-time password, for enrollment without counter value
hotp2	The 2nd counter based one-time password, for enrollment without counter value
hotp3	The 3rd counter based one-time password, for enrollment without counter value

Table 3-75 *The HOTP one-time password's formats.*

One-time password format	Description
dec4	4 decimal digits
dec6	6 decimal digits
dec7	7 decimal digits
dec8	8 decimal digits

Resource will return a JSON object with status of the enrollment process.

The HOTP authentication method supports this list of the enrollment reasons.

Table 3-76 *The HOTP method's enrollment reasons.*

Enrollment reason value	Description
CANT_FIND_COUNTER	Method could not find a counter from the provided passwords list.

Example

On the following example, the user JSmith from the COMPANY repository will try to enroll the HOTP method, user already have a login session with the identifier "sz1bs8hCMNr2JOfDZJIhsCbAYwM2lHzN".

```
HTTP POST
https://authserver.example.com/api/v1/enroll
Request
{
"method_id":"HOTP:1",
"login_session_id":"sz1bs8hCMNr2JOfDZJIhsCbAYwM2lHzN"
}
Response
{
"enroll_process_id":"bweoSmHkB3FP17IOES4GtlReCmBxAEop"
}
HTTP POST
https://authserver.example.com/api/v1/enroll/bweoSmHkB3FP17IOES4GtlReCmBxAEop/
do_enroll
Request
{
"response":
{
"secret":"12345678901234567890",
"counter":"0"
},
"login_session_id":"sz1bs8hCMNr2JOfDZJIhsCbAYwM2lHzN"
}
Response
{
"status":"OK",
"method_id":"HOTP:1",
"msg": "",
"reason": ""
}
```

3.10.7 LDAP Password Authentication Method

The LDAP password authentication method provide user authentication by LDAP password from internal or external user’s repository. This method could be used as a single method in the authentication chain or as a part of the authentication chain.

Authentication

For the authentication, you should create a logon process with the LDAP password authentication method or continue with the LDAP password as next authentication method in case, when the chain has more than one method. You should provide a JSON object, which contain user’s password, by POST request.

Table 3-77 The authentication data for the LDAP password authentication method.

Parameter name	Description
answer	This parameter contain user’s LDAP password

Resource will return JSON-object with information about current state of the authentication.

LDAP password authentication method supports list of authentication reasons.

Table 3-78 The LDAP method’s authentication reasons.

Reason value	Description
LDAP_PASSWORD_UNDEFINED	The LDAP password undefined for user.
LDAP_PASSWORD_WRONG	LDAP password provided on authentication was wrong
LDAP_PASSWORD_ACCOUNT_RESTRICTION	LDAP account has restrictions
LDAP_PASSWORD_INVALID_LOGON_HOURS	User used invalid logon hours
LDAP_PASSWORD_INVALID_WORKSTATION	User used for logon invalid workstation
LDAP_PASSWORD_EXPIRED	LDAP password expired
LDAP_PASSWORD_ACCOUNT_DISABLED	LDAP account disabled
LDAP_PASSWORD_TOO_MANY_CONTEXT_IDS	Appliance has too many IDS context
LDAP_PASSWORD_ACCOUNT_EXPIRED	LDAP account expired
LDAP_PASSWORD_MUST_CHANGE	User must change password
LDAP_PASSWORD_ACCOUNT_LOCKED_OUT	LDAP account locked out

Example

On the following example, we provide LDAP password authentication for the user JSmith from the COMPANY repository, user already create endpoint session with the identifier “46kGFB3MUUebkcqosO9t4pVAVURsCMyz”.

```
HTTP POST
https://authserver.example.com/api/v1/logon
Request
{
"method_id":"LDAP_PASSWORD:1",
"user_name":"COMPANY\\JSmith",
"event":"NAM",
"endpoint_session_id":"46kGFB3MUUebkcqosO9t4pVAVURsCMyz"
}
Response
{
"plugins": [],
"event_name":"NAM",
"msg":"Process started",
"status":"MORE_DATA",
"reason":"PROCESS_STARTED",
"current_method":"LDAP_PASSWORD:1",
"completed_methods":[],
"chains":[
{
"is_enabled":true,
"apply_for_ep_owner":false,
"is_trusted":null,
"image_name":"default",
"name":"LDAP password",
"position":0,
"methods":["LDAP_PASSWORD:1"],
"short_name": ""
}],
"logon_process_id":"zGjcx3Kbh3scIbTXXCefo1lR86BZ5V0k",
"event_data_id":"OSLogon"
}
HTTP POST
https://authserver.example.com/api/v1/logon/zGjcx3Kbh3scIbTXXCefo1lR86BZ5V0k/
do_logon
Request
{
"response":
{
"answer":"123"
},
"endpoint_session_id":"46kGFB3MUUebkcqosO9t4pVAVURsCMyz"
}
Response
{
"msg":"Welcome!",
"user_mobile_phone":"+16086783619",
"event_name":"NAM",
"repo_id":"d3fba00652d211e5a19a000c2951aca4",
"user_email":"jsmith@company.com",
"user_name": "COMPANY\\JSmith",
"user_dn":"CN=JSmith,CN=Users,DC=company,DC=com",
"completed_methods":["LDAP_PASSWORD:1"],
"chains":[
{
"is_enabled":true,
"apply_for_ep_owner":false,
"is_trusted":null,
"image_name":"default",
"name":"LDAP password",
"position":0,
"methods":["LDAP_PASSWORD:1"],
"short_name": ""
}],
"user_sid":"S-1-5-21-4279545561-3293806183-1755797738-500",
"user_name_netbios":"COMPANY\\JSmith",
"user_cn":"JSmith",
"user_id":"ab2b845652d311e5a19a000c2951aca4",
"login_session_id":"BF7xjPVyO9wSqFj9UY0II1qzsemfcVqD",
"plugins":["LdapRules"],
"data_id":"OSLogon",
"status":"OK",
"reason":"CHAIN_COMPLETED",
"current_method":"LDAP_PASSWORD:1",
"user_sid_hex":"010500000000000515000000d9ae14ff677e53c4ea58a768f40100",
"logon_process_id":"zGjcx3Kbh3scIbTXXCefo1lR86BZ5V0k",
"event_data_id":"OSLogon",
"repo_obj_id":"2d3c89ccb3ea7b4dacbdfda13e26f450"
}
```

Enrollment

The LDAP password authentication method does not support enrollment – all user’s password provided by LDAP server.

3.10.8 Password Authentication Method

The password authentication method provide authentication by password for users, this password is not LDAP user’s password, and it is a virtual password stored in the appliance, which user can enroll. The password authentication method could be used as a single method in the authentication chain or as a part of the authentication chain.

Authentication

For the authentication, you should create a logon process with the password authentication method or continue with the password authentication method in case, when the authentication chain has more than one authentication methods. You should provide a JSON container with user’s password by POST request.

Table 3-79 *The authentication data for the password authentication method.*

Parameter name	Description
answer	This parameter contain user’s password

Resource will return a JSON-object with information about current state of the authentication.

The password authentication method supports the list of the authentication reasons.

Table 3-80 The password method’s authentication reasons

Reason value	Description
PASSWORD_UNDEFINED	The password undefined for user
PASSWORD_EXPIRED	The password was expired
PASSWORD_WRONG	The password provided at authentication was wrong

Example

On the following example, the user JSmith from the COMPANY repository will try to authenticate by the password method, user has an endpoint session with the identifier “TksW8P1T8nTee3LX2ZgAUCHEUPVCKOYC”.

```
POST
https://authserver.example.com/api/v1/logon
Request
{
"method_id":"PASSWORD:1",
"user_name":"COMPANY\\JSmith",
"event":"NAM",
"endpoint_session_id":"TksW8P1T8nTee3LX2ZgAUCHEUPVCKOYC"
}
Response
{
"reason":"PROCESS_STARTED",
"current_method":"PASSWORD:1",
"msg":"Process started",
"chains":[
{
"is_trusted":null,
"is_enabled":true,
"short_name":"",
"position":0,
"methods":["PASSWORD:1"],
"name":"Password",
"apply_for_ep_owner":false,
"image_name":"default"
}],
"event_data_id":"OSLogon",
"status":"MORE_DATA",
"plugins":[],
"logon_process_id":"jzZ5qMd9b4drh5chgcLk1KgHxtHh67Yo",
"completed_methods":[],
"event_name": "NAM"
}
HTTP POST
https://authserver.example.com/api/v1/logon/jzZ5qMd9b4drh5chgcLk1KgHxtHh67Yo/
do_logon
Request
{
"response":
{
"answer":"P@$sw0rd"
},
"endpoint_session_id":"TksW8P1T8nTee3LX2ZgAUCHEUPVCKOYC"
}
Response
```

```
{
"reason":"CHAIN_COMPLETED",
"user_id":"ab2b845652d311e5a19a000c2951aca4",
"event_name":"NAM",
"chains":[
{
"is_trusted":null,
"is_enabled":true,
"short_name":"",
"position":0,
"methods":["PASSWORD:1"],
"name":"Password",
"apply_for_ep_owner":false,
"image_name":"default"
}],
"data_id":"OSLogon",
"repo_obj_id":"2d3c89ccb3ea7b4dacbdfda13e26f450",
"msg":"Welcome (CHAP)!",
"plugins":["LdapRules"],
"user_name":"COMPANY\\JSmith",
"user_email":"jsmith@company.com",
"current_method":"PASSWORD:1",
"user_sid":"S-1-5-21-4279545561-3293806183-1755797738-500",
"status":"OK",
"logon_process_id":"jzZ5qMd9b4drh5chgcLk1KgHxtHh67Yo",
"user_dn":"CN=JSmith,CN=Users,DC=company,DC=com",
"user_sid_hex":"010500000000000515000000d9ae14ff677e53c4ea58a768f40100",
"login_session_id":"qgsrzukT8D2MjRq9exBcys0OaQnJcBUt",
"user_mobile_phone":"+16086783619",
"event_data_id":"OSLogon",
"completed_methods":["PASSWORD:1"],
"user_name_netbios":"COMPANY\\JSmith",
"repo_id":"d3fba00652d211e5a19a000c2951aca4",
"user_cn": "JSmith"
}
```

Enrollment

For the enrollment, the password authentication method you should start enrollment process and provide an enrollment data as a JSON container.

Table 3-81 The enrollment data for the password authentication method

Parameter name	Description
password	The new password value
confirmation	The confirmation of the new password value, if password will not equals to the confirmation – enrollment will fail.

Resource will return a JSON object with status of the enrollment process.

The password authentication method supports this set of the enrollment reasons.

Table 3-82 The password method's enrollment reasons

Reason value	Description
PASSWORD_BAD_CONFIRMATION	The password and the confirmation is not equals
PASSWORD_EMPTY	The provided password or confirmation was empty
PASSWORD_UNCHANGED	The provided password is equals to the current password
PASSWORD_TOO_SHORT	The provided password is too small, please check password length in the policy values
PASSWORD_TOO_SIMPLE	The provided password is too simple, user should use a stronger password

Example

On the following example, the user JSmith from the COMPANY repository will enroll the password authentication method, user has a login session with the identifier "QI98qz5PJazhpcFgI6HRbLeIGVSFOAOt".

```
HTTP POST
https://authserver.example.com/api/v1/enroll
Request
{
"method_id":"PASSWORD:1",
"login_session_id":"QI98qz5PJazhpcFgI6HRbLeIGVSFOAOt"
}
Response
{
"enroll_process_id":"SsKV3uOyJDskhqI6nUuqW0XYo4dJCtcP"
}
HTTP POST
https://authserver.example.com/api/v1/enroll/SsKV3uOyJDskhqI6nUuqW0XYo4dJCtcP/
do_enroll
Request
{
"response":
{
"password":"new_P@$sw0rd",
"confirmation":"new_P@$sw0rd"
},
"login_session_id":"QI98qz5PJazhpcFgI6HRbLeIGVSFOAOt"
}
Response
{
"reason":"",
"msg":"",
"status":"OK",
"method_id":"PASSWORD:1"
}
```

3.10.9 RADIUS Authentication Method

The RADIUS authentication method provide users' authentication by password on external RADIUS server. The RADIUS authentication method could be used as a single method in the authentication chain or as a part of the authentication chain.

Authentication

For the authentication, you should create a logon process with the RADIUS authentication method or continue with the RADIUS authentication method in case, when the authentication chain has more than one authentication methods. You should provide a JSON container with the user's RADIUS password by POST request.

Table 3-83 *The authentication data for the RADIUS authentication method*

Parameter name	Description
answer	This parameter contain user's RADIUS password

Resource will return a JSON-object with information about current state of the authentication.

The RADIUS authentication method supports the set of the authentication reasons.

Table 3-84 *The RADIUS method's authentication reasons*

Reason value	Description
RADIUS_WRONG_PASSWORD	The RADIUS password provided for the authentication was wrong.

Example

On the following example, the user JSmith from the COMPANY repository will try to authenticate with the RADIUS authentication method, user already has an endpoint session with the identifier "7XrFFQB28O6pod8Y5ElCs4K75TYBAffy".

```
HTTP POST
https://authserver.example.com/api/v1/logon
Request
{
"method_id":"RADIUS:1",
"user_name":"COMPANY\\JSmith",
"event":"NAM",
"endpoint_session_id":"7XrFFQB28O6pod8Y5ElCs4K75TYBAffy"
}
Response
{
"current_method":"RADIUS:1",
"chains":[
{
"name":"Radius Client",
"methods":["RADIUS:1"],
"image_name":"default",
"apply_for_ep_owner":false,
"position": 0,
"is_enabled":true,
"is_trusted":true,
"short_name": ""
}],
"completed_methods":[],
"msg": "Process started",
"logon_process_id":"wZFkJ6kTvrbohSH6o5X5swGvZXDMoCQW",
"plugins":[],
"event_data_id":"OSLogon",
"event_name":"NAM",
"status":"MORE_DATA",
"reason":"PROCESS_STARTED"
}
HTTP POST
https://authserver.example.com/api/v1/logon/wZFkJ6kTvrbohSH6o5X5swGvZXDMoCQW/
do_logon
Request
{
"response":
{
"answer":"Str0nG_P@$WorD"
},
"endpoint_session_id":"7XrFFQB28O6pod8Y5ElCs4K75TYBAffy"
}
Response
{
"event_data_id":"OSLogon",
"user_mobile_phone":"+16086783619",
"msg":"Welcome!",
"user_dn":"CN=JSmith,CN=Users,DC=company,DC=com",
"data_id":"OSLogon",
"user_email":"jsmith@company.com",
"repo_id":"d3fba00652d211e5a19a000c2951aca4",
"chains":[
{
"name":"Radius Client",
"methods":["RADIUS:1"],
"image_name":"default",
"apply_for_ep_owner":false,
"position": 0,
"is_enabled":true,
"is_trusted":true,
"short_name": ""
}],
"user_sid":"S-1-5-21-4279545561-3293806183-1755797738-500",
"user_cn":"JSmith",
"completed_methods":["RADIUS:1"],
"user_sid_hex":"010500000000000515000000d9ae14ff677e53c4ea58a768f40100",
"logon_process_id":"wZFkJ6kTvrbohSH6o5X5swGvZXDMoCQW",
"current_method":"RADIUS:1",
"user_name":"COMPANY\\JSmith",
"repo_obj_id":"2d3c89ccb3ea7b4dacbdfda13e26f450",
"login_session_id":"LN8WNzoregyveYy69igAAKA77p3GC0RB",
"event_name":"NAM",
"status":"OK",
"user_name_netbios":"COMPANY\\JSmith",
"plugins":["LdapRules"],
"user_id":"ab2b845652d311e5a19a000c2951aca4",
"reason": "CHAIN_COMPLETED"
}
```

Enrollment

For enrollment, RADIUS authentication method you should start an enrollment process and provide by POST request enrollment data as a JSON container.

Table 3-85 The enrollment data for the RADIUS authentication method

Parameter name	Description
user_name	The user name that will be used for RADIUS authentication. This is optional parameter
send_reponame	The boolean flag – send or not repository name with user name, if it true – result will be “repo\username”, if false – “username”. Optional parameter

Resource will return a JSON object with status of the enrollment process.

Example

On the following example, the user JSmith from the COMPANY repository will enroll the RADIUS authentication method, user has a login session with the identifier “HJ6OZ6wBtfGAyEWTlO1BeGJOSOU5zwi3”.

```
HTTP POST
https://authserver.example.com/api/v1/enroll
Request
{
"method_id":"RADIUS:1",
"login_session_id":"HJ6OZ6wBtfGAyEWTlO1BeGJOSOU5zwi3"
}
Response
{
"enroll_process_id":"FIOgJueuovKsGYrgHB6AKWO7cWXHFd6u"
}
HTTP POST
https://authserver.example.com/api/v1/enroll/FIOgJueuovKsGYrgHB6AKWO7cWXHFd6u/
do_enroll
Request
{
"response":
{
"user_name":"johnsmith"
},
"login_session_id":"HJ6OZ6wBtfGAyEWTlO1BeGJOSOU5zwi3"
}
Response
{
"method_id":"RADIUS:1",
"status":"OK",
"msg": "",
"reason": ""
}
```

3.10.10 Security Questions Authentication Method

The security questions authentication method provide users’ authentication by set of answers for a question. The security questions authentication method could be used as a single method in the authentication chain or as a part of the authentication chain.

Authentication

For the authentication, you should create a logon process with the security questions authentication method or continue with the security questions authentication method in case, when the authentication chain has more than one authentication methods. You should provide a JSON container with the user's answers by POST request. Method has two step for the authentication, on first step you should make an empty request for getting list of the security question from the appliance, on second step you should provide the user's answers. The answer is a JSON-container with security question identifier and user's answer.

***Table 3-86** The authentication data for the security question authentication method*

Parameter name	Description
answer	This parameter contain the set of user's answers to the security questions. Each answer is a JSON object with "question identifier" and "user's answer".

Resource will return a JSON-object with information about current state of the authentication.

The security questions authentication method supports the list of the authentication reasons.

***Table 3-87** The security question method's authentication reasons*

Reason value	Description
SECQUEST_WRONG_ANSWERS	The answers provided at the authentication was wrong
SECQUEST_WAITING_ANSWERS	The security questions authentication method wait for user's answers.

Example

On the following example, the user JSmith form the COMPANY repository will try to authenticate by the security question authentication method, user already has an endpoint session with the identifier "nVVcBJ8GacuBfT7ur3tUKG2k54I6016T".

```
HTTP POST
https://authserver.example.com/api/v1/logon
Request
{
"method_id":"SECQUEST:1",
"user_name":"COMPANY\\JSmith",
"event":"NAM",
"endpoint_session_id":"nVVcBJ8GacuBfT7ur3tUKG2k54I6016T"
}
Response
{
"current_method":"SECQUEST:1",
"chains":[
{
"name":"Security Questions",
"methods":["SECQUEST:1"],
"image_name":"default",
"apply_for_ep_owner":false,
"position":0,
"is_enabled":true,
"is_trusted":true,
"short_name":""
}],
"completed_methods":[],
"msg":"Process started",
"logon_process_id":"rqakRSR1pTxLsfxQy6UQRGXZBF4jLZC3",
"plugins":[],
"event_data_id":"OSLogon",
"event_name": "NAM",
"status":"MORE_DATA",
"reason":"PROCESS_STARTED"
}
HTTP POST
https://authserver.example.com/api/v1/logon/rqakRSR1pTxLsfxQy6UQRGXZBF4jLZC3/
do_logon
Request
{
"response":{},
"endpoint_session_id":"nVVcBJ8GacuBfT7ur3tUKG2k54I6016T"
}
Response
{
"current_method":"SECQUEST:1",
"event_data_id":"OSLogon",
"msg":"Waiting for answers...",
"event_name":"NAM",
"completed_methods":[],
"chains":[
{
"name":"Security Questions",
"methods":["SECQUEST:1"],
"image_name":"default",
"apply_for_ep_owner":false,
"position":0,
"is_enabled":true,
"is_trusted":true,
"short_name":""
}],
"questions":
{
"0":"What is your dog name?",
"1":"What is your favorite band name?"
},
"status":"MORE_DATA",
"logon_process_id":"rqakRSR1pTxLsfxQy6UQRGXZBF4jLZC3",
"plugins":[],
"method_id":"SECQUEST:1",
"reason":"SECQUEST_WAITING_ANSWERS"
}
HTTP POST
https://authserver.example.com/api/v1/logon/rqakRSR1pTxLsfxQy6UQRGXZBF4jLZC3/
do_logon
Request
{
"response":
{
"answers":
{
"0":"Spotty",
"1":"The Beatles"
}
```

```
},
"endpoint_session_id":"nVVcBJ8GacuBfT7ur3tUKG2k54I6016T"
}
Response
{
"event_data_id":"OSLogon",
"user_mobile_phone":"+16086783619",
"msg":"Welcome!",
"user_dn":"CN=JSmith,CN=Users,DC=company,DC=com",
"data_id":"OSLogon",
"user_email":"jsmith@company.com",
"repo_id":"d3fba00652d211e5a19a000c2951aca4",
"chains":[
{
"name":"Security Questions",
"methods":["SECQUEST:1"],
"image_name":"default",
"apply_for_ep_owner":false,
"position":0,
"is_enabled":true,
"is_trusted":true,
"short_name":""
}],
"user_sid":"S-1-5-21-4279545561-3293806183-1755797738-500",
"user_cn":"JSmith",
"completed_methods":["SECQUEST:1"],
"user_sid_hex":"0105000000000515000000d9ae14ff677e53c4ea58a768f40100",
"logon_process_id":"rqakRSR1pTxLsfxQy6UQRGXZBF4jLZC3",
"current_method":"SECQUEST:1",
"user_name":"COMPANY\\JSmith",
"repo_obj_id":"2d3c89ccb3ea7b4dacbdfda13e26f450",
"login_session_id":"C0rAfRFyPCALSK36eSXf56q7rQgmhle6",
"event_name":"NAM",
"status":"OK",
"user_name_netbios":"COMPANY\\JSmith",
"plugins":["LdapRules"],
"user_id":"ab2b845652d311e5a19a000c2951aca4",
"reason":"CHAIN_COMPLETED"
}
```

Enrollment

For the enrollment , the security questions authentication method you should start an enrollment process and provide an enrollment data as a JSON container. The security question authentication method has two step on enrollment process: on first step you should make an empty response for getting set of the security questions, on second step you should provide the user’s answers.

Table 3-88 *The enrollment data for the security question authentication method*

Parameter name	Description
answers	The set of the answers for the security questions, this is a JSON object with question identifier as key and answer as the value.

Resource will return a JSON object with status of the enrollment process.

The security questions authentication method supports this set of the enrollment reasons.

Reason value	Description
SECQUEST_WAITING_ANSWERS	The security question authentication method waiting for user’s answers set for enrollment.

Example

On the following example, the user JSmith from the COMPANY repository will try to enroll the security questions authentication method, user has a login session with the identifier “Lfz8BRaSBoGuYI8sVbCBp4bZEtEXL04h”.

```
HTTP POST
https://authserver.example.com/api/v1/enroll
Request
{
"method_id":"SECQUEST:1",
"login_session_id":"Lfz8BRaSBoGuYI8sVbCBp4bZEtEXL04h"
}
Response
{
"enroll_process_id":"DVqM8pH4FC8cBgUG6wKvMlljU6Lhpi6d"
}
HTTP POST
https://authserver.example.com/api/v1/enroll/DVqM8pH4FC8cBgUG6wKvMlljU6Lhpi6d/
do_enroll
Request
{
"response":{},
"login_session_id":"Lfz8BRaSBoGuYI8sVbCBp4bZEtEXL04h"
}
Response
{
"method_id":"SECQUEST:1",
"questions":
{
"0":"What is your dog name?",
"1":"What is your favorite band name?"
},
"status":"MORE_DATA",
"msg":"Waiting for answers...",
"reason":"SECQUEST_WAITING_ANSWERS"
}
HTTP POST
https://authserver.example.com/api/v1/enroll/DVqM8pH4FC8cBgUG6wKvMlljU6Lhpi6d/
do_enroll
Request
{
"response":
{
"answers":
```

```
{
"0":"Spotty",
"1":"The Beatles"
}
},
"login_session_id":"Lfz8BRaSBoGuYI8sVbCBp4bZEtEXL04h"
}
Response
{
"method_id":"SECQUEST:1",
"status":"OK",
"msg":"",
"reason":""
}
```

3.10.11 Smartphone Authentication Method

The smartphone authentication method provide users' authentication by their smartphone with special application (for iOS, Android). This method has two ways for the authentication: online and offline. For the online authentication users authenticate by application – accept or reject the authentication for their smartphone application. For the offline authentication users authenticate by application generated one-time password, this way provide authentication for cases without connection to the appliance. The smartphone authentication method could be used as a single method in the authentication chain or as a part of the authentication chain.

Authentication

For the authentication, you should create a logon process with the smartphone authentication method or continue with the smartphone authentication method in case, when the authentication chain has more than one authentication methods. You should provide an empty response for starting and checking status of the online authentication, or provide a JSON container with application generated one-time password for the offline authentication.

Table 3-89 *The authentication data for the smartphone authentication method*

Parameter name	Description
otp	The application generated one-time password for the offline authentication by smartphone authentication method.

Resource will return a JSON-object with information about current state of the authentication.

The smartphone authentication method supports the list of the authentication reasons.

Table 3-90 The smartphone method's authentication reasons.

Reason value	Reason description
SMARTPHONE_SAME_TOTP	The application generated one-time password provided for the authentication was same that last time
SMARTPHONE_WRONG_TOTP	The application generated one-time password provided for the authentication was wrong.
SMARTPHONE_AUTH_CONFIRM_TIMEOUT	The time for the authentication confirmation was out
SMARTPHONE_AUTH_REJECTED	The authentication was rejected from the smartphone application
SMARTPHONE_LOGON_IN_PROGRESS	The authentication by the smartphone authentication method in process
SMARTPHONE_WAITING_DATA	The smartphone authentication method wait for the authentication data from the user.

Example

On the following example, the user JSmith from the COMPANY repository will try to authenticate by the smartphone authentication method, he will use his smartphone with installed authentication application. User already has an endpoint session with the identifier “JTx5s5DW4HDCLvUfRG8W426etdgQ34uu”.

```
HTTP POST
https://authserver.example.com/api/v1/logon
Request
{
"method_id":"SMARTPHONE:1",
"user_name":"COMPANY\\JSmith",
"event":"NAM",
"endpoint_session_id":"JTx5s5DW4HDCLvUfRG8W426etdgQ34uu"
}
Response
{
"event_data_id":"OSLogon",
"current_method":"SMARTPHONE:1",
"completed_methods":[],
"plugins":[],
"status":"MORE_DATA",
"logon_process_id":"FSo7XFgzZU2X7186rQyUgac8KEajSwHP",
"chains":[
{
"image_name":"default",
"position":0,
"name":"Smartphone",
"short_name":"",
"is_enabled":true,
"apply_for_ep_owner":false,
"is_trusted":true,
"methods":["SMARTPHONE:1"]
}],
"reason":"PROCESS_STARTED",
"event_name":"NAM",
"msg":"Process started"
}
```

With next request, the appliance will send the authentication request to the user's smartphone.

HTTP POST

https://authserver.example.com/api/v1/logon/FSo7XFgzZU2X7186rQyUgac8KEajSwHP/do_logon

Request

```
{
"response":{},
"endpoint_session_id":"JTx5s5DW4HDCLvUfRG8W426etdgQ34uu"
}
```

Response

```
{
"event_data_id":"OSLogon",
"current_method":"SMARTPHONE:1",
"completed_methods":[],
"plugins":[],
"status":"MORE_DATA",
"logon_process_id":"FSo7XFgzZU2X7186rQyUgac8KEajSwHP",
"chains":[
{
"image_name":"default",
"position":0,
"name":"Smartphone",
"short_name":"",
"is_enabled":true,
"apply_for_ep_owner":false,
"is_trusted":true,
"methods":["SMARTPHONE:1"]
}],
"reason":"SMARTPHONE_WAITING_DATA",
"event_name":"NAM",
"method_id":"SMARTPHONE:1",
"msg":"Waiting for smartphone data..."
}
```

The user accepted the authentication request and status was changed.

HTTP POST

https://authserver.example.com/api/v1/logon/FSo7XFgzZU2X7186rQyUgac8KEajSwHP/do_logon

Request

```
{
"response":{},
"endpoint_session_id":"JTx5s5DW4HDCLvUfRG8W426etdgQ34uu"
}
```

Response

```
{
"user_name_netbios":"COMPANY\\JSmith",
"event_data_id":"OSLogon",
"login_session_id":"tbKNRLZnEPLeWRA53LRA3dPJ6fuYmT31",
"user_name":"COMPANY\\JSmith",
"repo_id":"d3fba00652d211e5a19a000c2951aca4",
"data_id":"OSLogon",
"user_cn":"JSmith",
"current_method":"SMARTPHONE:1",
"status":"OK",
"chains":[
{
"image_name":"default",
"position":0,
"name":"Smartphone",
"short_name":"",
"is_enabled":true,
"apply_for_ep_owner":false,
"is_trusted":true,
"methods":["SMARTPHONE:1"]
}],
"reason":"CHAIN_COMPLETED",
"user_sid":"S-1-5-21-4279545561-3293806183-1755797738-500",
"user_mobile_phone":"+16086783619",
"user_id":"ab2b845652d311e5a19a000c2951aca4",
"logon_process_id":"FSo7XFgzZU2X7186rQyUgac8KEajSwHP",
"event_name":"NAM",
"user_sid_hex":"010500000000000515000000d9ae14ff677e53c4ea58a768f40100",
"completed_methods":["SMARTPHONE:1"],
"plugins":["LdapRules"],
"repo_obj_id":"2d3c89ccb3ea7b4dacbdfda13e26f450",
"user_email":"jsmith@company.com",
"user_dn":"CN=JSmith,CN=Users,DC=company,DC=com",
"method_id":"SMARTPHONE:1",
"msg":"Auth accepted"
}
```

On the following example, the user JSmith from the COMPANY repository will try to authenticate by the smartphone authentication method with offline authentication and the application generated one-time password.

```
HTTP POST
https://authserver.example.com/api/v1/logon
Request
{
"method_id":"SMARTPHONE:1",
"user_name":"COMPANY\\JSmith",
"event":"NAM",
"endpoint_session_id":"JTx5s5DW4HDCLvUfRG8W426etdgQ34uu"
}
Response
{
"event_data_id":"OSLogon",
"current_method":"SMARTPHONE:1",
"completed_methods":[],
"plugins":[],
"status":"MORE_DATA",
"logon_process_id":"9JoA6EPM7AFAh3WoK6kO2TIGQXc308qM",
"chains":[
{
"image_name":"default",
"position":0,
"name":"Smartphone",
"short_name":"",
"is_enabled":true,
"apply_for_ep_owner":false,
"is_trusted":true,
"methods":["SMARTPHONE:1"]
}],
"reason":"PROCESS_STARTED",
"event_name":"NAM",
"msg":"Process started"
}
HTTP POST
https://authserver.example.com/api/v1/logon/9JoA6EPM7AFAh3WoK6kO2TIGQXc308qM/
do_logon
```

```
Request
{
"response":
{
"totp":"263965"
},
"endpoint_session_id":"JTx5s5DW4HDCLvUfRG8W426etdgQ34uu"
}
Response
{
"user_name_netbios":"COMPANY\\JSmith",
"event_data_id":"OSLogon",
"login_session_id":"xGTHxdOd4FbKT5EyWQvtA4CGHSfdXawf",
"user_name":"COMPANY\\JSmith",
"repo_id":"d3fba00652d211e5a19a000c2951aca4",
"data_id":"OSLogon",
"user_cn":"JSmith",
"current_method":"SMARTPHONE:1",
"status":"OK",
"chains":[
{
"image_name":"default",
"position":0,
"name":"Smartphone",
"short_name":"",
"is_enabled":true,
"apply_for_ep_owner":false,
"is_trusted":true,
"methods":["SMARTPHONE:1"]
}],
"reason":"CHAIN_COMPLETED",
"user_sid":"S-1-5-21-4279545561-3293806183-1755797738-500",
"user_mobile_phone":"+16086783619",
"user_id":"ab2b845652d311e5a19a000c2951aca4",
"logon_process_id":"9JoA6EPM7AFAh3WoK6kO2TIGQXc308qM",
"event_name":"NAM",
"user_sid_hex":"010500000000000515000000d9ae14ff677e53c4ea58a768f40100",
"completed_methods":["SMARTPHONE:1"],
"plugins":["LdapRules"],
"repo_obj_id":"2d3c89ccb3ea7b4dacbdfda13e26f450",
"user_email":"jsmith@company.com",
"user_dn":"CN=JSmith,CN=Users,DC=company,DC=com",
"method_id":"SMARTPHONE:1",
"msg":"Auth accepted"
}
```

Enrollment

For the enrollment, the smartphone authentication method you should start an enrollment process and provide an enrollment data by the smartphone application. For starting you should sent empty response by POST and send empty response for getting an enrollment status.

Resource will return a data for a QR code; you should create a QR code and scan it by the smartphone application. Users cannot start enrollment in offline mode.

Table 3-91 The enrollment response for the smartphone authentication method

Responce parameter name	Description
qrdata	The data for a QR code, you should user this parameter for the QR code generation and then you should show this QR code to the user.

Resource will return a JSON object with status of the enrollment process.

The smartphone authentication method supports this set of the enrollment reasons.

Table 3-92 The smartphone method's enrollment reasons

Reason value	Reason description
SMARTPHONE_ENROLL_TIMEOUT	The time for the enrollment was out.
SMARTPHONE_WAITING_DATA	The smartphone authentication method waiting enrollment data from the smartphone application.
SMARTPHONE_SCAN_QR	The smartphone authentication method provide data for the QR code.

Example

On the following example, the user JSmith from the COMPANY repository will try to enroll the smartphone authentication method by his smartphone application, user already has a login session with the identifier “4ZqmTy05KmwJ2ZbejUGUM5XypdcUpD4F”.

```
HTTP POST
https://authserver.example.com/api/v1/enroll
Request
{
"method_id":"SMARTPHONE:1",
"login_session_id":"4ZqmTy05KmwJ2ZbejUGUM5XypdcUpD4F"
}
Response
{
"enroll_process_id":"U8emvI9mnKXTbwCWBJQBzu6SQMbwx2X7"
}
HTTP POST
https://authserver.example.com/api/v1/enroll/U8emvI9mnKXTbwCWBJQBzu6SQMbwx2X7/
do_enroll
Request
{
"response":{},
"login_session_id":"4ZqmTy05KmwJ2ZbejUGUM5XypdcUpD4F"
}
Response
{
"qrdata":"OOBDATAqhNL0wMaG/H64GoEFqqd3eJWmWkNI6JrDh+vRuo/mDvHJc/
PhnpS4iOtqMz9OLG1ItO++ccCPciDOAO6Fhicvux9eWocZ91oI5W82yy+X3eht/
V1JxGM2neihZVuxAol4nr5XhDUBXmx9PtLoUKl+HSncX9YUWkF/MbEX3XTlbvRtiXs/
AFJTkZDkYLgm8mRM2I3z5sht+ToVM+SE8UFTInjZjqgg3MxTY3VqZOYR1Jsf/
iC6dr+NJY1NdjAsP2ErEw8uBfR2AOX/Q/
+oCbYFuxZKRr0+SM46eCQhtcmktyClKlyZtpbHResPNAtkaRl6SD2zHbUe48nk/R922Il+w==",
"msg":"Scan this QR with smartphone app",
"reason":"SMARTPHONE_SCAN_QR",
"method_id":"SMARTPHONE:1",
"status":"MORE_DATA"
}
On this step user does not scan QR code yet
HTTP POST
https://authserver.example.com/api/v1/enroll/U8emvI9mnKXTbwCWBJQBzu6SQMbwx2X/
do_enroll
Request
{
"response":{},
"login_session_id":"4ZqmTy05KmwJ2ZbejUGUM5XypdcUpD4F"
}
Response
{
"msg":"Waiting for smartphone data...",
"reason":"SMARTPHONE_WAITING_DATA",
"method_id":"SMARTPHONE:1", "status": "MORE_DATA"
}
User successfully scanned a QR code and enrolled method
HTTP POST
https://authserver.example.com/api/v1/enroll/U8emvI9mnKXTbwCWBJQBzu6SQMbwx2X/
do_enroll
Request
{
"response":{},
"login_session_id":"4ZqmTy05KmwJ2ZbejUGUM5XypdcUpD4F"
}
Response
{
"reason":"",
"msg":"",
"status":"OK",
"method_id":"SMARTPHONE:1"
}
```

3.10.12 SMS Authentication Method

The SMS one-time password authentication method provide authentication by one-time password that will send to the user’s phone in SMS. The appliance use a mobile phone numbers from LDAP repository, if user have not a mobile phone number in repository, he could not use this authentication method. The SMS one-time password authentication method could be used as a single method in the authentication chain or as a part of the authentication chain.

Authentication

For the authentication, you should create a logon process with the SMS one-time password authentication method or continue with the SMS one-time password authentication method in case, when the authentication chain has more than one authentication methods. You should provide a JSON container with the user’s one-time password by POST request. This method has two step for authentication, on first step you should send empty data for sending SMS with one-time password to an user’s phone, on second step you should provide an one-time password.

Table 3-93 *The authentication data for the SMS authentication method*

Parameter name	Description
answer	This parameter contain user’s one-time password

Resource will return a JSON-object with information about current state of the authentication.

The SMS one-time password authentication method supports the list of the authentication reasons.

Table 3-94 The SMS method’s authentication reasons.

Reason value	Description
OTP_CANNOT_SEND	The appliance cannot send by SMS one-time password
OTP_TOO_MANY_SENT	The appliance was sent too many one-time passwords
OTP_WAITING_PASSWORD	The authentication method waiting for one-time password
OTP_NO_PASSWORD	The password provided for the authentication was empty
OTP_PASSWORD_EXPIRED	The one-time password was expired
OTP_WRONG_PASSWORD	The one-time password was wrong
OTP_TOO_MANY_REQUESTS	The appliance got too many requests

Example

On the following example, the user JSmith from the COMPANY repository will try to authenticate by the SMS one-time password authentication method, user already has an endpoint session with the identifier “Uho2sBV9AgIBUObuxrZYvKxw7ZKs84fV”.

```
HTTP POST
https://authserver.example.com/api/v1/logon
Request
{
"method_id":" SMS_OTP:1",
"user_name":"COMPANY\\JSmith",
"event":"NAM",
"endpoint_session_id":"Uho2sBV9AgIBUObuxrZYvKxw7ZKs84fV"
}
Response
{
"reason":"PROCESS_STARTED",
"current_method":"SMS_OTP:1",
"msg":"Process started",
"chains":[
{
"is_trusted":true,
"is_enabled":true,
"short_name":"",
"position":0,
"methods":["SMS_OTP:1"],
"name":"SMS",
"apply_for_ep_owner":false,
"image_name":"default"
}],
"event_data_id":"OSLogon",
"status":"MORE_DATA",
"plugins":[],
"logon_process_id":"RnWlothKg8kI4zO6pwMhsnT10PZ6dpSh",
"completed_methods":[],
"event_name":"NAM"
}
```

```
HTTP POST
https://authserver.example.com/api/v1/logon/RnWlothKg8kI4zO6pwMhsnT10PZ6dpSh/
do_logon
Request
{
"response":{},
"endpoint_session_id":"Uho2sBV9AgIBUObuxrZYvKxw7ZKs84fV"
}
Response
{
"reason":"OTP_WAITING_PASSWORD",
"current_method":"SMS_OTP:1",
"msg":"OTP password sent, please enter",
"chains":[
{
"is_trusted":true,
"is_enabled":true,
"short_name":"",
"position":0,
"methods":["SMS_OTP:1"],
"name":"SMS",
"apply_for_ep_owner":false,
"image_name":"default"
}],
"event_data_id":"OSLogon",
"status":"MORE_DATA",
"plugins":[],
"logon_process_id":"RnWlothKg8kI4zO6pwMhsnT10PZ6dpSh",
"completed_methods":[],
"event_name": "NAM"
}
HTTP POST
https://authserver.example.com/api/v1/logon/RnWlothKg8kI4zO6pwMhsnT10PZ6dpSh/
do_logon
Request
{
"response":
{
"answer":"12345678"
},
"endpoint_session_id":"Uho2sBV9AgIBUObuxrZYvKxw7ZKs84fV"
}
Response
{
"reason":"CHAIN_COMPLETED",
"user_id":"ab2b845652d311e5a19a000c2951aca4",
"event_name":"NAM",
"chains":[
{
"is_trusted":true,
"is_enabled":true,
"short_name":"",
"position":0,
"methods":["SMS_OTP:1"],
"name":"SMS",
"apply_for_ep_owner":false,
"image_name":"default"
}],
"data_id":"OSLogon",
"repo_obj_id":"2d3c89ccb3ea7b4dacbdfda13e26f450",
"msg":"Welcome!",
"plugins":["LdapRules"],
"user_name":"COMPANY\\JSmith",
"user_email":"jsmith@company.com",
"current_method":"EMAIL_PASSWORD:1",
"user_sid":"S-1-5-21-4279545561-3293806183-1755797738-500",
"status":"OK",
"logon_process_id":"RnWlothKg8kI4zO6pwMhsnT10PZ6dpSh",
"user_dn":"CN=JSmith,CN=Users,DC=company,DC=com",
"user_sid_hex":"010500000000000515000000d9ae14ff677e53c4ea58a768f40100",
"login_session_id":"nNSn6EepNQBk9hxD5X7lY3H064flsP1K",
"user_mobile_phone":"+16086783619",
"event_data_id":"OSLogon",
"completed_methods":["SMS_PASSWORD:1"],
"user_name_netbios":"COMPANY\\JSmith",
"repo_id":"d3fba00652d211e5a19a000c2951aca4",
"user_cn":"JSmith"
}
```

Enrollment

The SMS one-time password authentication method could not be enrolled, this method based on user’s LDAP attribute and all users from repository with mobile phone could authenticate by this method.

3.10.13 TOTP Authentication Method

The OATH TOTP authentication method provide the users’ authentication by time based one-time password, this otp can be generated by any OATH TOTP compliant token. This method can be used as single method in an authentication chain or as a part of an authentication chain.

Authentication

For the authentication, you should create a logon process with the TOTP method or continue with the TOTP as next authentication method in case, when the authentication chain has more than one authentication method. You should provide a JSON container with the user’s time based one-time password by POST request.

Table 3-95 *The authentication data for the TOTP authentication method*

Parameter name	Description
answer	This parameter contain the user’s time based one-time password

Resource will return a JSON-object with information about current state of the authentication.

The TOTP authentication method supports the list of the authentication reasons.

Table 3-96 *The TOTP method's authentication reasons*

Reason value	Description
TOTP_PASSWORD_UNDEFINED	The time based one-time password undefined for user
TOTP_WAIT_MINUTE	The time based one-time password was already used, please wait one minute and try again
TOTP_PASSWORD_WRONG	The time based one-time password provided at authentication was wrong

Example

On the following example, we provide authentication by the TOTP authentication method for the user JSmith from the COMPANY repository, user already created an endpoint session with the identifier "uaoVUar5v9hL3aRFFpAXCmhNTnpQg8L5".

```
HTTP POST
https://authserver.example.com/api/v1/logon
Request
{
"method_id":"TOTP:1",
"user_name":"COMPANY\\JSmith",
"event":"NAM",
"endpoint_session_id":"uaoVUar5v9hL3aRFFpAXCmhNTnpQg8L5"
}
Response
{
"event_name":"NAM",
"completed_methods":[],
"reason":"PROCESS_STARTED",
"event_data_id":"OSLogon",
"msg":"Process started",
"logon_process_id": "5SkULTwh0CC6z4162wL7qGmBscJEUiM0", "plugins": [],
"status":"MORE_DATA",
"current_method":"TOTP:1",
"chains":[
{
"name":"Time based one time password",
"apply_for_ep_owner":false,
"is_enabled":true,
"is_trusted":true,
"short_name":"",
"image_name":"default",
"position":0,
"methods":["TOTP:1"]
}]
}
HTTP POST
https://authserver.example.com/api/v1/logon/5SkULTwh0CC6z4162wL7qGmBscJEUiM0/
do_logon
Request
{
"response":
{
"answer":"604630"
},
"endpoint_session_id":"uaoVUar5v9hL3aRFFpAXCmhNTnpQg8L5"
}
Response:
```

```
{
"reason":"CHAIN_COMPLETED",
"msg":"Welcome!",
"event_name":"NAM",
"repo_obj_id":"2d3c89ccb3ea7b4dacbdfda13e26f450",
"user_sid":"S-1-5-21-4279545561-3293806183-1755797738-500",
"user_name":"COMPANY\\JSmith",
"user_mobile_phone":"+16086783619",
"user_dn":"CN=JSmith,CN=Users,DC=company,DC=com",
"event_data_id":"OSLogon",
"user_name_netbios":"COMPANY\\JSmith",
"repo_id":"d3fba00652d211e5a19a000c2951aca4",
"logon_process_id":"5SkULTwh0CC6z4162wL7qGmBscJEUiM0",
"status":"OK",
"current_method":"TOTP:1",
"chains":[
{
"name":"Time based one time password",
"apply_for_ep_owner":false,
"is_enabled":true,
"is_trusted":true,
"short_name":"",
"image_name":"default",
"position":0,
"methods":["TOTP:1"]
}],
"user_sid_hex":"010500000000000515000000d9ae14ff677e53c4ea58a768f40100",
"user_email":"jsmith@company.com",
"completed_methods":["TOTP:1"],
"login_session_id":"U0ubr6cq1UIsTJrzjrLLftC9Czpghq83",
"user_cn":"JSmith",
"plugins":["LdapRules"],
"data_id":"OSLogon",
"user_id":"ab2b845652d311e5a19a000c2951aca4"
}
```

Enrollment

For the enrollment, the time based one-time password authentication method you should start enrollment process and provide enrollment data as a JSON container.

Table 3-97 The enrollment data for the TOTP authentication method

Parameter name	Description
secret	The secret for one-time password generation, parameter could be a hex string or a base32 string
is_base32_secret	The boolean identifier, set it true, when secret is a base32 string. This is optional parameter with default value – false.
period	The period in seconds for generated password, this parameter determines lifetime for one-time password. This is optional password with default value – 30 seconds.
otp_format	The password format from supporting password format list, this is optional parameter
hash	The name of hashing algorithm, this is optional parameter. List of the hashing algorithm provided in Python library hashlib.algorithms_guaranteed

Table 3-98 The one-time password's formats

One-time password format	Description
dec4	4 decimal digits
dec6	6 decimal digits
dec7	7 decimal digits
dec8	8 decimal digits

Resource will return a JSON object with status of the enrollment process.

The TOTP authentication method supports this set of the enrollment reasons.

Table 3-99 The TOTP method's enrollment reasons.

Enrollment reason value	Description
TOTP_SCAN_QR	Method wait when QR code with secret will be scanned

Example

On the following example, the user JSmith from the COMPANY repository will enroll the time based one-time password authentication method, user has a login session with the identifier “JXjt1OLSRLcGeJ9G9Unt9HX4aq5lHYmB”.

```
HTTP POST
https://authserver.example.com/api/v1/enroll
Request
{
"method_id":"TOTP:1",
"login_session_id":"JXjt1OLSRLcGeJ9G9Unt9HX4aq5lHYmB"
}
Response
{
"enroll_process_id":"rfiU0Xf6ghvV03HGiPvV0DV6fhHtYVTF"
}
HTTP POST
https://authserver.example.com/api/v1/enroll/rfiU0Xf6ghvV03HGiPvV0DV6fhHtYVTF/
do_enroll
Request
{
"response":
{
"secret":"12345678901234567890",
"period":60
},
"login_session_id":"JXjt1OLSRLcGeJ9G9Unt9HX4aq5lHYmB"
}
Response
{
"reason":"",
"msg":"",
"status":"OK",
"method_id":"TOTP:1"
}
```

3.10.14 Voice Call Authentication Method

The voice call authentication method provide users' authentication by a pin code inputted via phone. The voice call authentication method could be used as a single method in the authentication chain or as a part of the authentication chain.

Authentication

For the authentication, you should create a logon process with the voice call authentication method or continue with the voice call authentication method in case, when the authentication chain has more than one authentication methods. This method does not accept data from user. This method has two steps: on first step you should make an empty POST request for starting calling to the user's phone, on next steps you should make an empty POST request for checking authentication status.

The voice call authentication method supports the list of the authentication reasons.

Table 3-100 *The voice call method's authentication reasons*

Reason value	Description
VOICE_PIN_NOT_VERIFIED	The PIN code provided for the authentication was not verified
VOICE_CALL_IN_PROGRESS	The voice call in progress
VOICE_PIN_EXPIRED	The PIN code provided for the authentication was expired
VOICE_CANNOT_CALL	The voice call authentication method cannot start call
VOICE_CALL_INITIATED	The voice call authentication method was initiated

Example

On the following example, the user JSmith from the COMPANY repository will try to authenticate by the voice call authentication method, user already has an endpoint session with the identifier "655WwK8qwGt8cJlhMe82o3HVJWFFi0ec".

```
HTTP POST
https://authserver.example.com/api/v1/logon
Request
{
"method_id":"VOICE:1",
"user_name":"COMPANY\\JSmith",
"event":"NAM",
"endpoint_session_id":"655WwK8qwGt8cJlhMe82o3HVJWFFi0ec"
}
Response
{
"event_data_id":"OSLogon",
"event_name":"NAM",
"reason":"PROCESS_STARTED",
"chains":[
{
"is_enabled":true,
"image_name":"default",
"short_name":"",
"is_trusted":true,
"apply_for_ep_owner":false,
"methods":["VOICE:1"],
"name":"Voice call",
"position":0
}],
"logon_process_id":"FjVLwfybejy1brptN45p9uQvuVva5hA5",
"completed_methods":[],
"plugins":[],
"current_method":"VOICE:1",
"status":"MORE_DATA",
"msg":"Process started"
}
On this request we start calling
HTTP POST
https://authserver.example.com/api/v1/logon/FjVLwfybejy1brptN45p9uQvuVva5hA5/
do_logon
Request
{
"response":{},
"endpoint_session_id":"655WwK8qwGt8cJlhMe82o3HVJWFFi0ec"
}
Response
{
"event_data_id":"OSLogon",
"event_name":"NAM",
"reason":"VOICE_CALL_INITIATED",
"chains":[
{
"is_enabled":true,
"image_name":"default",
"short_name":"",
"is_trusted":true,
"apply_for_ep_owner":false,
"methods":["VOICE:1"],
"name":"Voice call",
"position":0
}],
"logon_process_id":"FjVLwfybejy1brptN45p9uQvuVva5hA5",
"completed_methods":[],
"plugins":[],
"current_method":"VOICE:1",
"status":"MORE_DATA",
"msg":"Call initiated"
}
HTTP POST
https://authserver.example.com/api/v1/logon/FjVLwfybejy1brptN45p9uQvuVva5hA5/
do_logon
Request
{
"response":{},
"endpoint_session_id":"655WwK8qwGt8cJlhMe82o3HVJWFFi0ec"
}

Response
{
"event_data_id":"OSLogon",
"event_name":"NAM",
"reason":"VOICE_CALL_IN_PROGRESS",
"chains":[
{
"is_enabled":true,
"image_name":"default",
"short_name":"",
"is_trusted":true,
"apply_for_ep_owner":false,
"methods":["VOICE:1"],
"name":"Voice call",
"position":0
}],
"logon_process_id":"FjVLwfybejy1brptN45p9uQvuVva5hA5",
"completed_methods":[],
"plugins":[],
"current_method":"VOICE:1",
"status":"MORE_DATA",
"msg":"Call in progress"
}
```

User entered a PIN code from the phone and authentication status changed

```
HTTP POST
https://authserver.example.com/api/v1/logon/FjVLwfybejy1brptN45p9uQvuVva5hA5/
do_logon
Request
{
"response":{},
"endpoint_session_id":"655WwK8qwGt8cJlhMe82o3HVJWFFi0ec"
}
Response
{
"current_method":"VOICE:1",
"user_cn":"JSmith",
"user_name_netbios":"COMPANY\\JSmith",
"user_dn":"CN=JSmith,CN=Users,DC=company,DC=com",
"reason":"CHAIN_COMPLETED",
"event_name":"NAM",
"user_email":"jsmith@company.com",
"msg":"Welcome!",
"user_name":"COMPANY\\JSmith",
"repo_obj_id":"2d3c89ccb3ea7b4dacbdfda13e26f450",
"data_id":"OSLogon",
"logon_process_id":"FjVLwfybejy1brptN45p9uQvuVva5hA5",
"repo_id":"d3fba00652d211e5a19a000c2951aca4",
"user_sid_hex":"010500000000000515000000d9ae14ff677e53c4ea58a768f40100",
"status": "OK",
"event_data_id":"OSLogon",
"chains":[
{
"is_enabled":true,
"image_name":"default",
"short_name":"",
"is_trusted":true,
"apply_for_ep_owner":false,
"methods":["VOICE:1"],
"name":"Voice call",
"position":0
}],
"user_sid":"S-1-5-21-4279545561-3293806183-1755797738-500",
"login_session_id":"znX7jRuvaZogxGpPW5zlGt77f4FsJSzL",
"completed_methods":["VOICE:1"],
"user_id":"ab2b845652d311e5a19a000c2951aca4",
"plugins":["LdapRules"],
"user_mobile_phone":"+16086783619"
}
```

Enrollment

For the enrollment, the voice call authentication method you should start an enrollment process and provide the enrollment data as a JSON container.

***Table 3-101** The enrollment data for the voice call authentication method*

Parameter name	Description
pin	The PIN code for the user's authentication

Resource will return JSON object with status of the enrollment process.

Example

On the following example, the user JSmith from the COMPANY repository will enroll the password authentication method, user has a login session with identifier the "PJHyEFQFTjt7fgHmW28avMnKFMHldiT7".

```
HTTP POST
https://authserver.example.com/api/v1/enroll
Request
{
"method_id":"VOICE:1",
"login_session_id":"PJHyEFQFTjt7fgHmW28avMnKFMHldiT7"
}
Response
{
"enroll_process_id":"lHpM6G8Z8DiHBw0iFoAK8LfCMDxIu9hk"
}
HTTP POST
https://authserver.example.com/api/v1/enroll/lHpM6G8Z8DiHBw0iFoAK8LfCMDxIu9hk/
do_enroll
Request
```

```
{
"response":
{
"pin":"135978"
},
"login_session_id":"PJHyEFQFTjt7fgHmW28avMnKFMHldiT7"
}
Response
{
"reason":"",
"msg":"",
"status":"OK",
"method_id":"VOICE:1"
}
```

3.10.15 NotarisID Authentication Method

The NotarisID authentication method is a method for the authentication by Notaris identifier. The NotarisID authentication method could be used as a single method in the authentication chain or as a part of the authentication chain. For more information, see (https://www.notarisid.nl/).

Authentication

For the authentication, you should create a logon process with the NotarisID authentication method or continue with the NotarisID authentication method in case, when the authentication chain has more than one authentication methods. When authentication will start, appliance create a request to the NotarisID system with enrolled user's Notaris Indentifier, for checking status of authentication in NotarisID system, you should send an empty response to the appliance, the appliance will return status of method authentication.

The NotarisID method supports the list of the authentication reasons.

Table 3-102 *NotarisID method's authentication reasons.*

Reason Value	Reason Description
OK	The authentication was successful.
WRONG_STATE	The NotarisID system return a wrong state.
TOO_OFTEN_POLL	The requests to the NotarisID system is too often.
WAITING_FOR_USER_ACCEPT	The appliance wait for accepting authentication by user in NotarisID system.
ERROR	The authentication failed.

Example

On next example, the user JSmith form the Advanced Authentication repository will try to authenticate by the NotarisID, user has an endpoint session with the identifier "zQ9YQ1Txpax09iRBBTQJN71tSDgsMiuA".

HTTP POST`https://authserver.example.com/api/v1/logon`

```
Request
```

```
{
  "method_id":"NOTARIS_ID:1",
  "user_name":"COMPANY\\JSmith",
  "event":"NAM",
  "endpoint_session_id":"zQ9YQ1Txpax09iRBBTQJN71tSDgsMiuA"
}
Response
{
  "reason":"PROCESS_STARTED","current_method":"NOTARIS_ID:1","msg":"Process
started","chains":[
  {
    "is_trusted":null,
    "is_enabled":true,
    "short_name":"",
    "position": 0,
    "methods":["NOTARIS_ID:1"],
    "name": "NotarisID",
    "apply_for_ep_owner":false,
    "image_name":"default"
  }],
  "event_data_id":"OSLogon",
  "status":"MORE_DATA","plugins":[],
"logon_process_id":"ULMzchEGWNPnutRROYDM18p24tTvQh2g",
  "completed_methods":[],
  "event_name":"NAM"}
```

HTTP POSThttps://authserver.example.com/api/v1/logon/ULMzchEGWNPnutRROYDM18p24tTvQh2g/do_logon

```
Request
{
  "response": {},
  "endpoint_session_id":"zQ9YQ1Txpax09iRBBTQJN71tSDgsMiuA"
}
Response
{
  "reason":"OK",
  "user_id":"ab2b845652d311e5a19a000c2951aca4",
  "event_name":"NAM",
  "chains":[
  {
    "is_trusted":null,
    "is_enabled":true,
    "short_name":"",
    "position": 0,"methods":["NOTARIS_ID:1"],
    "name": "NotarisID",
    "apply_for_ep_owner":false,
    "image_name":"default"
  }],
  "data_id":"OSLogon",
  "repo_obj_id":"2d3c89ccb3ea7b4dacbdfda13e26f450",
  "msg":"Welcome!",
  "plugins":["LdapRules"],
  "user_name":"COMPANY\\JSmith",
  "user_email":"jsmith@company.com",
  "current_method":"NOTARIS_ID:1",
  "user_sid":"S-1-5-21-4279545561-3293806183-1755797738-500",
  "status":"OK",
  "logon_process_id":"ULMzchEGWNPnutRROYDM18p24tTvQh2g",
  "user_dn":"CN=JSmith,CN=Users,DC=company,DC=com",
  "user_sid_hex":"010500000000000515000000d9ae14ff677e53c4ea58a768f40100",
  "login_session_id":"6XFwPdePJ6y0pq9dBWFIcyerTVH2yMRJ",
  "user_mobile_phone":"+16086783619",
  "event_data_id":"OSLogon",
  "completed_methods":["NOTARIS_ID:1"],
  "user_name_netbios":"COMPANY\\JSmith",
  "repo_id":"d3fba00652d211e5a19a000c2951aca4",
  "user_cn":"JSmith"
}
```

Enrollment

For the enrollment, the NotarisID authentication method you should start an enrollment process and provide an enrollment data as JSON container.

Table 3-103 Enrollment data for the NotarisID method

Parameter Name	Parameter Description
notaris_id	The NotarisID identifier.

Resource will return a JSON object with status of the enrollment process.

The NotarisID authentication method supports this set of the enrollment reasons.

Table 3-104 NotarisID method's enrollment reasons

Reason Value	Reason Description
NO_NOTARISID	The identifier is not valid NotarisID identifier.

Example

On next example, the user JSmith from the Advanced Authentication repository will enroll the NotarisID method, user has a login session with the identifier "wQabBzcnJTBqOqTdClHJTtrkpHFUzg40".

```
HTTP POST

https://authserver.example.com/api/v1/enroll

Request

{

  "method_id":"NOTARIS_ID:1",

  "login_session_id":"wQabBzcnJTBqOqTdClHJTtrkpHFUzg40"

}

Response

{

  "enroll_process_id":"j7wpbJRTJ3LHIhSFSn2UWAEnTA15ldTK"

}

HTTP POST

https://authserver.example.com/api/v1/enroll/j7wpbJRTJ3LHIhSFSn2UWAEnTA15ldTK/
do_enroll

Request

{

  "response":

  {

    "notaris_id":"CZA7GXTO@notarisid"

  },

  "login_session_id":"wQabBzcnJTBqOqTdClHJTtrkpHFUzg40"

}
```

```
Response
{
  "reason":"",
  "msg":"",
  "status":"OK",
  "method_id":"NOTARIS_ID:1"
}
```

3.10.16 PKI Authentication Method

The PKI authentication method is a method for the authentication by public key infrastructure, with this method user have a private key on a smartcard and public key on appliance, user sign a challenge from the appliance and appliance validate this request. The PKI authentication method could be used as a single method in the authentication chain or as a part of the authentication chain.

Authentication

For the authentication, you should create a logon process with the PKI authentication method or continue with the PKI authentication method in case, when the authentication chain has more than one authentication methods. You should provide a JSON container with user's PKI data by POST request. The PKI method have a two steps, at first you should get a challenge from the appliance with empty request, at second step you should provide signed challenge to the appliance for validation.

Table 3-105 *The authentication data for the PKI authentication method for second step*

Parameter Name	Parameter Description
signature	The signature for validation.
padding	The padding of the signature.The method supports PKCS#1, PSS, and OAEP.
hash	The name of the hash algorithm.The method supports SHA1, SHA224, SHA256, SHA384, SHA512, and RIPEMD160.

Resource will return a JSON-object with information about current state of the authentication and the signature on first step.

Table 3-106 *The response parameter for PKI method.*

Parameter Name	Parameter Description
challenge	The challenge for signing.

The PKI authentication method supports the list of the authentication reasons.

Table 3-107

Reason Value	Reason Description
PKI_WAITING_AUTH_SIGN	The appliance waiting for the authentication sign.
PKI_SIGN_VERIFICATION_FAILED	The verification of the sign was failed.
PKI_CERT_VALIDATION_FAILED	The validation of the certificate was failed.
PKI_WRONG_CARD	The card is wrong.

```
HTTP POST
https://authserver.example.com/api/v1/logon
Request
{
  "method_id":"PKI:1",
"user_name":"COMPANY\\JSmith",
  "event":"NAM",
"endpoint_session_id":"P7p3JJuenqo0SnyJ4HnbRbbJIqDhtt0u"
}
Response
{
"reason":"PROCESS_STARTED",
  "current_method":"PKI:1",
"msg":"Process started",
  "chains":[
{
    "is_trusted":null,
"is_enabled":true,
    "short_name":"",
"position": 0,
    "methods":["PKI:1"],
"name": "PKI",
    "apply_for_ep_owner":false,
"image_name":"default"
  }],
"event_data_id":"OSLogon",
  "status":"MORE_DATA",
"plugins":[],
  "logon_process_id":"t9HZES2DFD6vDjhA3wNtizyjAuqMTrwi",
"completed_methods":[],
  "event_name":"NAM"
}
HTTP POST
https://authserver.example.com/api/v1/logon/t9HZES2DFD6vDjhA3wNtizyjAuqMTrwi/
do_logon
Request
{
"response": {},
  "endpoint_session_id":"P7p3JJuenqo0SnyJ4HnbRbbJIqDhtt0u"
}
Response
{
  "reason":"PKI_WAITING_AUTH_SIGN",
"current_method":"PKI:1",
  "msg":"Process started",
"chains":[
  {
"is_trusted":null,
    "is_enabled":true,
"short_name":"",
    "position": 0,
"methods":["PKI:1"],
    "name": "PKI",
"apply_for_ep_owner":false,
    "image_name":"default"
}],
  "event_data_id":"OSLogon",
"status":"MORE_DATA",
  "plugins":[],
"logon_process_id":"t9HZES2DFD6vDjhA3wNtizyjAuqMTrwi",
  "completed_methods":[],
"event_name":"NAM",
  "challenge":"f81e9d6882aca80cbe97e291ee5771aba7cc13facb3c79a5ae924e788bc4f7d2"
}
HTTP POST
https://authserver.example.com/api/v1/logon/t9HZES2DFD6vDjhA3wNtizyjAuqMTrwi/
do_logon
Request
{
"response":
  {
"signature":"d84f3a9b7244031aa5....42d123bdb715a153974e992b16d02",
    "padding":"PKCS#1",
"hash":"SHA1"
  },
"endpoint_session_id":"P7p3JJuenqo0SnyJ4HnbRbbJIqDhtt0u"
}
Response
{
"reason":"CHAIN_COMPLETED",
  "user_id":"ab2b845652d311e5a19a000c2951aca4",
"event_name":"NAM",
  "chains":[
{
    "is_trusted":null,
"is_enabled":true,
    "short_name":"",
"position": 0,
    "methods":["PKI:1"],
"name": "PKI",
    "apply_for_ep_owner":false,
"image_name":"default"
  }],
"data_id":"OSLogon",
  "repo_obj_id":"2d3c89ccb3ea7b4dacbdfda13e26f450",
"msg":"Welcome!",
  "plugins":["LdapRules"],
"user_name":"COMPANY\\JSmith",
  "user_email":"jsmith@company.com",
"current_method":"PKI:1",
  "user_sid":"S-1-5-21-4279545561-3293806183-1755797738-500",
"status":"OK",
  "logon_process_id":"t9HZES2DFD6vDjhA3wNtizyjAuqMTrwi",
"user_dn":"CN=JSmith,CN=Users,DC=company,DC=com",
  "user_sid_hex":"01050000000000005150000000d9ae14ff677e53c4ea58a768f40100",
"login_session_id":"6XFwPdePJ6y0pq9dBWFIcyerTVH2yMRJ",
  "user_mobile_phone":"+16086783619",
"event_data_id":"OSLogon",
  "completed_methods":["PKI:1"],
"user_name_netbios":"COMPANY\\JSmith",
  "repo_id":"d3fba00652d211e5a19a000c2951aca4",
"user_cn":"JSmith"
}
```

Enrollment

For the enrollment, the PKI authentication method you should start an enrollment process and provide an enrollment data as JSON container. The method can be enrolled in two formats: modulus with exponent (generate a key pair) or card certificate. You can choose a required format for your case.

Table 3-108 *The enrollment data for the PKI method*

Parameter Name	Parameter Description
card_uid	The card UID.
modulus	The modulus for the sign.
exponent	The exponent for the sign.
card_cert	The card certificate in DER format.

Resource will return a JSON object with status of the enrollment process.

The PKI authentication method supports this set of the enrollment reasons.

Table 3-109 *The emergency password method’s enrollment reasons*

Reason Value	Reason Description
PKI_CERT_VALIDATION_FAILED	The validation of the certificate was failed.

Example

On next example, the user JSmith from the Advanced Authentication repository will enroll the PKI authentication method, user has a login session with the identifier “WXogCpZcHlsGsmmYiXkcxbg1qJMHJ4eD”.

```
HTTP POST
```

```
https://authserver.example.com/api/v1/enroll
```

```
Request
```

```
{

  "method_id":"EMERG_PASSWORD:1",
  "login_session_id":"WXogCpZcHlsGsmmYiXkcxbg1qJMHJ4eD"
}
Response
{
  "enroll_process_id":"hn0k1pTx1T6MQwOIbhBPUlLIopROig4Q"
}
HTTP POST
https://authserver.example.com/api/v1/enroll/hn0k1pTx1T6MQwOIbhBPUlLIopROig4Q/
do_enroll
Request
{
  "response":
  {
    "card_uid":"11223344",
    "modulus":"a49c35fdc669519e9d0c713c....91daaa9d2604eeeaad73d13b1",
    "exponent":"010001"
  },
  "login_session_id":"WXogCpZcHlsGsmmYiXkcxbg1qJMHJ4eD"
}
Response
{
  "reason":"",
  "msg":" Enrollment complete" ",
```

3.11 Errors

On error server, return JSON-object with error information.

Table 3-110 *JSON-object for error.*

Parameter name	Description
errors	Array of errors JSON-objects. Each object contains information about error.
status	Current status

Error object is JSON-object with parameters.

Table 3-111 JSON-object with detailed information about error.

Parameter name	Description
name	Error name
location	Location, where error occurs.
description	Error full description.

Example

The following example shows simple server error response.

```
{
"status":"error",
"errors":
[
{
"description":"You are logged to empty event.data_id. It is not for data",
"location":"server",
"name":"AuError"
}
]
}
```

3.12 Troubleshooting

Table 3-112 Errors' codes and possible solution.

HTTP status	Solution
400	Error in API method, take error object from response to get more information.
404	API method not found on server.
434	Login session not found or expired, update logon session
434	Endpoint session not found or expired, update endpoint session.

4 Usage of Device Services

Currently the supported opened ports are {8440, 8441, 8442} but it is better to use 8440, as other ports may be deprecated in the future releases.

4.1 Card Plug-in

To check the Card Service you may open the following URL: https://127.0.0.1:8440/api/v1/card/getmessage?nowait.

The response format:

```
{
result: [<status>],
cardid: <card id>,
readerid: <reader id>
}
```

The following statuses are implemented:

- NO_READER means that the Card service didn't detect a card reader connected,
- READER_ON means that the Card service detected a card reader connected,
- NO_CARD means that there is no card on the reader,
- CARD_ON means that a card is presented to the reader.

NOTE: cardid is used only with CARD_ON and NO_CARD statuses.

Examples of commands:

- https://127.0.0.1:8440/api/v1/card/getmessage?nowait - immediately returns a current status. Possible values [NO_READER, NO_CARD, CARD_ON]
- https://127.0.0.1:8440/api/v1/card/getmessage?wait - waits for a next event (e.g. card presented or card removed)

  NOTE: When you disconnect the reader with a card on, two messages will arrive: NO_CARD, NO_READER. But the first one will be caught with getmessage?wait.When you plug in a reader with a card on, there will be the two events: READER_ON, CARD_ON. And as a result READER_ON will be returned.

- https://127.0.0.1:8440/api/v1/card/getreaderon?nowait - immediately returns READER_ON if a reader is attached and NO_READER otherwise.
- https://127.0.0.1:8440/api/v1/card/getreaderon?wait - immediately returns READER_ON if a reader is attached or waits till it's attached
- https://127.0.0.1:8440/api/v1/card/getcardon?nowait - immediately returns NO_READER if a reader isn't attached, NO_CARD if a card isn't presented or CARD_ON if a card is presented
- https://127.0.0.1:8440/api/v1/card/getcardon?wait - immediately returns NO_READER if a reader isn't attached or wait till the card will be presented on a reader.

NOTE: It will wait the next tap of a card even if a card is already on a reader.

- https://127.0.0.1:8440/api/v1/card/getcardoff?nowait&cardid=<cardid> - immediately returns NO_READER if a reader isn't attached, NO_CARD if a card isn't presented on the reader or CARD_ON if a card is presented on the reader. Use cardid to wait when a specific card is removed.
- https://127.0.0.1:8440/api/v1/card/getcardoff?wait - returns immediately with NO_READER if a reader isn't attached. If there is no card presented on a reader, it returns NO_CARD immediately else waits till the card is removed from the reader
- https://127.0.0.1:8440/api/abort?cancel-cookie=xxx - all of the "wait" methods support cancel-cookie=xxx parameter.E.g. https://127.0.0.1:8440/api/v1/card/getmessage?wait&cancel-cookie=xxx.And by calling abort with a cancel-cookie, all waiting methods with the same specfied cookie are terminated.

4.2 FIDO U2F Plug-in

To check the FIDO U2F Service you may open the following URL: https://127.0.0.1:8441/api/v1/fidou2f/abort (https://127.0.0.1:8441/api/v1/fidou2f/abort)The service should return: { "result":"ok" } when a FIDO U2F token is connected.

Available methods

FIDO U2F Service provides the following POST-methods:

https://127.0.0.1:8441/api/v1/fidou2f/sign - Performs the U2F Authenticate operation.

```
{
"signRequests":
[
{"challenge":"tRiTY3C8YerfmH6IIlfoCZjs5CMkKUWDrNhS7v5gCPQ",
"version":"U2F_V2,
"keyHandle":"knQD88Ue6ZT6tyutHr8ipZaiTRV2uT9qzwGqWjYo5HCwAiV5z2kc1vr08tWbdOLQ4S-
ODg09vpp62P6owh4qmQ",
"appId":"https://demo.yubico.com"
}
]
}
```

https://127.0.0.1:8441/api/v1/fidou2f/register - Performs the U2F Register operation.

```
{
"registerRequests":
[
{"challenge":"tRiTY3C8YerfmH6IIlfoCZjs5CMkKUWDrNhS7v5gCPQ",
"version":"U2F_V2,
"appId":"https://demo.yubico.com"
}
],
"signRequests":[]
}
```

signRequest can be empty, or contain serial of for the key handle validation

```
{
"challenge":"tRiTY3C8YerfmH6IIlfoCZjs5CMkKUWDrNhS7v5gCPQ",
"version":"U2F_V2,
"keyHandle":"knQD88Ue6ZT6tyutHr8ipZaiTRV2uT9qzwGqWjYo5HCwAiV5z2kc1vr08tWbdOLQ4S-
ODg09vpp62P6owh4qmQ",
"appId":"https://demo.yubico.com"
}
```

In case of success both methods above returns JSON reply in the U2F specification format:

or an error:

```
{ "errorCode"=1, "errorMessage"="Error Text"}
```

where:

errorCode - error code

errorMessage - additional error text

errorCode description:

1. Device other error. If the token is missing, errorMessage contains "Please connect a U2F token."
2. Device bad request. The visited URL doesn't match the App ID or not using HTTPS
3. Configuration unsupported
4. Token is not registers - for authentication process or token already registered - for register process, to enable this check, specify "signRequests" in the body of the register request).
5. Timeout - no answer from token. (if the user didn't press a button within a given timeout)

And the following GET-methods:

https://127.0.0.1:8441/api/v1/fidou2f/abort - Aborts all pending operations

4.3 Fingerprint Plug-in

To check the WBF Capture Service you may open the following URL: https://127.0.0.1:8442/api/v1/fingerprint/capture. Present your finger on the reader while the URL is loading.

The following fields are included into the output:

- captureStatus - can be 'Ok', 'Timeout', 'Error', 'NoReader'.
- Width, Height - fingerprint image size (in pixels).
- Dpi - dots per inch (used on matching side).
- BitsPerPixel - bits per pixel (usually 8 bits).
- BytesPerLine - bytes per one line in image (include align).
- Image - fingerprint image encoded using base-64 in gray scale.

E.g.
{"BitsPerPixel":8,"BytesPerLine":256,"Dpi":508,"Height":360,"Image":"<fingerprintdata>","Width":256,"captureStatus":"Ok"}.

4.4 PKI Plug-in

PKI plug-in supports the following options:

- `vendorModule=eTPKCS11.dll` - PKCS#11 implementation library of a needed vendor.
- `hash=SHA1 or SHA224, SHA256` (this is a default value if not presented), SHA384, SHA512, RIPEMD160.
- `padding=PKCS#1` (this is a default value if not presented) or PSS, OAEP.
- `modulusBits=2048` - key size (this is a default value if not presented). E.g. eToken PRO 32k doesn't support it and you need to set 1024 to use it.
- `blockingMode=True`. The default value is True. OpenSC supports the 'waiting for card' mechanism not completely and it requires to change the option to False. The most of vendors should work fine with the default mode.

PKI plugin uses the simulatar API for card / token detection and two new POST methods pki/enroll, pki/login:

Available methods:

Card service provides the following POST-methods

- `https://127.0.0.1:8440/api/v1/pki/getcertificates` - GET method to get all certificates from a token

```
{ "readerid"=0, "certificates" : [{
"keypairid":"9beb","certificate":"30820371308202daa00....0b90d7290a1a76b0450264dd5
36d2cb057230f8dbfa8cfda05"}] }
```

slotid - slot ID

keypairid - id of the key pair in the certificate. Save it and use later for future logon operations.

certificate - certificate value in DER format.

```
- https://127.0.0.1:8440/api/v1/pki/generatekeypair- POST method, Request Body:
{"pin":"your_pin"}
```

// Replace with your token pin or empty if there is no pin

```
{ "readerid"=your_reader_id, "keypairid":"6f4712e554544ac3",
"modulus":"a1709fb049c35fdc6695193e9dd980c713c....91daaa9d2604eeeaad73d13b1",
"exponent":"010001"}
```

keypairid - id of the key pair in the certificate. save it and use later for future logon operations.

modulus - modulus

exponent - big exponent

```
- https://127.0.0.1:8440/api/v1/pki/signchallenge - POST method, Request Body:
{"challenge":"3128", "pin":"your_pin", "keypairid":"9beb" }
```

challenge in hex-string format(even length, sice one byte is two hex symbols)

pin - pin to the token

keypairid - id of the keypair from token, you can get it from previous enroll operation in case of success it returns signature for the given challenge in the hex format`{ "readerid"=your_reader_id, "hash":"SHA1", "padding":"PKCS#1", "signature":"58ad84f3a9b7244031aa55c0d0ad753b1a480ae709a37210d48....493130d7b11f128ea2be1fcc42d123bdb715a153974e992b16d022" }`

hash - used hash method

padding - used padding

- https://127.0.0.1:8440/api/v1/pki/verifychallenge - POST method, Request Body {"challenge":"3128", "pin":"your_pin", "keypairid":"9beb", "signature":"58ad84f3a9b72....bdb715a153974e992b16d022" }

in case of an error two methods above returns an error:

`{ "errorCode"="ERROR_ID"}`

Possible values of ERROR_ID:

`PLUGIN_NOT_INITTED` - not initted library, etc. dll was not provided

`METHOD_NOT_FOUND` - method not found

`NO_CARD` - no token or no card are presented. Use wait methods to get an event.

`JSON_PARSE_FAILED` - bad request body

`WRONG_PIN`- Wrong PIN

`GET_PRIVATE_KEY_FAILED` - error getting a private key from a token

`OPERATION_FAILED`- general operation failure

- `https://127.0.0.1:8440/api/v1/pki/getmessage?nowait` - returns immediately the current status. Possible values [NO_READER, NO_CARD, CARD_ON].

- `https://127.0.0.1:8440/api/v1/pki/getmessage?wait` - waits till the next event occurs.

NOTE: When you plug off the reader with a card on, two messages are displayed: `NO_CARD, NO_READER`. But the first one will be catch with getmessage?wait.

When you plug in a reader with a card on, occures `READER_ON, CARD_ON`. And as a result `READER_ON` will be returned.

- `https://127.0.0.1:8440/api/v1/pki/getreaderon?nowait` - returns immediately with READER_ON if it's attached and NO_READER otherwise.

- `https://127.0.0.1:8440/api/v1/pki/getreaderon?wait` - returns immediately with READER_ON if a reader is attached or waits till it's attached.

- `https://127.0.0.1:8440/api/v1/pki/getcardon?nowait` - returns immediately with NO_READER if a reader isn't attached, NO_CARD if a card isn't inserted or CARD_ON if a card is inserted.

- `https://127.0.0.1:8440/api/v1/pki/getcardon?wait` - returns immediately with NO_READER if a reader isn't attached or wait till the card will be on a reader.

NOTE: It will wait the next tap of a card even if a card is already on a reader.

- https://127.0.0.1:8440/api/v1/pki/getcardoff?nowait&cardid=<cardid> - returns immediately with NO_READER if a reader isn't attached, NO_CARD if a card isn't inserted or CARD_ON

if a card is inserted. Use cardid to wait when a specific card is removed.

- https://127.0.0.1:8440/api/v1/card/getcardoff?wait - returns immediately with NO_READER if a reader isn't attached. if there is no card on a reader return NO_CARD immediately else waits till the card is removed from the reader

- https://127.0.0.1:8440/api/abort?cancel-cookie=xxx - all of the wait methods support cancel-cookie=xxx parameter.

For example, https://127.0.0.1:8440/api/v1/card/getmessage?wait&cancel-cookie=xxx.

And by calling abort with a cancel-cookie, all waiting methods with the same specfied cookie are terminated.

Response format:

```
Response format

{

result: [NO_READER, READER_ON, NO_CARD, CARD_ON],

cardid: <card id>,

readerid: <reader id>

}

cardid is used only with CARD_ON, and NO_CARD result.
```