Proxedo API Security based on VM environment: Administration Guide

2025-12-16

Preface

Typographical conventions

Before you start using this guide, it is important to understand the terms and typographical conventions used in the documentation. For more information on specialized terms and abbreviations used in the documentation, see the Glossary at the end of this document.

The following text formatting principles and icons identify special information in the document.

Tips provide best practices and recommendations.

Notes provide additional information on a topic, and emphasize important facts and considerations.

Warnings mark situations where loss of data or misconfiguration of the device is possible if the instructions are not obeyed.

`Command`
: Commands you have to execute.

Emphasis
: Reference items, additional readings.

`/path/to/file`
: File names.

Parameters
: Parameter and attribute names.

Additional marks used specifically in the Web User Interface (UI):

Key	Description
*	The elements marked with * in the configuration reference tables are mandatory to be configured.
(Default)	For some of the configuration elements there are recommended default values, marked as (Default). In case the value is not defined during the configuration, the default value will be considered for the actual element.
+	By clicking this sign you can add the actual element to the list of configuration elements.

Contact and support information

This product is developed and maintained by Balasys IT Ltd..

Contact:

```
Balasys IT Ltd.
4 Alíz Street
H-1117 Budapest, Hungary
Tel: +36 1 646 4740
E-mail: <info@balasys.hu>
Web: http://balasys.hu/
```

Sales contact

You can directly contact us with sales-related topics at the e-mail address <sales@balasys.hu>, or leave us your contact information and we call you back.

Support contact

To access the Balasys Support System, sign up for an account at the Balasys Support System page. Online support is available 24 hours a day.

Balasys Support System is available only for registered users with a valid support package.

Support e-mail address: <support@balasys.hu>.

Training

Balasys IT Ltd. holds courses on using its products for new and experienced users. For dates, details, and application forms, visit the https://www.balasys.hu/en/services#training webpage.

1. Scope of this document

This document describes installation, configuration and operation of Proxedo API Security in VM. The purpose of this document is to present the designed approach for different installation scenarios, base system configuration, and the usage of the Web User Interface (UI). It also documents common use cases for operation and troubleshooting. The primary intended audience of this document are system engineers and system designers for configuring Proxedo API Security systems.

2. Introduction to Proxedo API Security

2.1. What is Proxedo API Security

The Proxedo API Security (PAS) is a security solution that protects API serving endpoints. It is positioned in the network flow between consumers of the APIs (clients) and backend solutions serving the API (servers) as a transparent HTTP proxy.

Proxedo API Security can:

- handle incoming Transport Layer Security v1 (TLS) connections from clients & outgoing TLS connections to servers separately and selectively
- verify that the communication conforms to HTTP specifications
- verify that the content of the messages conform to their specified content type
- verify that the content of messages conform to API specification(s) as described in schemas
- evaluate the level of risk with regards to the API call using the data collected from call data

- provide rule-based protection against a variety of web-based application layer attacks
- extract parts of the content of the messages and relay them to external data stores such as log servers, SIEM systems or other data warehouses

2.2. Where to start

Depending on what you need to do the following starting points are suggested:

- To understand what the product does and how, see Overview of Proxedo API Security.
  - If you are familiar with API terminology jump right to Architecture for Proxedo API Security.
- See Installation of Proxedo API Security based on VMs if you need to set up a new PAS.
- The Operation of Proxedo API Security based on VMs chapter is about how to manage a working system on the level of the operating system.
- Configuration of Proxedo API Security on the Web User Interface contains in-depth information about everything that can be configured with the help of the Web User Interface.
- If you are already familiar with the system and need to find a component that suits your needs consult the Matcher types, Comparators, Extractor types or Insight Target.

3. Overview of Proxedo API Security

3.1. Main features

3.1.1. TLS

Transport Layer Security v1 (TLS) (successor of the now obsoleted Secure Socket Layer v3 (SSL)) is a widely used crypto protocol, guaranteeing data integrity and confidentiality in many PKI and e-commerce systems.

The TLS framework inspects TLS connections, and also any other connections embedded into the encrypted TLS channel. TLS connections initiated from the client are terminated on the Proxedo API Security, and two separate TLS connections are built: one between the client and the firewall, and one between the firewall and the server. If both connections match the configuration settings of PAS (for example, the certificates are valid, and only the allowed encryption algorithms are used), PAS inspects the protocol embedded into the secure channel as well. Note that the configuration settings can be different for the two connections, for example, it is possible to permit different protocol versions and encryption settings.

3.1.2. Enforcement

Proxedo API Security acts as an HTTP proxy and verifies that the traffic passing through conforms to HTTP's specifications. By using OpenAPI schemas, as defined in OpenAPI specifications (also known as Swagger), it also verifies that the traffic passing through conforms to the API endpoint's specification and can log or deny non-conforming traffic.

PAS also provides its own versatile filtering system to control passing traffic.

3.1.3. Fraud Detection

The Fraud Detection module of Proxedo API Security reduces the number of fraudulent transactions by harnessing device fingerprinting and enriching incoming data with alternate sources to provide the best accuracy and details about transactions.

3.1.4. Rule-based Enforcement

Besides its positive security model approach, Proxedo API Security also has a web application firewall module.

The *WAF Enforcer* protects against a variety of application layer attacks including credential theft, code injection, cross-site scripting (XSS), cookie poisoning, CSRF, SQL injection, DoS, ransomware, and more.

3.1.5. Insights

With Proxedo API Security it is possible to extract business-relevant information with extremely high resolution from the traffic and relay it to external data stores where further analysis can be implemented.

Thus, it is possible to feed Log Management solutions, Monitoring and SIEM systems, Data visualization tools with data extracted from the traffic, even to the level of specific fields deep inside API calls or URI parameters.

3.1.6. Security flow

The security flow binds most of PAS's features together. It allows flexible configuration for handling the traffic. *Multiple Enforcement*, *Filter* and *Insight plugins* can be mix-and-matched with control over error policies.

3.1.7. High Availability

Proxedo API Security offers the high availability (HA) feature optionally. With the help of this feature, two identical PAS servers provide redundancy so that the network traffic is not disturbed in case any of the nodes fails. Support for synchronizing configuration and setting remote services' state is also implemented.

3.1.8. Monitoring

The Monitoring system of Proxedo API Security core leverages the widely accepted Simple Network Management Protocol (SNMP) to monitor its network components and to collect data on the components systematically. The monitoring capability of PAS core relies on the SNMP daemon. The collected data, organized into an information database and shared between the SNMP daemon and the Monitoring Manager is called Management Information Base (MIB). For the analysis of the collected data, the BALASYS-SNMP-MIB and the PAS-SNMP-MIB Management Information Base (MIB) documents can be downloaded from Balasys customer documentation. Further recommended MIB files are, SNMPv2-MIB, IF-MIB and UCD-SNMP-MIB.

For the monitoring implementation, PAS depends on the `snmpd` service on the underlying host operating system. Therefore if `snmpd` fails or is stopped, PAS also stops.

3.2. Main Concepts in Proxedo API Security

This chapter provides an overview of the Proxedo API Security solution, introduces its main concepts, and explains the relationship of the various components.

API Endpoint

Proxedo API Security protects API endpoints. An API endpoint is the serving part of the communication channel and is the collection of all functions of a service. It resides at a list of well-known top URIs under which all the functions are accessible. APIs have well-defined HTTP Endpoints for all exposed calls, resources etc., usually through providing a schema that describes all parameters of these URI paths, including possible HTTP response codes, the format and fields of the data structure in the request's and response's body.

Client

It is a consumer of API endpoints. It is the source of the requests.

Backend

The backend constitutes of one or more servers that serve the API endpoint. It receives the requests of the client and sends the responses.

HTTP message

It can be an HTTP request coming from the client or an HTTP response coming from the backend.

Call

An HTTP conversation constitutes of a request — response interchange of HTTP messages between the client and the backend. Whenever the direction is irrelevant in the context — it applies to both requests and responses — the message is named Call.

Listener

It is the part of PAS that listens to incoming traffic for given API Endpoints. It is bound to a network port. Clients address this port when accessing API Endpoints through the gateway.

TLS

Transport Layer Security is the cryptographic protocol that secures HTTPS communications. PAS can apply TLS encryption both when communicating with Clients and Backends. TLS encryption can also be used with *Syslog Insight Target* and *Elastic Insight Target*.

Security flow

It provides a collection of security rules that PAS applies to a Call. It is two series of *Plugins*: one for requests and one for responses.

Plugin

It is an element of the security flow that applies a specific security function. It has different types based on the role they do.

Decompressor

A *Plugin* responsible for decompressing compressed content in the HTTP message's body. This ensures that the original content of the message is available for processing.

Compressor

A *Plugin* responsible for compressing the result of a flow and forwarding the compressed content.

Deserializer

A *Plugin* responsible for parsing the HTTP message's body to structured data. This ensures that a message is well-formed. The structured data will also be consumed by other *Plugins* that operate on the body of the message.

Serializer

A *Plugin* responsible for serializing the structured data to the format of the HTTP message's body.

Filter

A *Plugin* that rejects calls when they match defined rules.

Enforcer

A *Plugin* that validates calls against externally defined schemas.

Insight

A *Plugin* that extracts various data from the call and sends it to external systems (log servers, SIEMs, and other data analysis tools).

Brick

They are reusable components. They can be defined on their own and then shared by multiple other components.

Error policy

It is a brick that defines what happens if the *Plugin* has found an error. It decides if calls are rejected or merely logged, and defines the details of the HTTP error response sent to the client if a call is rejected.

Matcher

It is a brick that decides if the *Plugin* should be executed for a given call by checking various data in the HTTP message.

Selector

Selector is a brick that can extract a piece of information from a call. It is used by *Insight plugins*.

Insight Target

It is a brick that defines an external system to send extracted data to. It is used by *Insight plugins*.

High Availability

This feature enables two nodes serving as redundant PAS endpoints. It helps ensure service continuity in case of a node failure while being transparent to clients.

3.3. Architecture for Proxedo API Security

Proxedo API Security is based on a micro-services architecture separated into three deployment units: *Management*, *Storage*, and *Core*. These deployment units (or infrastructure components) can be scaled or moved between hosts to accommodate different throughput and reliability requirements.

3.3.1. Management component

Responsible for handling the security component configuration of the *Core* component, while the data itself resides in the *Storage* component. Contains the following services:

Config API

Exposes a configuration API that can be used to manage the product:

- Editing the security component configuration
- Applying the security component configuration
- Monitoring service status

Config WebUI

Provides a browser-based user interface to the configuration API.

3.3.2. Storage component

Stores and distributes different versions of the security component configuration to the *Core* component. Contains the following services:

Consul

Stores the different versions of the security component configuration, and monitors the status of PAS services.

Blob Store

Stores file resources that are part of the security component configuration.

3.3.3. Core component

The *Core* services are each responsible for a well-defined subset of handling traffic between the client and the backend. Contains the following services:

Transport Director

Manages the transport layer of API connections:

- Handles network connections from the client

- Handles network connections towards the backends
- Handles TLS on these connections
- Load balances between multiple backend servers
- Load balances between multiple *Flow Directors*
- Enforces HTTP protocol validity in calls

Flow Director

Responsible for the execution of the *Plugins* in the *Endpoints'* flow and for applying *Error Policies* as necessary.

Insight Director

Manages the connections to *Insight Targets*. Responsible for sending the data collected by *Insight plugins* to *Insight Target* systems.

Monitoring Manager

Provides monitoring data about the *Core* services and the host machine via an SNMP interface.

Content Filtering Director

Provides content filtering capabilities for the *WAF Enforcer plugin*.

High Availability Director

Responsible for maintaining the High Availability of the *Core* services.

3.3.4. The configuration process

While the configuration most commonly takes place on the Web UI, the process works the same way through the configuration API.

1. When a user logs in to the *Web UI*, the currently running configuration is visible.
   - When logging in to the *Web UI* for the first time after a fresh install, the current configuration is empty. Only a few mandatory and default components are added, and some mandatory components must be added to the configuration for the first configuration to become valid.
   - The running configuration is always stored in the *Storage* component.
2. The user can edit the configuration: add new components, delete existing components, and change fields on existing components.
   - The changes the user makes are only visible to the user, other users can only see the running configuration and their own changes.
   - The user's changes are always stored in the *Storage* component.
3. Individual components and the configuration as a whole are validated.
   - Partially configured components can be saved with missing fields, but they won't become valid until all mandatory fields are properly filled.
   - An invalid configuration is still saved, and can be fixed at a later time. Every user has their own set of changes.
4. When the configuration is valid, it can be applied to the running system.
   - When a user's configuration is applied, the changes are merged with the running configuration.
   - Applying the changes means reloading the *Core* services with the new configuration.
   - The new running configuration becomes visible to every user.

3.3.5. Connection handling example

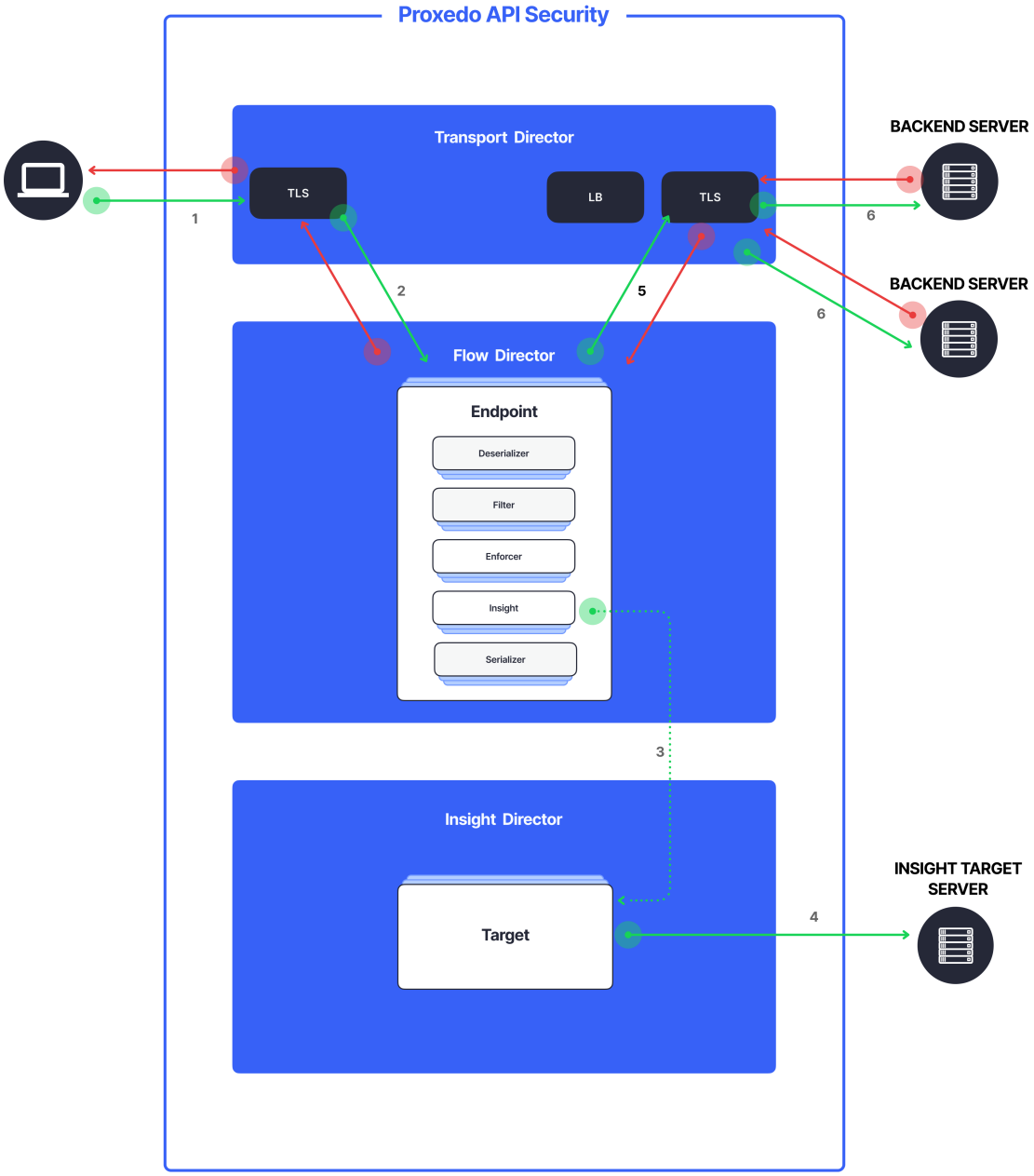


Figure 1. PAS Architecture

1. Incoming connections are accepted by the *Transport Director*.
    - It handles TLS with the client if necessary.
    - Chooses the *Endpoint* based on the URL.
2. It hands over the connection to the *Flow Director*.
    - The *Flow Director* applies the *Endpoint* specific *Request Security Flow*.
3. If an *Insight plugin* needs to send data to an external *Insight Target* it sends the collected data to the *Insight Director*.

4. The *Insight Director* sends the data further to the *Insight Target* with the appropriate protocol.
5. If a *WAF Enforcer plugin* is present in the *Request Security Flow* it sends data to the *Content Filtering Director* and receives a verdict.
6. If a *Fraud Detector plugin* is present in the *Request Security Flow* it sends data to the external *Fraud API* and receives a score.
7. The *Flow Director* hands the connection back to the *Transport Director*.
8. The *Transport Director* then sends the data to the *Backend*.
    - It handles TLS with the backends if necessary.
    - It performs load balancing among *Backend* servers if necessary.

The same procedure is executed with the response coming from the *Backend*.

3.3.6. Understanding processing flow

The figure on Proxedo API Security architecture and the steps following that describe how client connection is handled. The following figure explains how calls are processed in more details:

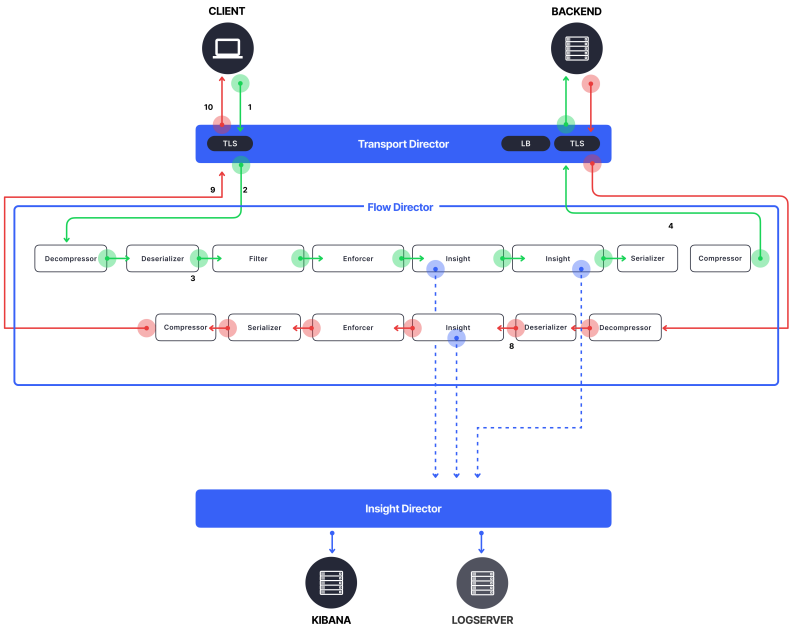


Figure 2. PAS processing flow

1. As shown in the figure above, the incoming connection from the client is handled by the *Transport Director*, applying TLS if needed.
2. The *Transport Director* then chooses the *Endpoint* based on the URL in the request. First endpoint that has a matching URL is chosen.
3. The *Transport Director* hands over the connection to the *Flow Director*, indicating which *Endpoint* the connection belongs to.
4. The *Flow Director* then starts applying the request part of the *Security Flow* definition.
5. For each *Plugin* the *Flow Director*:
    - Checks if the *Plugin*'s matcher matches the request.
    - If so, it executes the *Plugin*, if not, it executes the next *Plugin*.
    - If the *Plugin* indicates success it executes the next *Plugin*.
    - If the *Plugin* indicates an error it applies the *Plugin*'s error policy. If the policy dictates to abort the connection:

- It fills error details and hands back the connection to the *Transport Director*, aborting the execution of the flow.
- The *Transport Director* closes the connection, sending error details to the client if allowed by the policy.

6. Once the last *Plugin* has been executed the connection is handed back to the *Transport Director*.
7. The *Transport Director* initiates the connection towards the *Backend*:
   - It handles load balancing if necessary.
   - It handles TLS if necessary.
   - It sends the request itself to the *Backend* server.
8. The *Backend* server sends its response to the *Transport Director*.
9. Once, the response has been received the *Transport Director* again hands over the connection to the *Flow Director*.
10. The *Flow Director* then starts applying the response part of the *Security Flow* definition, executing the *Plugins* as above.
11. Once, the last *Plugin* has been executed the connection is handed back to the *Transport Director*.
12. Finally, the *Transport Director* sends the response to the client.

Usually, *Plugins* are organized in the following manner:

- A Decompressor *Plugin* extracts the compressed body.
- A Deserializer *Plugin* processes the decompressed request to understand the details in the body.
- Filters are applied to filter unnecessary traffic.
- Enforcers are applied for detailed validation of calls.
- Insights are applied to collect data from the call.
- Serializer *Plugin* serializes the body
- Compressor *Plugin* compresses the serialized body

Though the order of the plugins can be changed based on the needs, note the followings:

- When a Plugin needs access to the request body it requires Deserialized data. It is therefore strongly recommended that the first plugin is a Decompressor followed by a Deserializer.
- At the end of the flow it is strongly recommended to place a Serializer plugin followed by a Compressor.
- Generally Insights are applied after Filters and Enforcers so that they are not executed on possibly invalid calls.
- Anything that operates on the HTTP headers or the body of the message will be aware of the call direction: The same Plugin in the request and response flow will act on the request or response data.
- However, the *Flow Director* handles a request-response exchange together, so you can still use details from the request in Plugins of the response flow. The most notable example of this is using URI or method matchers in the response flow.
- *Plugins* in the request flow, however, cannot access details of the response flow (since they are not available yet).

It is also worth noting that *Insight Plugins* instantly hand over data to the *Insight Director*, and let the execution continue.

3.3.7. Architecture with High Availability

In case of HA operation, the core component is configured on both nodes participating in the HA operation. The architecture and the process are identical on both nodes, but they are set up to work redundantly. Only one node (the master) of the cluster is receiving traffic actively.

This operation uses the following additional resources:

- two nodes with PAS core installed

> At the moment, only clusters of two are supported. It also means, that you will need to have a node with both management and core components installed, and a node with only core installed.

- a virtual IP address through which the service is supposed to be accessible

The technical foundation of our HA solution is the Virtual Router Redundancy Protocol (VRRP). Once the requirements are properly set up, it operates as follows:

- At startup, the nodes send keepalive messages to each other to see if the other node is available. Both of them consider themselves to be in *BACKUP* state until they make sure the other node is not in *MASTER* state. If both nodes are newly connected to the cluster, they participate in an election and the one with the higher priority becomes the *MASTER*.
- After one of the nodes has become the *MASTER*, both of them keep sending keepalive messages so that they notice when the other node disappears.
- A node (re)connecting to the cluster does not result in the reelection of the *MASTER*.

4. Installation of Proxedo API Security based on VMs

PAS is mainly distributed as Docker images, and is also completed with a *.deb* package that sets up the operational environment.

You can install the management and the core components either on one single node in Standalone setup or on two separate ones in Multi node setup using the automated deployment tool. The synchronization of the core configuration is provided by the storage component.

The specific sections on the installation of the different components provide details on how to install them on a node. The installation order of the components is described in section Installation scenarios.

For the multi node setup, we provide a core deployment tool along with the management component. Using that, you can deploy and configure the core and the storage components on the remote node. The main starting points of its usage are described in section Multi node setup using the automated deployment tool.

4.1. Prerequisites for installing PAS

The followings are needed prior to the installation of Proxedo API Security:

- the license for PAS
- a technical user for accessing Balasys' download site and docker registry
- the PAS storage, core and management *.deb* packages

> You can download the *.deb* packages from the Balasys Download website.

- one or two servers with *Ubuntu 24.04* Operating System installed. See Installation scenarios.

> Make sure, that there is no user or group named "pas" in the environment where Proxedo API Security is planned to be installed.

4.1.1. Minimum system requirements

Each server of the PAS installation must meet the following minimum system requirements:

Table 1. Minimum system requirements

Operating system	Ubuntu 24.04 LTS
CPU cores	4
Memory	4 GB
Disk	40 GB
Network	1 interface, 1 Gbps

This minimum configuration can run a maximum of two *Flow Director* instances on servers with the core component installed.

4.2. Installation scenarios

4.2.1. Standalone setup

For a standalone setup one server is needed.

There are two major installation methods available for the standalone setup:

- **Simplified installation** - This simplified installation method directs the user with prompt windows throughout the installation process, offers default values.

Follow the instructions for simplified installation for standalone setup:

1. Simplified installation for the storage component
2. Simplified installation for the management component
3. Simplified installation for the core component

The simplified installation method is only available for the standalone setup.

- **Manual installation** - For manual installation in a standalone setup, all of the components must be installed on the same node in the following order:
    1. Storage component
    2. Management component
    3. Core component

4.2.2. Multi node setup

For a multi node setup two servers are needed, one server for the PAS core and another server for the management component.

In a multi node setup there will be a management node and another node with the core component. The storage component must be installed and configured on both nodes. The installation must be done in the following order:

1. Management node

a. Storage component
b. Management component

2. Core node

   The core node can be set up with the automation tool or manually installing the components in the following order:

   a. Storage component
   b. Core component

4.2.3. Multi node HA setup

Multi node HA setup is similar to the standard multi node setup, but in this scenario the management node will run the core component, as well. The installation order is the following:

1. Management node
   a. Storage component
   b. Management component
   c. Core component
2. Core node

   Core node can be set up with the automation tool or manually installing the components in the following order:

   a. Storage component
   b. Core component

The HA component is included in the core component, and it must be configured after the installation. See High availability configuration.

4.3. Simplified installation method for the standalone setup

This simplified installation method is only available for the standalone setup. By choosing this installation method, the user is directed through the installation with the help of prompt windows and suggested default values.

4.3.1. Simplified installation for the storage component

1. Log in as *root*.
2. Update the OS' package list: `apt update`.
3. Install the PAS storage *.deb* package: `apt install <path/to/deb>/proxedo-api-security-storage_<version>.deb`.

   This will:

   - Create a user named *pas* for running and configuring PAS, if it has not been created yet by the installation of other components previously.

     pas user must not be created manually beforehand.

   - Install the necessary configuration files and helper scripts under `/opt/balasys`.

- Create *systemd* services for managing the PAS storage component.

You need to use *apt* to install the *.deb* package locally as it installs its dependencies as well. *dpkg* will not resolve dependencies, and `apt-get` cannot install from a local file. Also note that to install PAS from the current directory, you must use the path `./` before the *.deb* package, or *apt* will try to download the package from a repository.

4. Define the registry you want to download the docker images from. Define it in the following format: `example.com`.

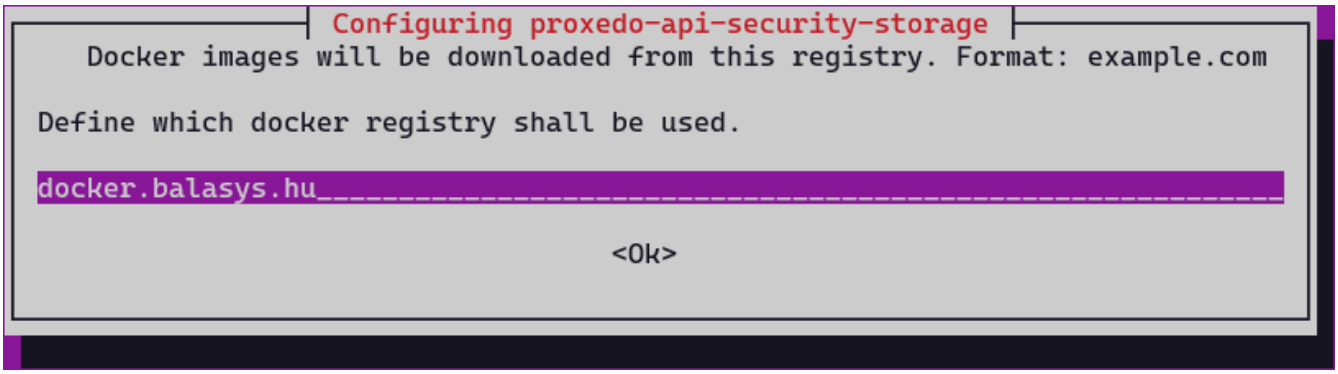


Figure 3. Registry to download the docker image from

5. Provide your user name for this registry.

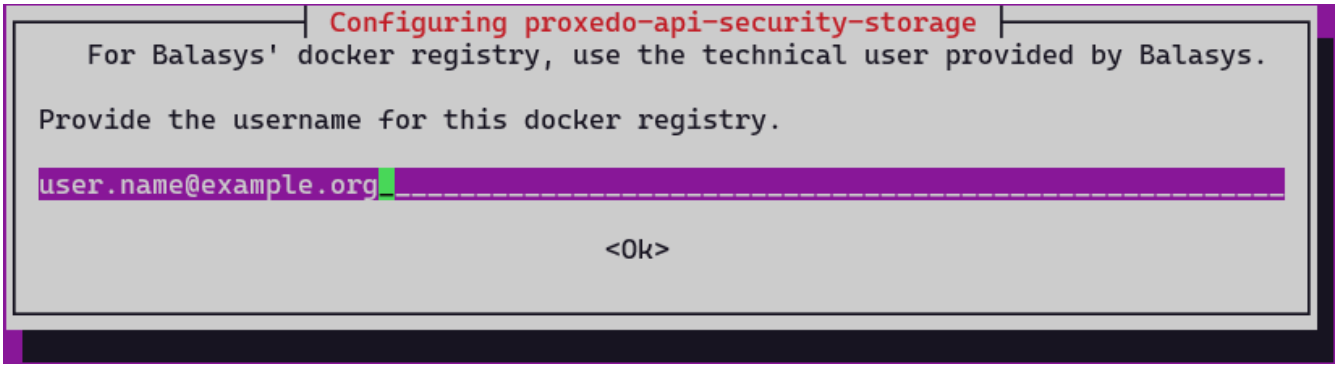


Figure 4. User name for the docker registry

6. Define the password for the docker registry.

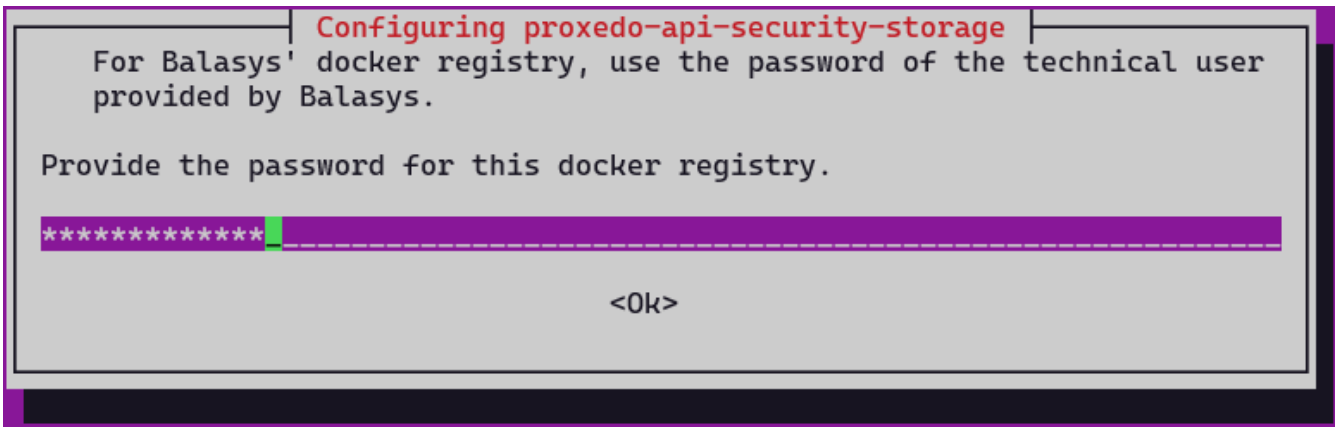


Figure 5. Password for the docker registry

7. Name the node.

This name will appear in the *Status dashboard*. As this value is optional, you can leave it blank to skip naming the

node.

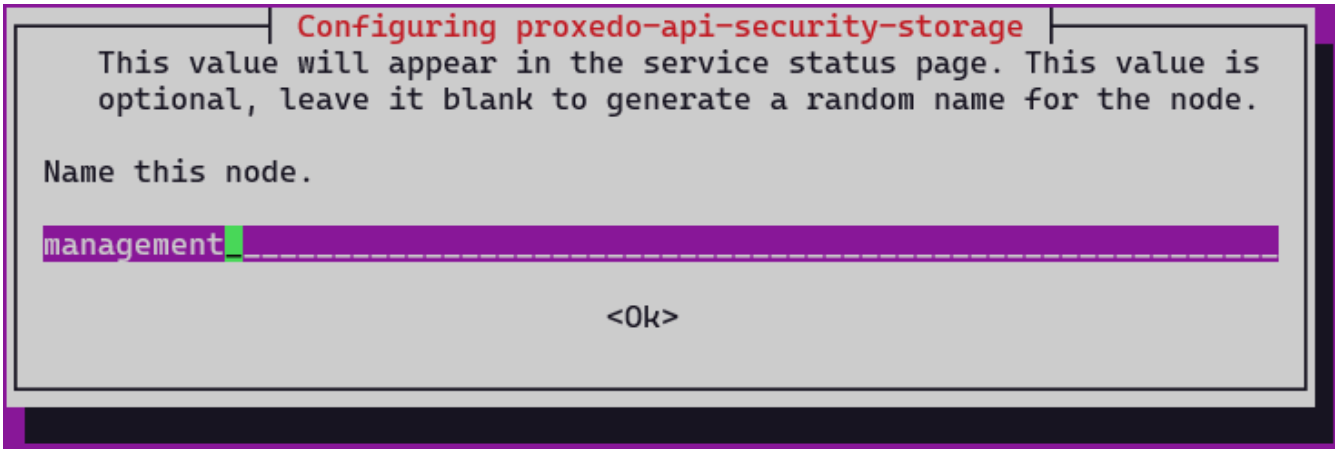


Figure 6. Naming the node

8. Generate a TLS certificate to secure the storage access. Define the identifier to be used in the storage TLS certificate.

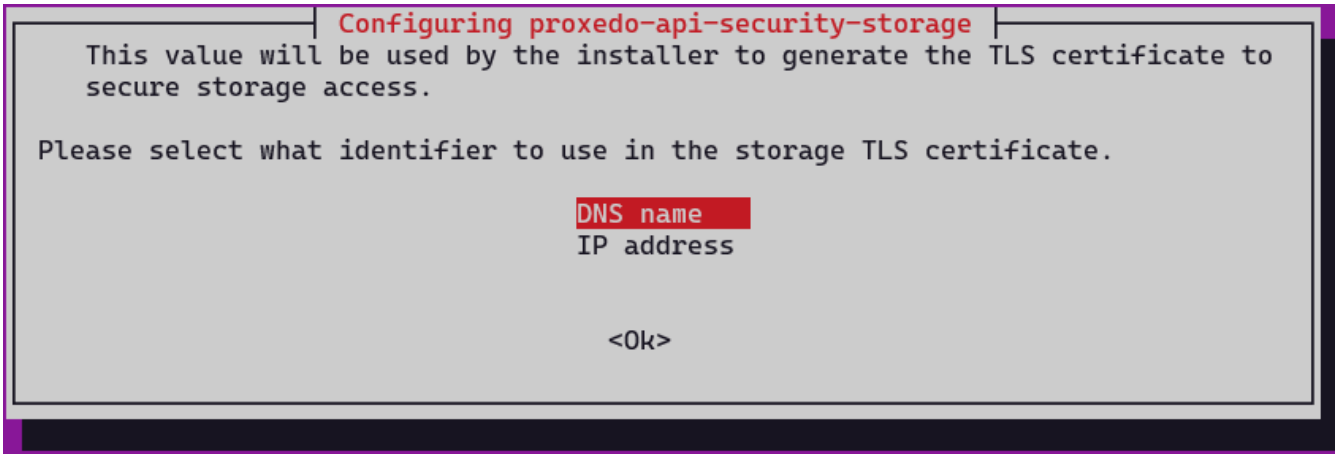


Figure 7. Identifier for TLS certificate

9. Provide a valid DNS name for the storage's TLS certificate.

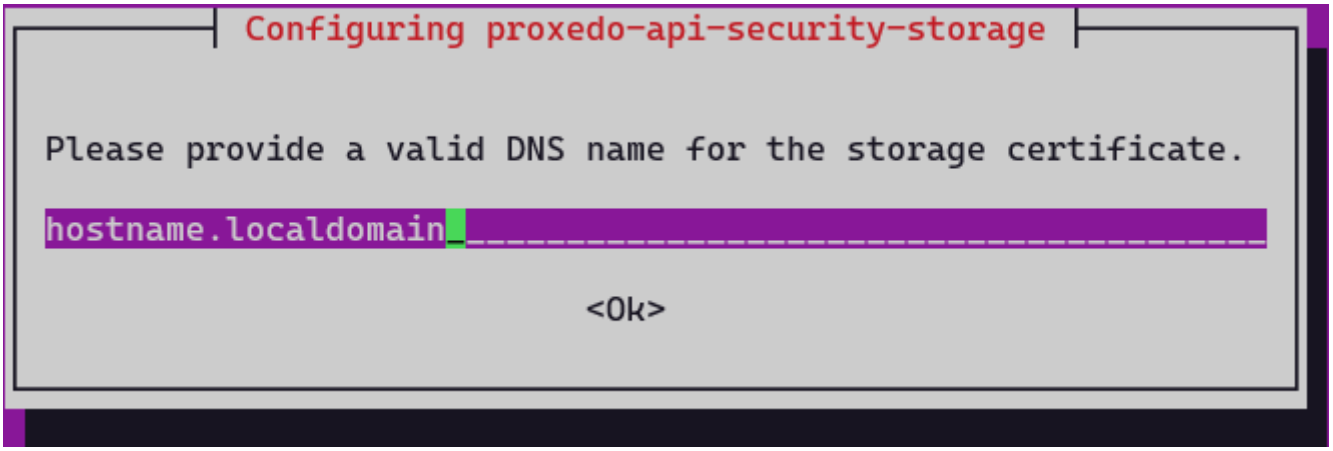


Figure 8. DNS name of the storage's TLS certificate

10. Define if the existing configuration files need to be overwritten or not.

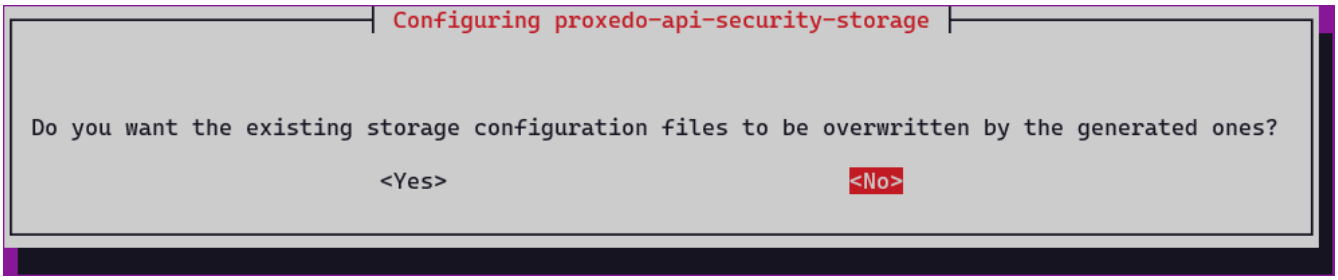


Figure 9. Confirming the need of existing configuration details

If the login to the docker registry is not successful, the following warning is displayed:

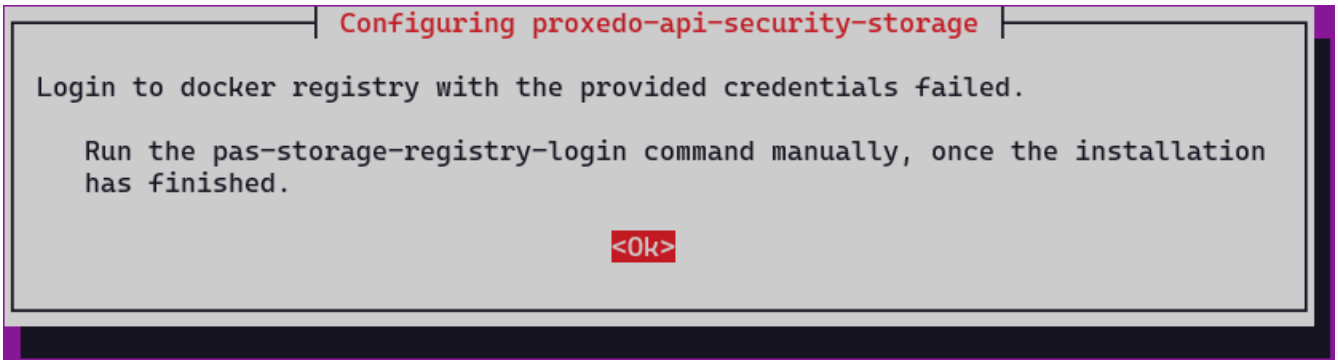


Figure 10. Docker login error warning

The installation can continue, but a Consul docker image is required for generating the secrets for the storage component. If it is not present, and the docker registry is not accessible, the following warning is displayed:

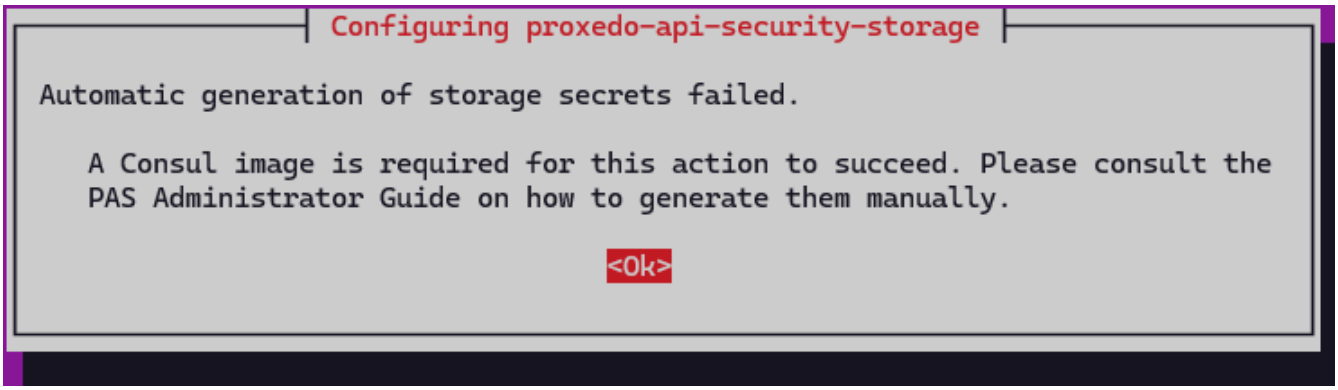


Figure 11. Automatic generation of storage secrets failed

11. Change to the PAS user: `su - pas`.
    a. Run `pas-storage-update` to download the docker images.
    b. Start PAS storage: `systemctl start proxedo-api-security-storage`.

This service is enabled by default, so the service starts on system restart.

4.3.2. Simplified installation for the management component

1. Log in as *root*.
2. Update the OS' package list: `apt update`.
3. Install the PAS storage *.deb* package: `apt install <path/to/deb>/proxedo-api-security-mgmt_<version>.deb`.

This will:

- Create a user named *pas* for running and configuring PAS, if it has not been created yet by the installation of other components previously.

  > *pas* user must not be created manually beforehand.

- Install the necessary configuration files and helper scripts under `/opt/balasys`.
- Create *systemd* services for managing the PAS storage component.

  > You need to use *apt* to install the *.deb* package locally as it installs its dependencies as well. *dpkg* will not resolve dependencies, and `apt-get` cannot install from a local file. Also note that to install PAS from the current directory, you must use the path `./` before the *.deb* package, or *apt* will try to download the package from a repository.

4. Provide the number of the port where you want the PAS Web UI to be available at, over HTTP:

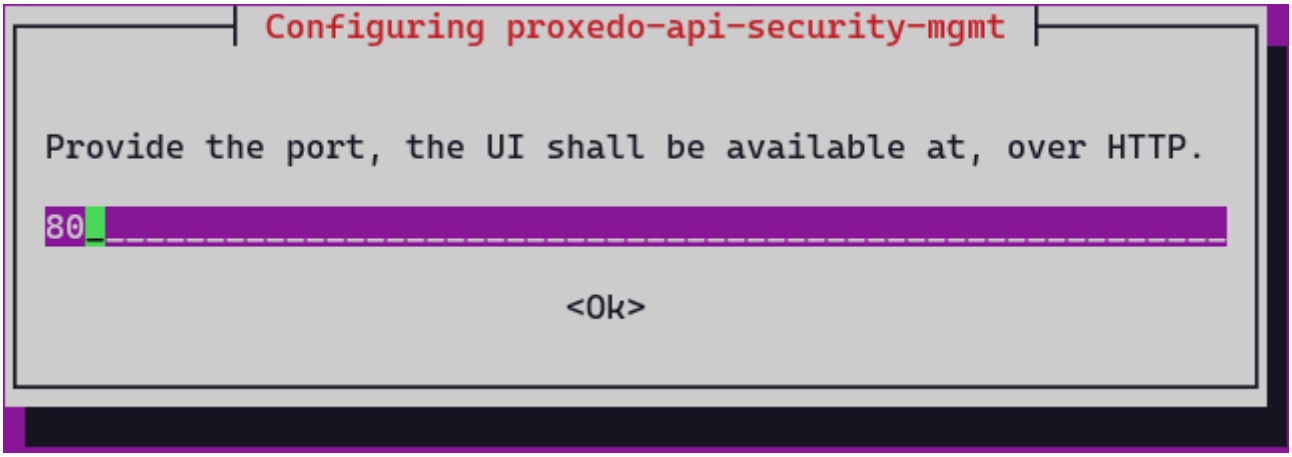


Figure 12. Defining port number for PAS Web UI over HTTP

5. Provide the number of the port where you want the PAS Web UI to be available at, over HTTPS:

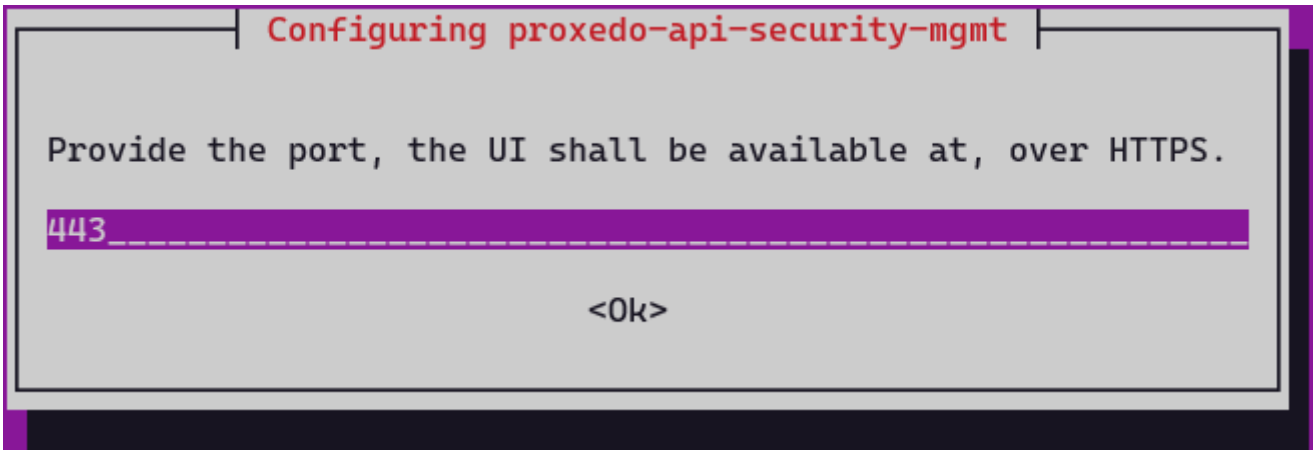


Figure 13. Defining port number for PAS Web UI over HTTPS

6. Define if the existing configuration files shall be overwritten by this generated configuration.

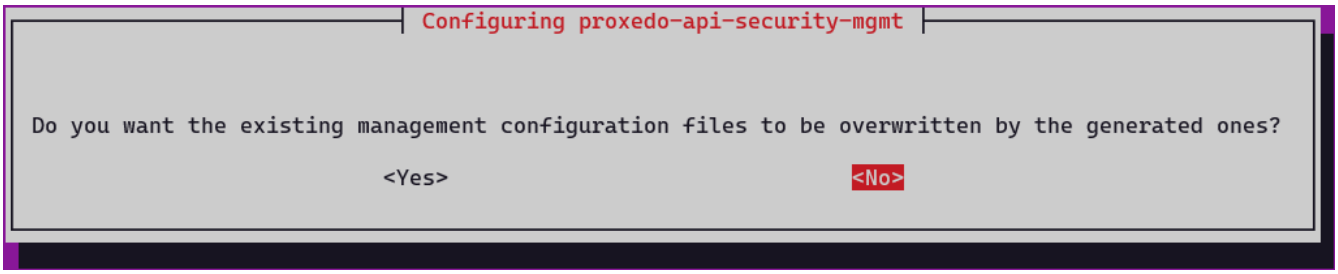


Figure 14. Defining if the existing configuration file shall be overwritten

7. Confirm if you want to manually specify an administrator password.

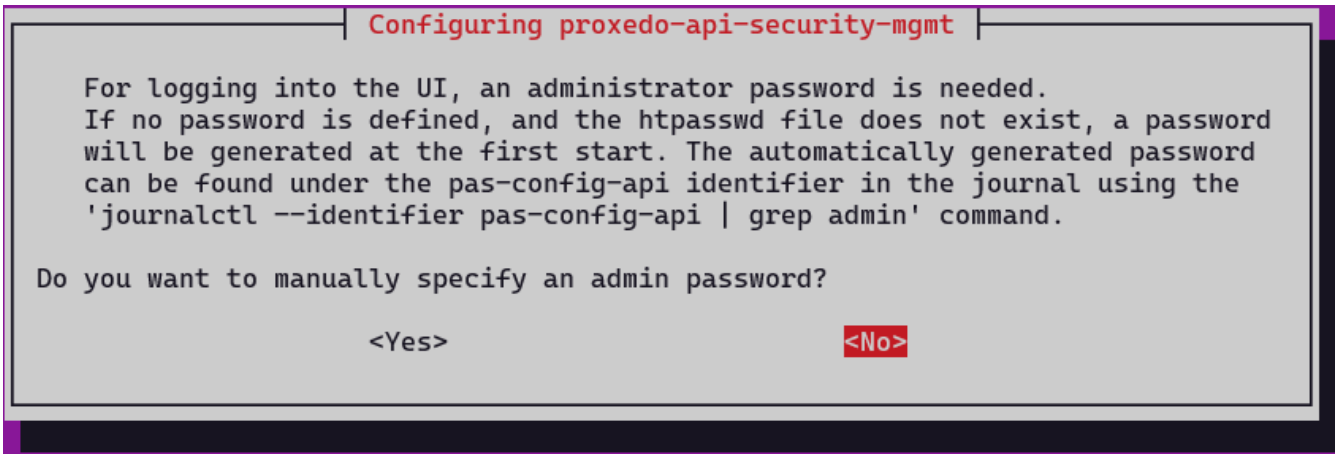


Figure 15. Confirming if manual administrator password is to be set

If the administrator selects 'No', that is, the administrator does not want to manually define the administrator password, a password is automatically generated and is shown on the next screen.

8. Define the administrator password for the management component.

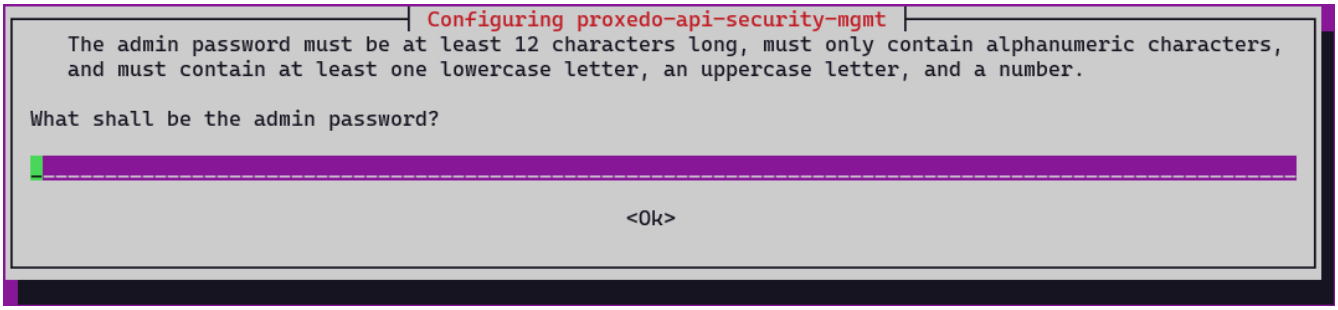


Figure 16. Defining password for the administrator

The following requirements must be met when defining the administrator's password. The password must contain:

- at least 12 characters
- only alphanumeric characters
- at least one lowercase character
- at least one uppercase character
- at least one number

9. Confirm your administrator password by reentering it.

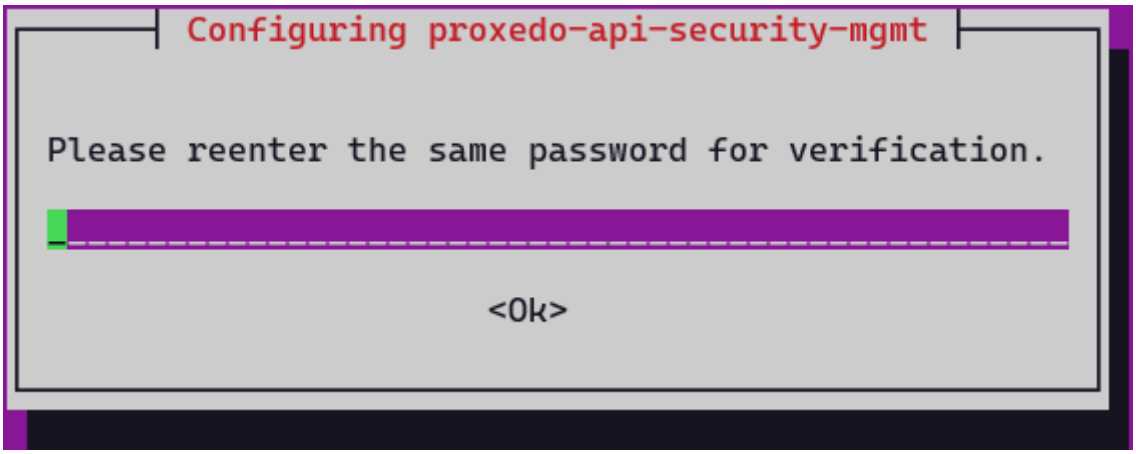


Figure 17. Confirming the administrator password

If the administrator password does not meet the requirements, an error window appears with the information that the password verification failed.

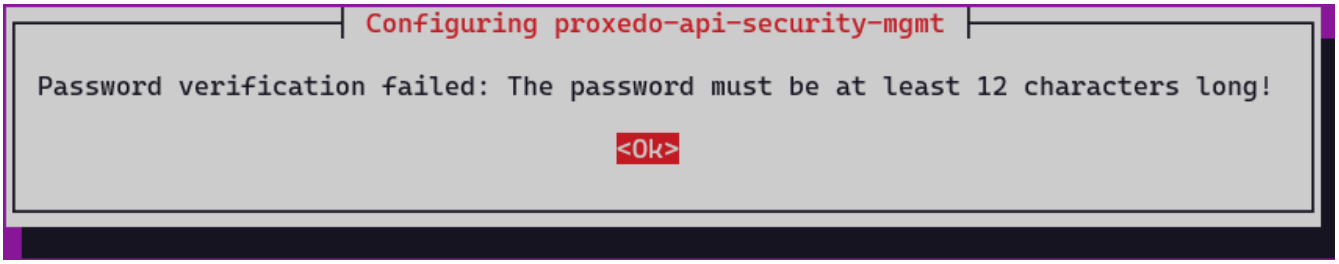


Figure 18. Password verification failed

10. Change to the PAS user: `su - pas`.
    a. Run `pas-mgmt-update` to download the docker images.
    b. Run systemctl start `proxedo-api-security-mgmt`.

4.3.3. Simplified installation for the core component

1. Log in as *root*.
2. Update the OS' package list: `apt update`.
3. Install the PAS storage *.deb* package: `apt install <path/to/deb>/proxedo-api-security_<version>.deb`.

   This will:

   - Create a user named *pas* for running and configuring PAS, if it has not been created yet by the installation of other components previously.

     pas user must not be created manually beforehand.

   - Install the necessary configuration files and helper scripts under `/opt/balasys`.
   - Create *systemd* services for managing the PAS storage component.

     You need to use *apt* to install the *.deb* package locally as it installs its dependencies as well. *dpkg* will not resolve dependencies, and `apt-get` cannot install from a local file. Also note that to install PAS from the current directory, you must use the path `./` before the *.deb* package, or *apt* will try to download the package from a repository.

4. Provide the number of Flow Director instances to run:

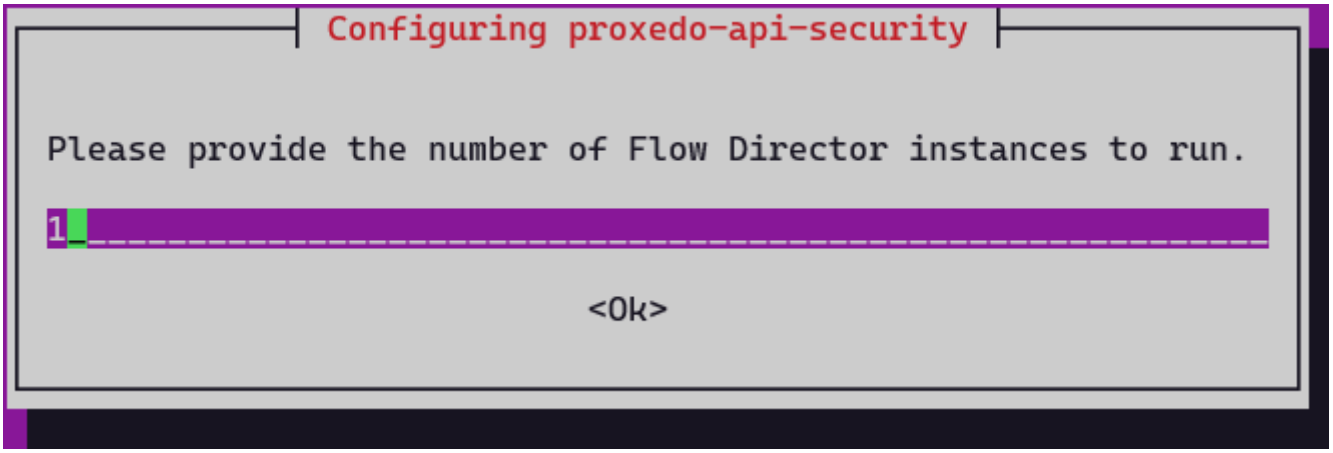


Figure 19. Defining the number of Flow Director instances

5. Provide the lower bound for the Transport Director's first port range:

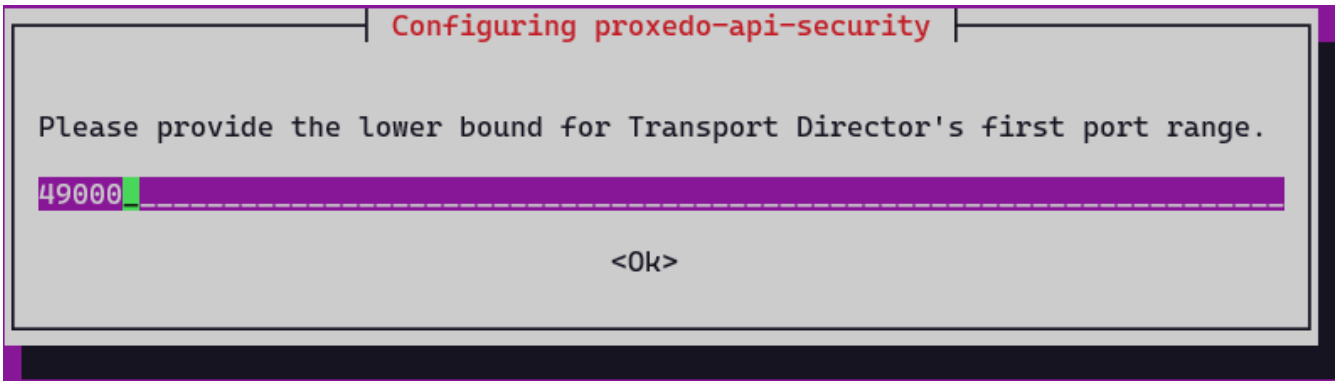


Figure 20. Defining the lower bound for the Transport Director's first port range

Defining a port range larger than the default 10+10 is possible, but might slow down the starting and stopping of services.

6. Provide the upper bound for the Transport Director's first port range:

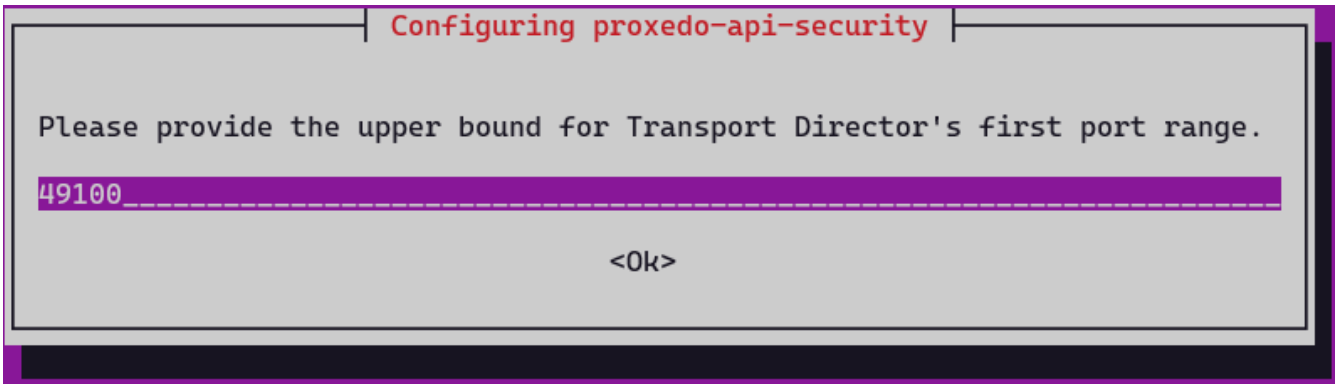


Figure 21. Defining the upper bound for the Transport Director's first port range

7. Provide the lower bound for the Transport Director's second port range:

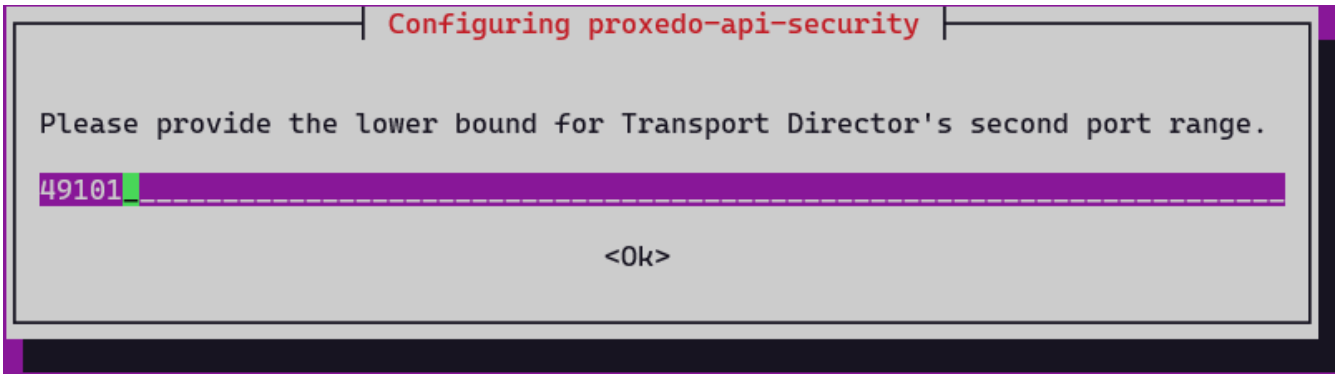


Figure 22. Defining the lower bound for the Transport Director's second port range

8. Provide the upper bound for the Transport Director's second port range:

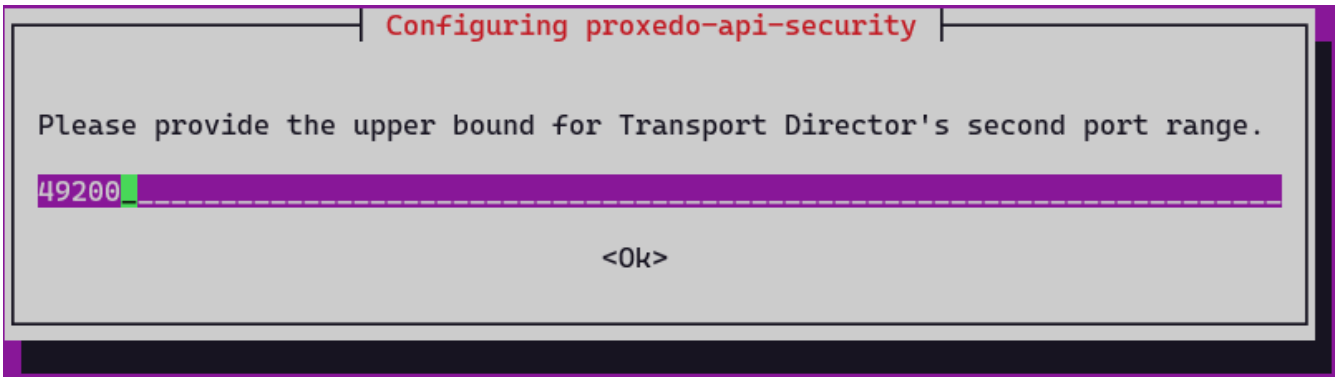


Figure 23. Defining the upper bound for the Transport Director's second port range

9. Define if the existing configuration files need to be overwritten or not.

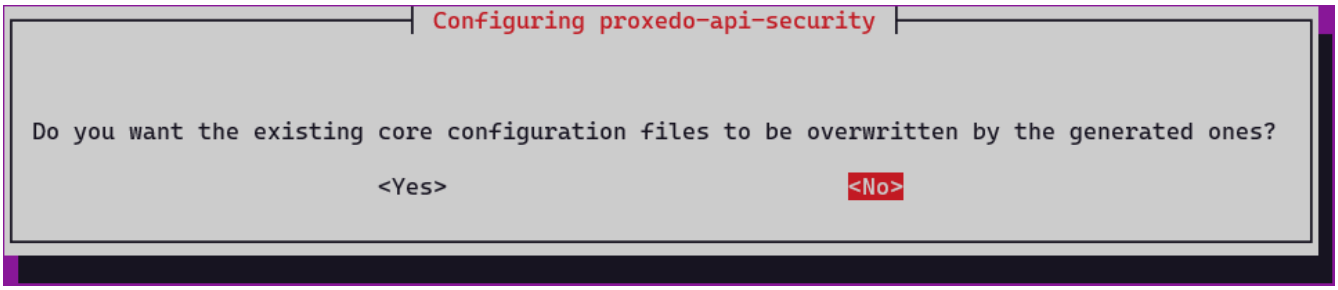


Figure 24. Confirming the need of existing configuration details

10. Change to the PAS user: `su - pas`.
    a. Run `pas-update` to download the docker images.
    b. Run systemctl start `proxedo-api-security`.

Note that as a registry login takes place during the installation of the core storage component, if the registry login fails, pas-update will also fail.

4.4. Installation steps for the storage component

1. Log in as *root*.
2. Update the OS' package list: `apt update`.
3. Install the PAS storage *.deb* package: `apt install <path/to/deb>/proxedo-api-security-storage_4.13.0_all.deb`.

This will:

- Create a user named *pas* for running and configuring PAS, if it has not been created yet by the installation of other components previously.

  > *pas* user must not be created manually beforehand.

- Install the necessary configuration files and helper scripts under `/opt/balasys`.
- Create *systemd* services for managing the PAS storage component.

  > You need to use *apt* to install the *.deb* package locally as it installs its dependencies as well. *dpkg* will not resolve dependencies, and `apt-get` cannot install from a local file. Also note that to install PAS from the current directory, you must use the path `./` before the *.deb* package, or *apt* will try to download the package from a repository.

4. Change to the PAS user: `su - pas`.
5. Set up parameters in `/opt/balasys/etc/infrastructure/storage/docker-compose.conf`. For details, see docker-compose.conf.
6. Run `pas-storage-registry-login` to set up authentication with the docker registry. Provide login credentials on the prompt. Contact support if you need assistance with your credentials.

   > Docker will, by default, save your credentials unencrypted in the home directory of the `pas` user. Using a password-management tool like `pass` is not enforced, but it is recommended.

7. Run `pas-storage-update` to download the docker images.
8. Run `pas-storage-consul-gossip-keygen` to generate gossip encryption key, and note it down. This key will be needed for the startup configuration. See Configuration options for the storage component.

   > This must be run only once, and the generated key must be used in the configuration of all storage components.

9. Run `pas-storage-consul-bootstrap-ca` to generate CA certificate for storage-storage communication.

   > This must be run only once on the management node. Do not execute it during the manual installation of the core node.

10. Run `pas-storage-consul-gen-server-cert` with *--dns-name FQDN* or with *--ip-address IP* parameter to generate the TLS certificate for storage-storage communication.

    FQDN is the domain name, *IP* is the IP address of the management or the core node.

    > There is no difference between using DNS or IP in the certificate, choose the one you prefer. Note, that in multi node setup the same DNS or IP address must be used in join_hosts parameter of the storage component's configuration, according to your choice.

This must be run on the management node, even in the case of installing the core node in a multi node setup.

11. Set up startup configuration in `/opt/balasys/etc/storage/config.yml`.

    For details, see Configuration options for the storage component.

12. Run `pas-storage-checkconfig` to validate the configuration.
13. Start PAS storage: `systemctl start proxedo-api-security-storage`.

This service is enabled by default, so the service starts on system restart.

4.5. Installation steps for the management component

1. Log in as *root*.
2. Update the OS' package list: `apt update`.
3. Install the PAS management *.deb* package: `apt install <path/to/deb>/proxedo-api-security-mgmt_4.13.0_all.deb`.

   This will:

   - Create a user named *pas* for running and configuring PAS, if it has not been created yet by the installation of other components previously.

     pas user must not be created manually beforehand.

   - Install the necessary configuration files and helper scripts under `/opt/balasys`.
   - Create *systemd* services for managing the PAS management component.

     You need to use *apt* to locally install the *.deb* package as it installs its dependencies as well. *dpkg* will not resolve dependencies, and `apt-get` cannot install from a local file. Also note that to install PAS from the current directory, you must use the path `./` before the *.deb* package, or *apt* will try to download the package from a repository.

4. Change to the PAS user: `su - pas`.
5. Set up `S3_*` parameters in `/opt/balasys/etc/infrastructure/mgmt/docker-compose.conf`. You can also modify other parameters if necessary, including port numbers and the version. For details, see docker-compose.conf.

S3_* parameters must be the same as defined in the config.yml of the storage component.

6. Run `pas-mgmt-registry-login` to set up authentication with the docker registry. Provide login credentials on the prompt. Contact support if you need assistance with your credentials.

Docker will, by default, save your credentials unencrypted in the home directory of the `pas` user. Using a password-management tool like `pass` is not enforced, but it is

recommended.

The following requirements must be met when defining the administrator's password. The password must contain:

- at least 12 characters
- only alphanumeric characters
- at least one lowercase character
- at least one uppercase character
- at least one number

7. Run `pas-mgmt-update` to download the docker images.
8. Set up startup configuration in **`/opt/balasys/etc/mgmt/config.yml`**.

   For details, see Configuration options for the management component

9. Run `pas-mgmt-checkconfig` to validate the configuration.
10. Start PAS management: `systemctl start proxedo-api-security-mgmt`.

This service is enabled by default, so the service starts on system restart.

4.6. Installation steps for the core component

1. Log in as *root*.
2. Update the OS' package list: `apt update`.
3. Install the PAS *.deb* package: `apt install <path/to/deb>/proxedo-api-security_4.13.0_all.deb`.

   This will:

   - Create a user named *pas* for running and configuring PAS, if it has not been created yet by the installation of other components previously.

pas user must not be created manually beforehand.

   - Install the necessary configuration files and helper scripts under **`/opt/balasys`**.
   - Create *systemd* services for managing PAS.

You need to use *apt* to locally install the *.deb* package as it installs its dependencies as well. *dpkg* will not resolve dependencies, and `apt-get` cannot install from a local file. Also note that to install PAS from the current directory, you must use the path `./` before the *.deb* package, or *apt* will try to download the package from a repository.

4. Change to the PAS user: `su - pas`.
5. Set up `S3_*` parameters and the number of *Flow Director* instances to run in **`/opt/balasys/etc/infrastructure/pas/docker-compose.conf`**. If necessary, also change the version you want to follow. For details, see docker-compose.conf.
6. Run `pas-registry-login` to set up authentication with the docker registry. Provide login credentials on the prompt. Contact support if you need assistance with your credentials.

Docker will, by default, save your credentials unencrypted in the home directory of the `pas` user. Using a password-management tool like `pass` is not enforced, but it is recommended.

7. Run `pas-update` to download the docker images.
8. Start PAS: `systemctl start proxedo-api-security`.

The core component will wait for a running config before starting. To create a running config, please refer to Configuration of Proxedo API Security on the Web User Interface.

This service is enabled by default, so the service starts on system restart.

9. If you configured Certificate Revocation List (CRL) verification in any of your Backends or Listeners you need to enable CRL updates:

```
systemctl enable proxedo-api-security-crl-update.service
systemctl enable proxedo-api-security-crl-update.timer
systemctl start proxedo-api-security-crl-update.timer
```

4.7. Multi node setup using the automated deployment tool

For a multi node setup, you first need to have a functional management component installed on a node. Its storage needs to be set up to work in cluster with the node the core component is going to be deployed to. Along with the management component, you get the automated deployment tool that helps you manage your remote node and the core component on it.

4.7.1. Configuring multi-node setup

To deploy a core component on a remote node, the following prerequisites need to be met:

- the core and the storage *.deb* packages are available on the management node
- a node with an Ubuntu is installed
- a TLS certificate for the storage-storage communication is generated, see step 10 in Installation steps for the storage component
- a user on the remote node is configured who can run `sudo` without providing password

As running `sudo` without a password grants virtually limitless privileges over the machine, it is strongly advised that SSH is only allowed using SSH keys.

To configure the automation and the remote nodes, you need to fill out three types of configuration files. However, before filling out the `/opt/balasys/etc/automation/common_vars.yml` file, make sure that the password to the docker registry is encrypted using Ansible Vault.

Example output for encryption

```
$ ansible-vault encrypt_string
New Vault password:
Confirm New Vault password:
```

```
Reading plaintext input from stdin. (ctrl-d to end input)
my_docker_password!vault |
          $ANSIBLE_VAULT;1.1;AES256
62623166386564303866653766656133616463633035643134313062383634336363633836336661
65663233313066356130346530623961663162623835356 60a32343366326166343532396263343 0
32636236393966643636636534626466626166366337303936386339663335653739306661303731
61626337323662346 30a37336434353637633638303566616538353331353065346365316231646 1
65323731633135613330343334663231316135343464373738383962303165393236
Encryption successful
```

The part displayed in the following example can be used as the encrypted docker registry password in the `/opt/balasys/etc/automation/common_vars.yml` file.

Example output for encrypted password

```
!vault |
          $ANSIBLE_VAULT;1.1;AES256
62623166386564303866653766656133616463633035643134313062383634336363633836336661
65663233313066356130346530623961663162623835356 60a32343366326166343532396263343 0
32636236393966643636636534626466626166366337303936386339663335653739306661303731
61626337323662346 30a37336434353637633638303566616538353331353065346365316231646 1
65323731633135613330343334663231316135343464373738383962303165393236
```

`pas-mgmt-deploy-core` will request the Ansible Vault password that was used to encrypt the password for the docker registry. For more details, see Deployment and remote management commands.

Fill out the following three types of configuration files to configure the automation and the remote nodes:

- `/opt/balasys/etc/automation/inventory.yml`: It provides the details of the nodes to deploy the core component to.
- `/opt/balasys/etc/automation/common_vars.yml`: It defines the variables that are common among all nodes.
- `/opt/balasys/etc/automation/host_vars`: It defines the directory holding variables for host-specific values. For each entry in `inventory.yml`, there must be a file in this directory. For example, if you have an entry named `pas-node-1` in your inventory, you need to have a file named `pas-node-1.yml` in the `host_vars` directory.

The last step to do before deploying the core component on a remote node is to implement login. Complete the following steps:

1. Generate an SSH key.

   ```
   pas@pas-node-mgmt:~$ ssh-keygen -t rsa -b 4096 -C "pas-node-1"
   ```

2. Add the SSH key to the user on the remote node which can run `sudo` without password.

   ```
   deployment@pas-node-1:~$ mkdir -p ~/.ssh
   deployment@pas-node-1:~$ cat <<generated_public_key_here>> >> ~/.ssh/authorized_keys
   deployment@pas-node-1:~$ chmod 700 ~/.ssh/
   deployment@pas-node-1:~$ chmod 600 ~/.ssh/authorized_keys
   ```

4.7.2. Deployment and remote management commands

To deploy core and manage it on the remote node, you need to run the `pas-mgmt-deploy-core` command. Running this command with different command line flags, you can execute different operations. See the list and explanation of possible operations as follows, or run `pas-mgmt-deploy-core --help`.

```
--deploy-core     Deploy the core component to all nodes. It will also be
                  started if not already running.
--restart-core    Restart the core component on all nodes. It will restart
                  ha-director as well.
--stop-core       Stop the core component on all nodes. It will stop
                  ha-director as well.
--deploy-ha       Deploy the HA component to all nodes. It will also be
                  started if not already running.
--restart-ha      Restart ha-director on all nodes.
--stop-ha         Stop ha-director on all nodes.
--sync-ntp        Copy NTP configuration to all nodes and restart the ntp
                  service.
```

5. Base system configuration for PAS based on VMs

This chapter explains configuration details for setting up a working PAS. Configuration settings are detailed here, provided by the *.deb* packages installed on an *Ubuntu 24.04 LTS* server.

The *.deb* packages carry convenience tools for managing the Proxedo API Security core and the management component, the actual installation and execution is done by Docker and docker compose.

5.1. Overview of configuration directories

PAS consists of multiple components and each may have multiple configuration files for both infrastructure definition and bootstrapping. In this section, a general overview is provided about how the corresponding configuration directories are structured.

As a rule of thumb, every configuration file for PAS is available under `/opt/balasys/etc`, and right after installation of all the components, the directory tree looks like the following.

Tree of initial configuration files

```
/opt/balasys/etc/
├── automation
│   ├── common_vars.yml -> ../../usr/share/automation/roles/deploy-core/vars/main.yml
│   ├── host_vars -> ../../usr/share/automation/host_vars
│   └── inventory.yml -> ../../usr/share/automation/inventory.yml
├── ha
│   └── config.yml
├── infrastructure
│   ├── ha
│   │   └── docker-compose.yml
│   ├── mgmt
│   │   ├── docker-compose.conf
│   │   └── docker-compose.yml
│   ├── pas
│   │   ├── docker-compose.conf
│   │   └── docker-compose.yml
│   └── storage
│       ├── docker-compose.conf
│       └── docker-compose.yml
├── mgmt
│   └── config.yml
```

```
├── pas
└── storage
    └── config.yml
```

In general, the following rules apply:

- `/opt/balasys/etc/infrastructure` holds the files to describe the infrastructure, the component will run in. Exposed ports and used images are the two most important parameters set up in the files in this directory.
- `/opt/balasys/etc/{ha,mgmt,pas,storage}` hold the bootstrap configuration of the respective components. By default, only configuration files are created, but for example, certificates also reside in these directories. They will be mounted to the containers so that the processes can use these files as well.
- `/opt/balasys/etc/automation` makes necessary configuration available for remote core deployment. As seen in the directory tree, files in this directory are symbolic links to ease access to the actual files. During debugging, or after installation, their original directories may also be useful to look at.

As there are more directories and subdirectories linked to each package, the following table defines the corresponding directories for each package.

Table 2. Directories grouped by package ownership

Package	Directories
proxedo-api-security	• `/opt/balasys/etc/ha` • `/opt/balasys/etc/pas` • `/opt/balasys/etc/infrastructure/ha` • `/opt/balasys/etc/infrastructure/pas`
proxedo-api-security-mgmt	• `/opt/balasys/etc/mgmt` • `/opt/balasys/etc/automation` • `/opt/balasys/etc/infrastructure/mgmt`
proxedo-api-security-storage	• `/opt/balasys/etc/storage` • `/opt/balasys/etc/infrastructure/storage`

5.2. config.yml

The main configuration of the storage, management and HA components is defined in the following files:

- `/opt/balasys/etc/storage/config.yml`
- `/opt/balasys/etc/mgmt/config.yml`
- `/opt/balasys/etc/ha/config.yml`

The format of the files must adhere to the YAML 1.1 specification.

There are different sections in these configuration files, some of which, as for example, the 'common' section, might not need specific configuration. However, the default values of these sections must be set by `{}`.

See configuration examples in Appendix B.

5.2.1. Configuration options for the storage component

The file `/opt/balasys/etc/storage/config.yml` controls:

- Standalone or Multi node setup

- Storage intercommunication
- Node name visible on Dashboard
- S3 keys

The configuration file has three main sections, namely **common**, **consul** and **blob-store**.

The 'common' section has no required parameters, the defaults can be set by `{}`.

Table 3. Storage configuration **common** *options*

Key	Default	Description
short_product_name	pas	This parameter must be set to 'pas'.
standalone_mode	true	It denotes whether the storage is run in standalone or in cluster mode.

Table 4. Storage configuration **consul** *options*

Key	Default	Description
bind_cluster_addr		The address to bind on as a cluster member. This will be used to communicate with other members. **This is a required parameter.**
gossip_encryption_key		The encryption key to use for the gossip protocol. It is a 32-byte shared key encoded into base64 format. Use `pas-storage-consul-gossip-keygen` to generate it. The same value must be used on different nodes in a multi node setup. **This is a required parameter.**
node_name		The name of the consul node. It must be unique in the cluster. This parameter sets the visible node name on the Status Dashboard. *This is a recommended parameter.*
log_level	INFO	The log level of consul. The possible values are: TRACE, DEBUG, INFO, WARN, ERR
server_tls	N/A	TLS settings for storage-storage communication. The certificates must be created using the `pas-storage-consul-bootstrap-ca` and `pas-storage-consul-gen-server-cert` scripts.
ca_path	consul-agent-ca.pem	The path of the CA file relative to `/opt/balasys/etc/storage/`.
cert_path	dc1-server-consul-0.pem	The path of the server cert file relative to `/opt/balasys/etc/storage/`.
key_path	dc1-server-consul-0-key.pem	The path of the server key file relative to `/opt/balasys/etc/storage/`.
join_hosts	N/A	The list of hosts to try for joining the cluster. Either the hostname or the IP address can be specified, and a port number is also necessary. **Specify it on all nodes of the cluster.**
hostname		The hostname of the storage node to join.
ip_address		The IP address of the storage node to join.
port	8301	The port used for the storage cluster. **Do not modify it unless the port is changed in `/opt/balasys/etc/infrastructure/storage/docker-compose.yml`.**

The options with 'N/A' default value are such sections that cannot have exact values, only the

values described afterwards in the table.

Table 5. Storage configuration **blob-store** *options*

Key	Default	Description
config.storage.blob_store.access_key		The access key used for connecting to the blob-store. It starts with `GK`, followed by 12 hex-encoded bytes. **This is a required parameter.**
config.storage.blob_store.secret_key		The secret key is a 32-byte hex-encoded random string, which can be generated with a command such as `openssl rand -hex 32`. **This is a required parameter.**
config.storage.blob_store.rpc_secret		The RPC secret is a 32-byte hex-encoded random string, which can be generated with a command such as `openssl rand -hex 32`. **This is a required parameter.**
config.storage.blob_store.admin_token		The token for accessing the API administration endpoints. Any string can be used for this value. A random token generated with `openssl rand -base64 32` is recommended. **This is a required parameter.**
join_hosts	N/A	A list of hosts to try for joining the cluster. Either of the hostname or the ip_address can be specified, and a port number is also necessary.
hostname		The hostname of the storage node to join.
ip_address		The IP address of storage nodes to join.
port	9000	The port used for the storage cluster. **Do not modify unless the port is changed in `/opt/balasys/etc/infrastructure/storage/docker-compose.yml`.**

The options with 'N/A' default value are such sections that cannot have exact values, only the values described afterwards in the table.

For configuration examples, see section Minimal storage configuration.

5.2.2. Configuration options for the management component

The `/opt/balasys/etc/mgmt/config.yml` file controls:

- Web service parameters
- Authentication
- TLS settings

The configuration file has two main sections, namely **frontend** and **configapi**.

The default values for both **frontend** and **configapi** sections are automatically effective. If the attributes have to be configured with specific values, other than the default values, the `{}` curly braces have to be deleted and the new values have to be added.

Table 6. Management configuration **frontend** *options*

Key	Default	Description
server_name	_	The hostname the web server should serve the requests on. The default value means that the management interface will be served regardless of the provided hostname.
tls	N/A	This section configures TLS settings.
certificate_path	`/tmp/tls/default.crt`	The path to the server certificate. It most likely resides somewhere under `/opt/balasys/etc/mgmt`. **The default path refers to an automatically generated certificate. It must not be trusted.**
key_path	`/tmp/tls/default.key`	The path to the server private key. It most likely resides somewhere under `/opt/balasys/etc/mgmt`. **The default path refers to an automatically generated key. It must not be trusted.**
hsts_max_age	63072000	The maximum age attribute of the strict transport security header.
cors_api	N/A	This section configures cross origin request sharing options for API access.
allow_origin*		The value of the Access-Control-Allow-Origin header. **This is a required parameter in case of enabled CORS API.**

The options with 'N/A' default value are such sections that cannot have exact values, only the values described afterwards in the table.

Table 7. Management configuration log level setting options - **configapi** *section*

Key	Default	Description
log_level	INFO	The log level can be set to DEBUG, INFO, WARNING, ERROR, CRITICAL.

Table 8. Management configuration user session options - **configapi** *section*

Key	Default	Description
session	N/A	This section configures the options for session lifetimes.
session_validity	600	The allowed lifetime of a login session token in seconds. It determines the time period between group membership and user existence checks. **This DOES NOT control the length of a user session.**
renew_validity	36000	The validity of the renew token. It determines for how long session tokens can be renewed. Therefore the maximum length of a user session is the sum of the two parameters.

The options with 'N/A' default value are such sections that cannot have exact values, only the values described afterwards in the table.

For further details on **configapi** section parameters related to LDAP authentication, see Management configuration LDAP authentication options - **configapi** section.

For configuration examples on the management component, see section Minimal management configuration and section Management configuration with HTTPS (TLS) and LDAP authentication.

5.2.2.1. Configuring authentication and local users in PAS

There are two methods available to configure authentication in PAS:

- *htpasswd* authentication
- Lightweight Directory Access Protocol (LDAP) authentication

An administrator password can be set or generated during installation.

Using *htpasswd* for authentication and for the configuration of local users

By using *htpasswd* authentication, the administrator can define individual user credentials directly in the *htpasswd* file. This file is stored at `/opt/balasys/etc/mgmt/users.htpass` and its location cannot be configured. As local users are stored in an *htpasswd* file, the standard *htpasswd* tool needs to be used.

It is not possible to configure user groups, or to define different access levels for the users with *htpasswd* authentication, yet it is possible to define as many user credentials as necessary one by one. The user credentials are encrypted in the configuration file using the bcrypt encryption method. If you want to add new users to the *htpasswd* file, run the `htpasswd /opt/balasys/etc/mgmt/users.htpass username` command and provide the password.

Example command and output

```
$ htpasswd -B /opt/balasys/etc/mgmt/users.htpass new-user
New password:
Re-type new password:
Adding password for user new-user
```

Consider the followings related to the command and the example output:

- `/opt/balasys/etc/mgmt/users.htpass` denotes the path of the *htpasswd* file.
- new-user is the name of the new user.

As a result, similar content is expected to appear in the referred file:

`new-user:$2y$05$jsvtfYMP1HJZlWCNGVk6d.j4yWU5gJ4D97Vr6z8yK9A2wy80g1iD`.

LDAP authentication

LDAP authentication is a more elaborate way to configure authentication for PAS. With LDAP authentication it is possible to define user groups and attach different levels of access to these users, however, PAS does not support different levels of authorization based on these attributes yet at the moment.

If LDAP authentication is used, only the administrator user - and no other user - can authenticate with the *htpasswd* file.

The following **configapi** parameters, which are part of the configuration file's **configapi** section, take part in LDAP authentication:

*Table 9. Management configuration LDAP authentication options - **configapi** section*

Key	Default	Description
ldap	N/A	This section configures the options for LDAP authentication. **LDAP authentication is disabled by default.**
ldap_url*		The URL of the LDAP server. It must start with `ldap[s]://`. **This is a required parameter in case of LDAP authentication.**

Key	Default	Description
bind_user*		The service user to use for searching the LDAP server. If the `use_ntlm` parameter is OFF, this must be the whole DN. If it is ON, it must be the Active Directory domain and the username concatenated by a backslash (eg. `AD_domain\administrator`). **This is a required parameter in case of LDAP authentication.**
bind_password*		The password of the service user. **This is a required parameter in case of LDAP authentication.**
use_ntlm	OFF	Set this parameter to ON to use NTLM authentication. This is only available when the LDAP server is Microsoft Active Directory.
tls_version	TLSv1_2	The TLS version for the LDAPS connection. It must be one of the following: SSLv23, TLS, TLS_CLIENT, TLS_SERVER, TLSv1, TLSv1_1, TLSv1_2, TLSv1_3.
validate_cert	no	Set it to yes to validate certificates.
ca_certs_file		This file contains the certificate files of the certificate authorities. Provide the path and filename for the certificate file. The certificate file must be in PEM format. See a single CA file configuration example in Single CA file example. In case a self-signed certificate is used, the server certificate must also be included in this file. In case a chain of certificates is used, the certificate of each level must be included in this file, beginning with the certificate of the signer of the server certificate, followed by the signer of that certificate up to the root certificate. For example on a Certificate chain with multiple CA, see Example on certificate chain with multiple CAs. In case multiple chains of certificates are used, the chains must be concatenated in the same file. The first matching chain will be used for verification. The above details are based on the Python SSL library documentation, available at https://docs.python.org/3.12/library/ssl.html#certificates.
user_base_dn*		The base DN under which users reside. **This is a required parameter in case of LDAP authentication.**
username_attribute	sAMAccountName	The attribute that contains the name of the user.
user_object_class	user	The object class of the users.
memberOf_attribute	memberOf	The attribute that contains membership information (groups) on user objects.
group_base_dn*		The base DN under which groups reside. **This is a required parameter in case of LDAP authentication.**
groupname_attribute	name	The attribute that contains the name of the group.
member_attribute	member	The attribute that contains membership information (users) on group objects.
group_object_class	group	The object class for groups.

Key	Default	Description
allowed_groups*		A list of group names (as contained by 'groupname_attribute') allowed to log in. **This is a required parameter in case of LDAP authentication.**

5.2.3. Configuration options for the HA component

The file `/opt/balasys/etc/ha/config.yml` controls:

- HA settings

The configuration file has a single section called **ha**, which does not have default values.

Table 10. HA configuration **ha** *options*

Key	Default	Description
interface		The network interface to use for VRRP communication.
virtual_router_id	1	The router ID to use in VRRP messages. Min: 1; Max: 255
priority		The VRRP priority of the node in the virtual router instance. Min: 1; Max: 254. One node needs to have a priority higher by 50 compared to all other nodes. For example: If the highest priority node has a priority of 150, all other nodes must have a priority of 100 or lower.
auth_pass		The authentication password to use in the VRRP protocol. It is an alphanumerical string of up to 8 characters.
virtual_ip		The virtual IP address to use for HA.

For a configuration example on HA, see section Minimal HA configuration.

5.3. docker-compose.yml

The main configuration of the running environment is defined in the following files:

- `/opt/balasys/etc/infrastructure/storage/docker-compose.yml`
- `/opt/balasys/etc/infrastructure/pas/docker-compose.yml`
- `/opt/balasys/etc/infrastructure/mgmt/docker-compose.yml`
- `/opt/balasys/etc/infrastructure/ha/docker-compose.yml`

They describe the containers running PAS.

The format of the files must adhere to the YAML 1.1 specification. For a brief overview of the YAML format look at the example here. For an in-depth reference of docker compose configuration see its documentation.

This file controls:

- the images to run the container from
- the persistent data storage (docker volumes) to attach to the containers
- the ports propagated to the containers
- the environment variables available inside the containers
- intra container communication channels (links)
- log *Insight Target* configuration

Unless inevitable, these files shall *not* be modified. There are two cases when they might need to be modified:

1. If the provided environment needs to be modified for the setup to be debugged.
2. If the default behavior of logging into the system's journal needs to be changed, change the **logging** parameters under all the services. See more details in docker compose's documentation.

If you modify these files, they will not be overwritten on package upgrade. Only interactive installations will notify on that.

Do not use docker compose directly to manage the installation. Always use *systemctl* as it handles dependencies and scaling.

5.4. docker-compose.conf

Some aspects of how the services are run by *docker compose* are configured through `/opt/balasys/etc/infrastructure/storage/docker-compose.conf`, `/opt/balasys/etc/infrastructure/pas/docker-compose.conf` and `/opt/balasys/etc/infrastructure/mgmt/docker-compose.conf`.

The format of this file is a shell environment file format: a key-value pair in each line, separated by an equal sign ("=").

There must **not** be spaces around the equal sign.

The configuration files of different components have a common portion along with other parameters that are only valid in one of them. For details, see the following tables.

There are no storage-specific configuration options.

Table 11. docker-compose.conf configuration common options

Key	Default	Description
PAS_IMAGE_TAG	4.13.0	The release track of Proxedo API Security to use. See Tracking version.
COMPOSE_FILE	`/opt/balasys/etc/infrastructure/pas/docker-compose.yml` or `/opt/balasys/etc/infrastructure/mgmt/docker-compose.yml`	The path to the compose file. You *must* not modify the default value.
COMPOSE_PROJECT_NAME	pas	The name used for the compose project. It must be kept synchronized over different files.
PAS_DOCKER_REGISTRY	docker.balasys.hu	The domain name of the docker registry to download images from.

Table 12. docker-compose.conf configuration management-specific options

Key	Default	Description
PAS_MGMT_WEBUI_HTTP_PORT	80	The port to expose for HTTP access of the management web user interface.
PAS_MGMT_WEBUI_HTTPS_PORT	443	The port to expose for HTTPS access of the management web user interface.
S3_ACCESS_KEY		The access key to be used for the blob-store authentication. It must be the same as defined in `/opt/balasys/etc/storage/config.yml`.
S3_SECRET_KEY		The secret key to be used for the blob-store authentication. It must be the same as defined in `/opt/balasys/etc/storage/config.yml`.

Table 13. docker-compose.conf configuration core-specific options

Key	Default	Description
PAS_FLOW_DIRECTOR_SCALE	1	The number of *Flow Director* instances to run. For details, see Scaling Flow Director.
PAS_TRANSPORT_DIRECTOR_PORT_RANGE1	49000-49010	A port range to expose to *Transport Director*. Listeners will work in this port range. **NOTE:** defining a port range larger than the default 10+10 is possible, but might slow down the starting and stopping of services.
PAS_TRANSPORT_DIRECTOR_PORT_RANGE2	49011-49020	An additional port range to expose to *Transport Director*. Listeners will work in this port range.
S3_ACCESS_KEY		The access key to be used for the blob-store authentication. It must be the same as defined in `/opt/balasys/etc/storage/config.yml`.
S3_SECRET_KEY		The secret key to be used for the blob-store authentication. It must be the same as defined in `/opt/balasys/etc/storage/config.yml`.

Storage example:

```
PAS_IMAGE_TAG=4.13.0
COMPOSE_FILE=/opt/balasys/etc/infrastructure/storage/docker-compose.yml
COMPOSE_PROJECT_NAME=pas
PAS_DOCKER_REGISTRY=docker.balasys.hu
```

Management example:

```
PAS_IMAGE_TAG=4.13.0
COMPOSE_FILE=/opt/balasys/etc/infrastructure/mgmt/docker-compose.yml
COMPOSE_PROJECT_NAME=pas
PAS_DOCKER_REGISTRY=docker.balasys.hu

PAS_MGMT_WEBUI_HTTP_PORT=80
PAS_MGMT_WEBUI_HTTPS_PORT=443

S3_ACCESS_KEY=your_s3_access_key
S3_SECRET_KEY=your_s3_secret_key
```

Core example:

```
PAS_IMAGE_TAG=4.13.0
COMPOSE_FILE=/opt/balasys/etc/infrastructure/pas/docker-compose.yml
COMPOSE_PROJECT_NAME=pas
PAS_DOCKER_REGISTRY=docker.balasys.hu
PAS_FLOW_DIRECTOR_SCALE=1

PAS_TRANSPORT_DIRECTOR_PORT_RANGE1=49000-49010
PAS_TRANSPORT_DIRECTOR_PORT_RANGE2=49011-49020

S3_ACCESS_KEY=your_s3_access_key
S3_SECRET_KEY=your_s3_secret_key
```

Changing any of the values requires the restart of the service.

5.5. PAS restart policy

PAS service lifecycle is managed by systemd and is by default set to restart if any of the components fails at any point. To avoid infinite restarting, the number of restarts within a short period of time is also limited. As a result, if PAS core or management stops with a non-zero exit code 3 times within 100 seconds, the corresponding systemd unit will enter failed state.

The default restart policy and the options are identical for the storage, core and management components.

The relevant part of the service file looks as follows:

```
[Unit]
StartLimitIntervalSec=100
StartLimitBurst=3

[Service]
Restart=on-failure
```

Modifying the restart policy is possible by editing the service file in override mode. To do so, run `systemctl edit proxedo-api-security-storage`, `systemctl edit proxedo-api-security` or `systemctl edit proxedo-api-security-mgmt`. This will open a text editor and will let you define the parameters you wish to override. For example, if you want to switch off all default restart settings, enter the following text in the override editing window:

```
[Unit]
StartLimitIntervalSec=
StartLimitBurst=

[Service]
Restart=no
```

Possible values for `Restart=` are documented by systemd. We recommend using `no` to avoid automatic restarting by systemd or `on-failure` to make the service restart on non-zero exit codes. If you want a more fine-tuned restart policy, please consult the `systemd.service(5)` man page and configure the desired options.

To discard your overrides, run `systemctl revert proxedo-api-security-storage`, `systemctl revert proxedo-api-security` or `systemctl revert proxedo-api-security-mgmt`.

You only need to enter the parameters you want to change.

Overriding systemd units is only possible as `root` user.

5.6. Systemd Journal log limit setting

Systemd journal settings, namely the configuration of the *RateLimitBurst* and the *RateLimitIntervalSec* parameters limit the number of log messages. An insufficiently low number assigned to the number of *RateLimitBurst* parameter unnecessarily limits the number of log messages sent. The recommended value is 1 000 000. The interval value configured for *RateLimitIntervalSec* parameter also affects the rate limiting values for log messages. If in the time interval, specified by the *RateLimitIntervalSec* parameter, more messages are logged than specified in the *RateLimitBurst* parameter, than all further messages within that given time interval will be dropped. To turn off rate limiting, either parameter can be set to value '0'.

Even if the *Verbosity* or the *Message Filter Expression* parameters are configured to a high value in PAS, the above rate limitation settings still need to be considered.

Update this value in `/etc/systemd/journald.conf`.

```
[Journal]
RateLimitBurst=1000000
```

5.7. Tracking version

Proxedo API Security has a version number in the form of **major.minor.patch**. The docker image labels control what version the services are running at. The version tags point to a specific release and will never be changed once released. If the label is changed to a new version tag, the services will be upgraded at the restarts.

5.8. Scaling Flow Director

A single instance of *Flow Director* uses a single processor core. It is necessary to adjust the number of instances to use all the available cores. This is controlled by the **PAS_FLOW_DIRECTOR_SCALE** variable. As the *Flow Director* handles the most demanding duties among the components, it must be assigned most of the cores. If there are up to four cores available, assign three cores to the *Flow Director*, and the remaining one core will be suitable for the *Transport and Insight Director*. If there are more than four cores, assign two cores for the *Transport and Insight Director* and assign the rest to the *Flow Director*.

5.9. Configuration of dockerd

The docker daemon is configured through `/etc/docker/daemon.json`. The full documentation can be found in the official docker documentation.

Balasys recommends the use of the default configuration.

Do not use `/etc/default/docker` as it is ignored when *systemd* is used.

5.10. High availability configuration

5.10.1. HA Director

The HA functionality is implemented by the *HA Director* included in the core component. It uses keepalived in VRRP mode to provide the service.

It can be configured in two ways:

1. When installed and configured manually on the host running core, the configuration file `/opt/balasys/etc/ha/config.yml` should be filled out.
2. When installed using the automation tool, the following configuration files need to be filled in on the management node:
   - `/opt/balasys/etc/automation/common_vars.yml`: Common HA parameters to be used on remote hosts.
   - `/opt/balasys/etc/automation/host_vars`: Host-specific HA parameters to be used on remote hosts.

As at the moment, only clusters of two are supported, you can only implement HA by installing core alongside the management component as well. That instance you need to configure and set up manually, while you can use the automated deployment tool to deploy core and HA on the remote node.

5.10.2. HA restart policy

PAS service lifecycle is managed by systemd and is by default set to restart if any of the components fails at any point. To avoid infinite restarting, the number of restarts within a short period of time is also limited. As a result, if PAS core or management stops with a non-zero exit code 3 times within 100 seconds, the corresponding systemd unit will enter failed state.

The default restart policy and the options are identical for the storage, core and management components.

The relevant part of the service file looks as follows:

```
[Unit]
StartLimitIntervalSec=100
StartLimitBurst=3

[Service]
Restart=on-failure
```

Modifying the restart policy is possible by editing the service file in override mode. To do so, run `systemctl edit proxedo-api-security-storage`, `systemctl edit proxedo-api-security` or `systemctl edit proxedo-api-security-mgmt`. This will open a text editor and will let you define the parameters you wish to override. For example, if you want to switch off all default restart settings, enter the following text in the override editing window:

```
[Unit]
StartLimitIntervalSec=
StartLimitBurst=

[Service]
Restart=no
```

Possible values for `Restart=` are documented by systemd. We recommend using `no` to avoid automatic restarting by systemd or `on-failure` to make the service restart on non-zero exit codes. If you want a more fine-tuned restart policy, please consult the `systemd.service(5)` man page and configure the desired options.

To discard your overrides, run `systemctl revert proxedo-api-security-storage`, `systemctl revert proxedo-api-security` or `systemctl revert proxedo-api-security-mgmt`.

You only need to enter the parameters you want to change.

Overriding systemd units is only possible as `root` user.

5.11. Setting up time synchronization

To ensure time synchronization on different nodes you need to configure NTP on them. The `ntp` package is already installed as a dependency, but it must be configured We recommend adding the following configuration to `/etc/ntp.conf`.

```
driftfile /var/lib/ntp/ntp.drift

restrict -4 default kod notrap nomodify nopeer noquery limited
restrict -6 default kod notrap nomodify nopeer noquery limited
restrict 127.0.0.1
restrict ::1

server time.nist.gov prefer
server ip-time-1.cern.ch
```

Use your own NTP servers in the `server` directives if you have any, or adjust the given values to ones that are allowed by your policies.

After creating the configuration, run the following commands.

```
# Disable systemd-timesyncd
timedatectl set-ntp false

# Restart ntp
systemctl restart ntp

# Enable ntp so that it starts on system startup
systemctl enable ntp
```

6. Configuration of Proxedo API Security on the Web User Interface

This chapter explains configuration details for setting up a working Proxedo API Security (PAS) with the help of the Web User Interface.

The Proxedo API Security Web User Interface (UI) is installed together with the installation of Proxedo API Security. The URL for Proxedo API Security Web UI and the necessary credentials are generated when the management

component is first started. The password for the administrator can be set or generated during installation.

For information on how to set up more users, see section *Configuring authentication and local users in PAS*.

By using OpenAPI schemas, as defined in OpenAPI specifications (also known as Swagger), Proxedo API Security verifies that the traffic passing through conforms to the API endpoint's specification. An OpenAPI Swagger schema detailing the Configuration API is available at: `<frontend_url>/api/v1/openapi`. `<frontend_url>` here refers to the URL address of the user's Proxedo API Security Web User Interface.

6.1. Minimum configuration

It is possible to run PAS with a minimum, basic configuration. For a minimum configuration the following items need to be configured in the Web UI:

- BRICKS / File
  - Name
  - Type: License
  - File
    For more details on the License *File*'s requirements, see File types.
- SYSTEM / License
  - License File
    For more details on the *License*'s parameters, see License's configuration.
- SERVICES / Backend
  - Name
  - Servers
    For more details on the *Backend*'s parameters, see Backend's configuration.
- SERVICES / Endpoint
  - Name
  - URLs
  - Backend
  - Request
  - Response
    For more details on the *Endpoint*'s parameters, see Security Flow and Endpoint's configuration.
- SERVICES / Listener
  - Name
  - Endpoints
    For more details on the *Listener*'s parameters, see Listener's configuration.

This basic configuration can be further improved with the completion of more configuration units later. The minimum configuration can also be used to test the installation settings.

6.2. Login Page

The main component of the Login page is the login form where the user needs to provide the credentials in order to be authorized to use the Web UI of Proxedo API Security.

As part of the initial configuration of Proxedo API Security, the administrator defines the necessary credentials, which can now be used.

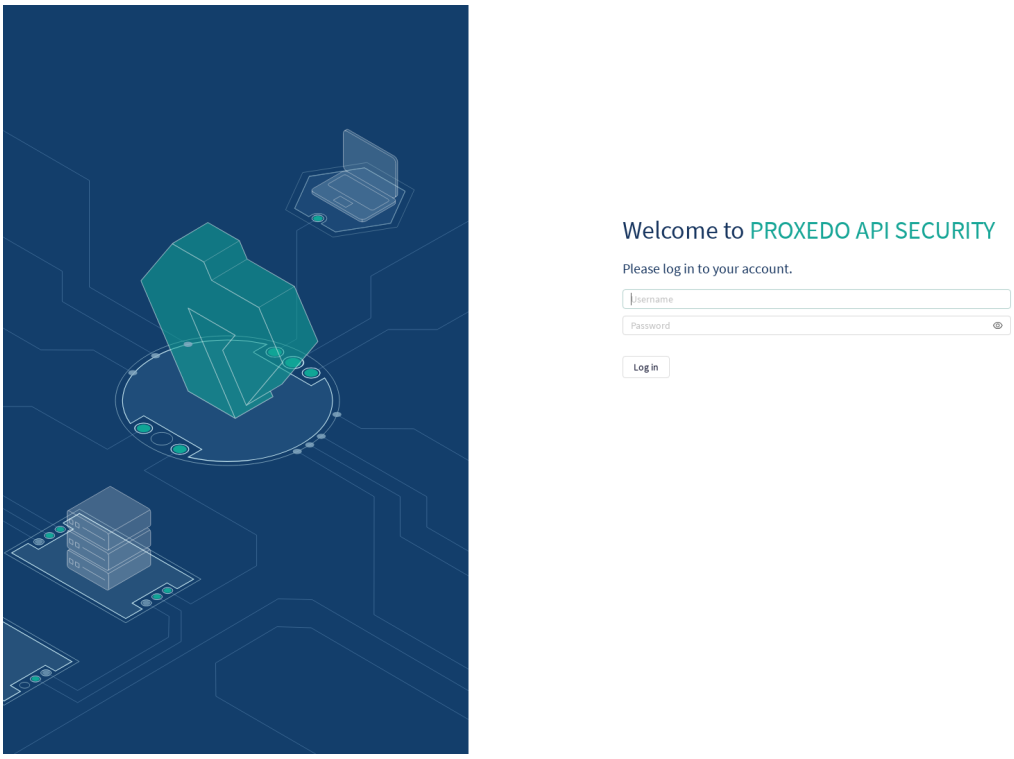


Figure 25. Login page for Proxedo API Security Web User Interface

For accessing the Web User Interface:

1. Enter the valid user credentials.
2. Click the **Log In** button.

After a successful login, the user has access to the Proxedo API Security Web UI.

6.3. Proxedo API Security Web User Interface main page

The configuration elements are organized into a logical order for easier usage.

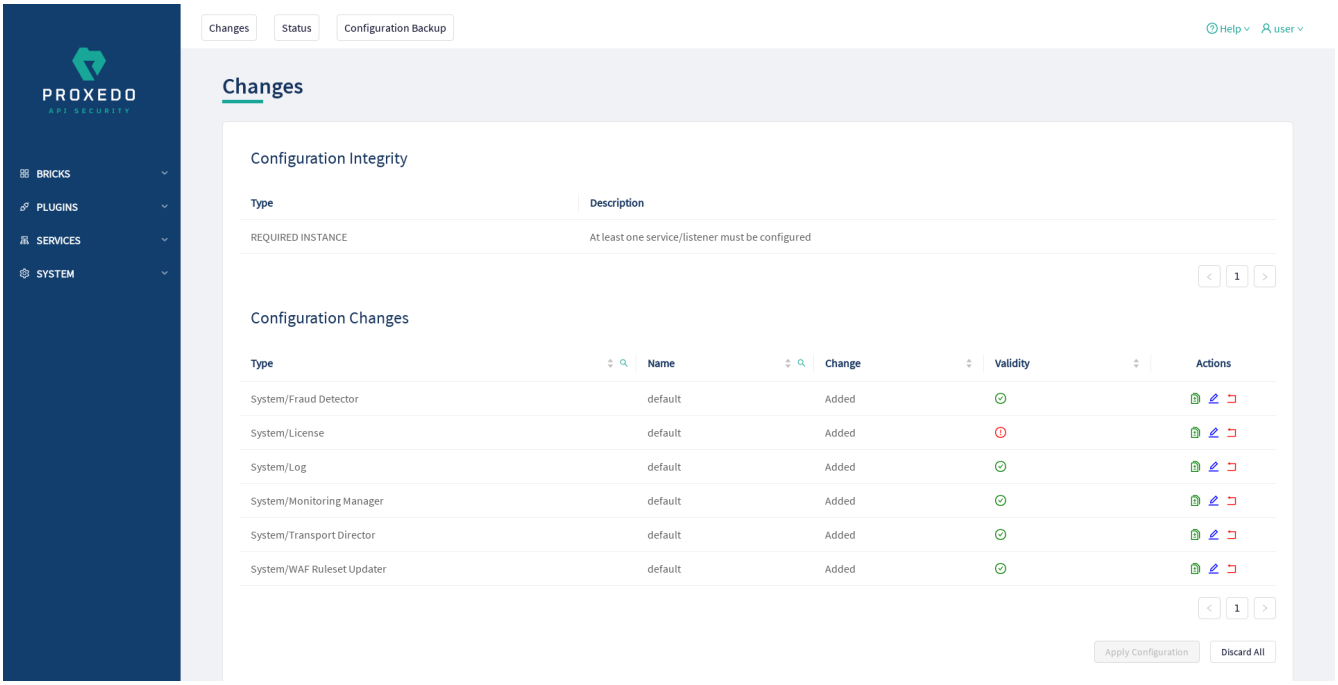


Figure 26. Proxedo API Security Web User Interface main page

6.3.1. Navigation

The PAS Web UI has the following navigation areas:

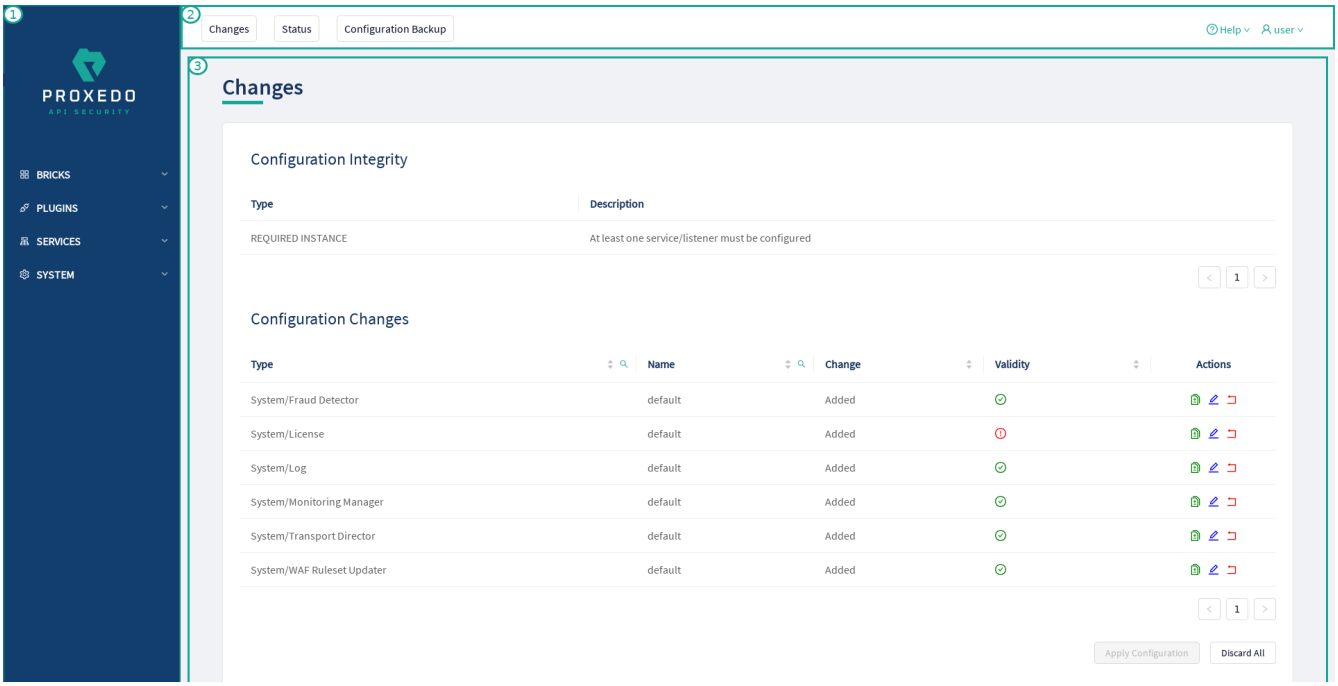


Figure 27. Navigation areas in the Proxedo API Security Web User Interface

The navigation areas are described here in more details:

Left navigation area (1)

This navigation area (1) presents the navigation units available for configuration.
When opening up the Proxedo API Security Web UI, four main navigation units are available, that is, BRICKS, PLUGINS, SERVICES, and SYSTEM.
These four main navigation units can be opened for further sub-navigation units by clicking on either the navigation item itself or on the arrow icon next to it. Alternatively, when the sub-navigation units are not in use, they can be hidden by clicking the arrow navigation icons next to the main navigation items, or similarly by clicking on the navigation item itself.

Top navigation area (2)

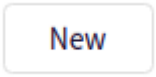
This Top navigation area (2) presents the *Changes*, *Status* and *Configuration Backup* buttons in the top left corner. For more information on these services, see Checking and finalizing changes in Proxedo API Security configuration, System-wide status information and Backup and restore running or user configuration for Proxedo API Security. The top right corner presents the *Help* button and a *Profile* button that shows the current user's name. The *Logout* option is present under the *Profile* button.

Main configuration area (3)

This is the main configuration area of the Web UI. Any navigation unit selected in the Left navigation area (1) presents the configuration details in this Main configuration area (3). The configuration details can be edited in this area.
In case there are already configured parameters, those are displayed in a table in the Main configuration area (3).
In order to add more configuration details, select the *New* navigation button in the upper right corner.

The Main configuration area (3) provides the following navigation and activity options. Note that some of these activities are also available when the configuration parameters are presented in list view:

Table 14. Navigation and activity options in the Main configuration area (3)

Navigation option	Description
	By selecting the *New* navigation button on the active window of a component, a new component can be configured.
	By selecting the *Pen* navigation button next to a component, the Web UI navigates back to the configuration page of the selected element. The so far configured details can be changed or new configuration details can be added.
	By selecting the *Copy* navigation button next to a component, the Web UI copies all the information of that component into a new instance, which instance can be saved with a new name, inheriting the same, copied parameters.
	By selecting the *Bin* button next to a component, the configuration element can be deleted. If an element is selected for deletion, a *Warning appears*, requesting confirmation on the deletion of the element.
	This icon is visible at the right side of every drop-down list during configuration. By selecting this icon it is possible to unselect an item of the drop-down list and to clear the selection field from any data. Clearing the field from data with the help of this icon gains importance when an earlier selected drop-down list item, saved in our configuration, has to be cleared from the configuration data.
	By selecting the *Next page* button it is possible to navigate to the next page of the parameter keys listed.

6.3.2. Naming Configuration components in the Web UI

When configuring the Proxedo API Security Web UI, name the configuration components with the usage of the English alphabet and numerals. When the name is composed of more than one word, use underscore. It is not allowed to use spacing or any special characters though.

6.4. BRICKS - Configuration units

Bricks are reusable components. They do not provide a complete security function themselves, instead, they are used as building blocks elsewhere (hence the name). They can be used by *Plugins* (like Selectors), or utilized by other bricks (like Extractors).

Certain bricks are so called *default* objects, which are in 'read-only' state and cannot be configured or modified. Such default objects are listed in the following table:

Table 15. Default objects - BRICKS

Default object name	Class
always	Matcher
never	Matcher
content_type_json	Matcher
content_type_json_pattern	Matcher
json_content	Matcher
content_type_xml_base	Matcher
content_type_xml_dtd	Matcher
content_type_xml_ext_parsed	Matcher

Default object name	Class
content_type_xml_pattern	Matcher
content_type_xml_text	Matcher
content_type_xml_text_ext_parsed	Matcher
xml_content	Matcher
error_policy	Error policy
enforcer_default	Error policy
insight_default	Error policy
client_address	Selector
client_port	Selector
server_address	Selector
server_port	Selector
error_policy	Selector
error_policy_action	Selector
error_policy_status_code	Selector
error_policy_silent	Selector
error_policy_message	Selector
plugin_name	Selector
plugin_verdict	Selector
plugin_error_message	Selector

These default objects are listed under the actual classes in the Web UI.

The *BRICKS* main page in the Web UI is as follows:

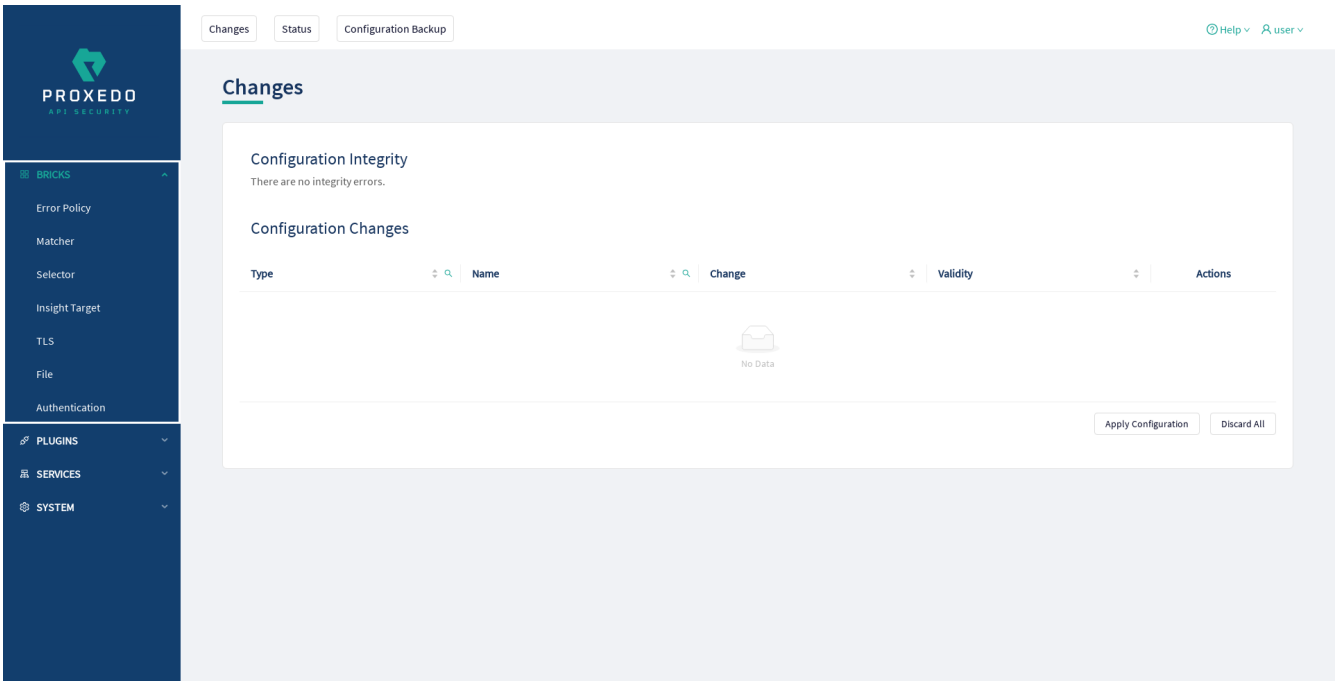


Figure 28. The BRICKS main page in the Web User Interface

1. Click on the *BRICKS* main configuration item in the Left navigation area. Alternatively you can also click on the sign to open up the sub-navigation items of *BRICKS*.
2. Click on the sub-navigation unit you would like to configure. The details of the sub-navigation menu open up in the Main configuration area.

6.4.1. Error Policy

Error Policies define how to proceed if a *Plugin* decides to have found an error. For example, when an *Enforcer plugin* decides that the call is invalid.

It is the error policy that enables the user to act differently in case the error appears in a request or a response.

Every Plugin has a default error policy, namely, the 'error_policy', except for the Enforcer and the Insight Plugins, which have their own default error policies already configured for usage, the enforcer_default and the insight_default error policies.

6.4.1.1. Configuring Error Policies

1. Click on the *BRICKS* main navigation item in the Left navigation area. Alternatively you can also click on the sign to open up the sub-navigation items of *BRICKS*.
2. Select *Error Policy*.

The configuration window that appears presents the default error policies, as listed in Default objects - BRICKS and the configuration values already set by the user:

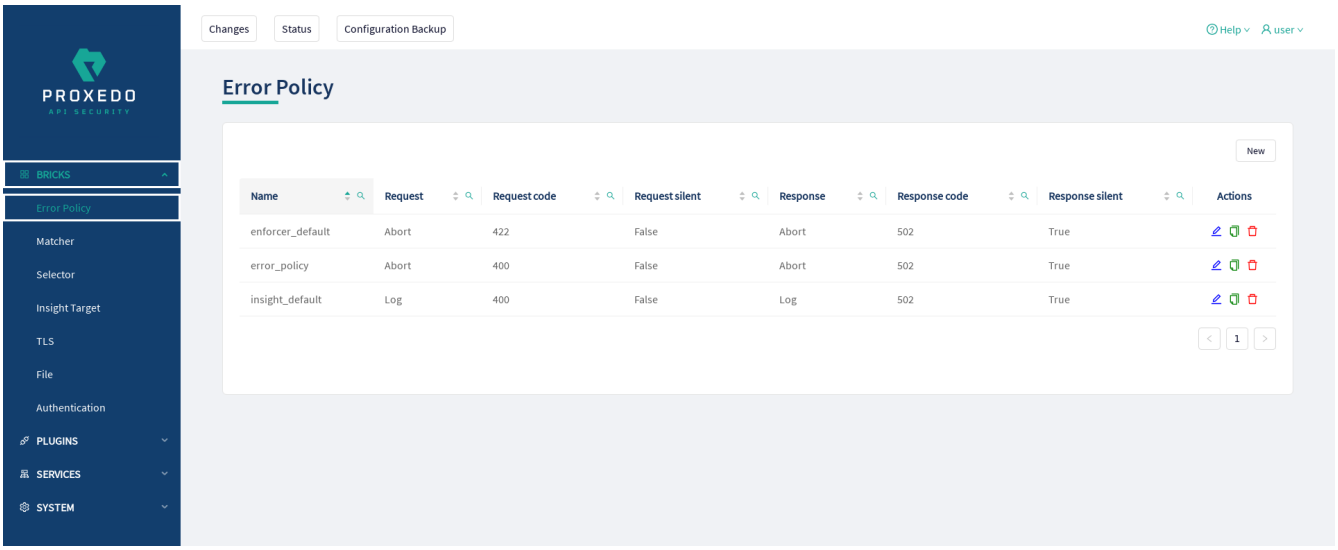


Figure 29. Error policy's main page in the Web User Interface

3. Click on the *New* navigation button to create an Error Policy.

Error Policies have default values for each of their fields. They form a strict security policy: all errors are fatal, and only errors made by the client are reported in detail.

4. Configure the necessary parameters for the error policy based on the details provided in the table Error policy configuration options.
5. Click the *Validate* button to check if the defined parameters are of the correct type and all required parameters have been filled out for configuring the component. If the configuration is erroneous or incomplete, the Web UI provides a warning that the 'Component validation failed'. Also a warning with information on the missing or faulty elements appears at the problematic field. If the configuration of the component is valid, after clicking the *Validate* button, a 'Component validation successful' notification is shown.
6. Save the component configuration by clicking the *Save* button.

The error policies configured here can be used in the *Plugin's* configuration, by referencing their name.

The following values can be configured for the *Error Policy Brick*:

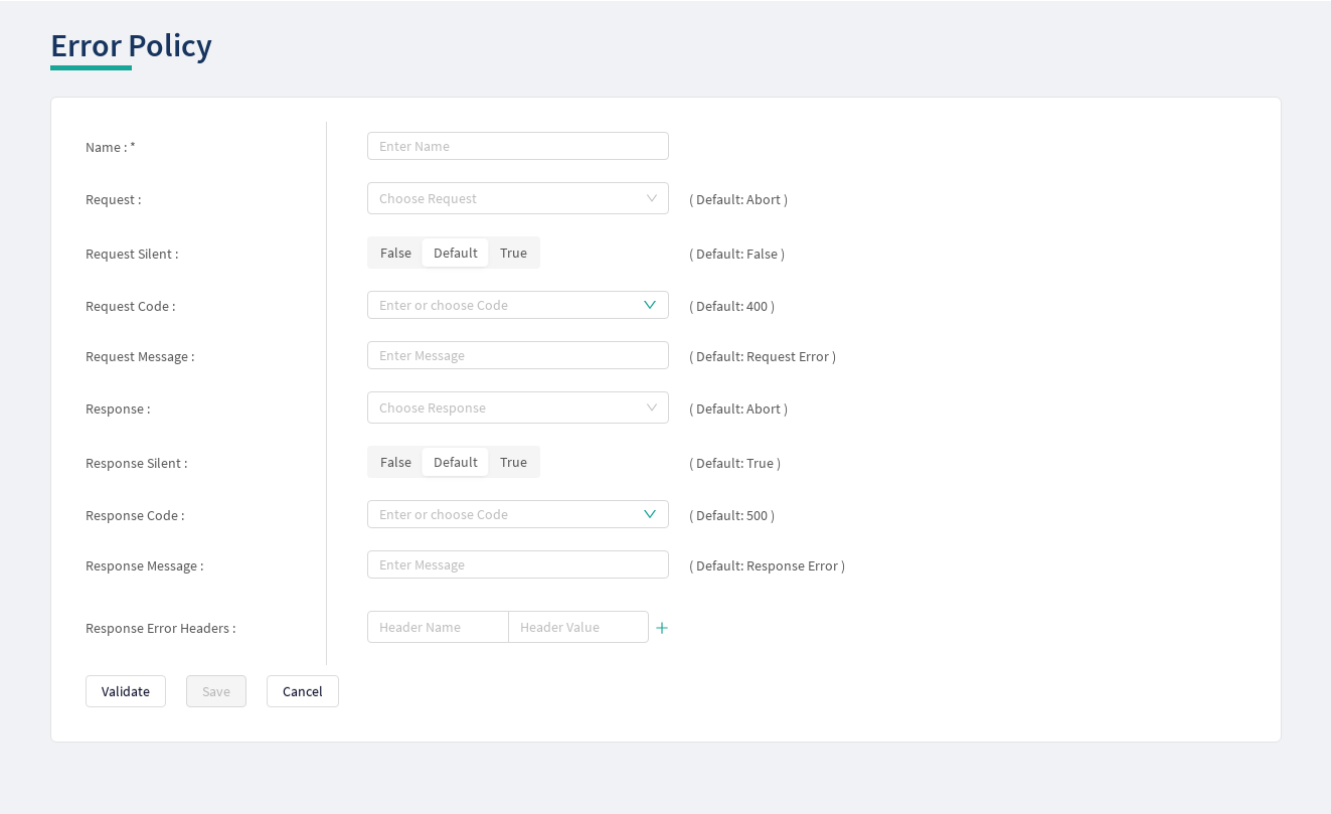


Figure 30. Configuring error policies in the Web User Interface

Table 16. Error policy configuration options

Key	Values	Default value	Description
Name*	Free text. Alphanumeric, may contain underscores, may not start with a number.		The name identifying the error policy. This name of the error policy can be referenced from other parts of the configuration, that is, the error policy is reusable.
Request	The available values are: • Abort • Log	Abort	It defines what action shall take place if there is an error on the request side: • Abort: the request is denied if the *Plugin* fails. Use the other parameters to control the content of the error sent to the client. • Log: the invalid requests are allowed, but are logged.
Request Silent	True or False.	True	When turned on, the *Plugins* do not report on the denial of the invalid request. When turned off, the *Plugins* have the ability to report the error in detail in the body of the HTTP error request.
Request Code	The values are available from a drop-down list. If the elements of the drop-down list are selected, it will make the list of the actual request codes visible. The applicable request code can be selected.	422	The HTTP status code to be used when denying invalid requests.
Request Message	The message can be provided in free text.	Request error	The HTTP response line when denying invalid requests.

Key	Values	Default value	Description
Response	Response error mode: • Abort • Log	Abort	It defines what action shall take place if there is an error on the request side: • Abort: the request is denied if the *Plugin* fails. Use the other parameters to control the content of the error sent to the client. • Log: the invalid requests are allowed, but are logged.
Response Silent	True or False.	True	When turned on, the *Plugins* do not report on the denial of the invalid response. When turned off, the *Plugins* have the ability to report the error in detail in the body of the HTTP error response.
Response Code	The values are available from a drop-down list. Note that the response codes are grouped, so that if the elements of the drop-down list are selected, further groups of response codes will be made visible in a tree structure. The applicable request code can be selected.	502	The HTTP status code to be used when denying invalid requests.
Response Message	The message can be provided in free text.	Response error	The HTTP response line when denying invalid requests.
Response Error Headers	A list of header name-value pairs.	Empty list	A list of HTTP header key-value pairs to include in the response when denying invalid requests.

6.4.2. Matcher

Matchers decide if the Plugin should be executed for a given call by checking various data in the HTTP message. They provide an extremely versatile way of defining the circumstances that must be met for the *Plugin* to execute.

Matchers need four pieces of information:

- **Name**: The **Name** field can be defined in free text and it is not related to the extractor that will be used. This **Name** can be referenced in Plugins.
- **Type**: This parameter defines what part of the call needs to be checked.
- **Comparator**: The Comparator shows by what means the collected value of the call is compared with the provided pattern. (Some comparators also take flags or arguments.)
- **Expression**: A regular expression specifies a set of strings that match it. A complete explanation on how to write expressions is not in the scope of this document.

The matchers can be used in Plugin configurations' match option by referencing their name.

There are some named Matchers available without explicit configuration:

- **always** and **never** are instances of Always matcher and Never matcher.
- **json_content** that matches requests with the Content-Type headers representing JSON.

Also note that no other matchers can be defined with these names.

Matchers internally utilize Extractors to fetch the information from the call to compare with. The **Type** of the matcher resembles the name of the extractor that will be used.

All matchers have a default comparator that is applied implicitly.

If you want to use comparator parameters, the comparator name should be given even if the default comparator is used.

6.4.2.1. Configuring Matchers

1. Click on the *BRICKS* main navigation item in the Left navigation area. Alternatively you can also click on the  sign to open up the sub-navigation items of *BRICKS*.
2. Select *Matcher*.

The configuration window that appears presents the default matchers, as listed in Default objects - BRICKS and the configuration values already set by the user:

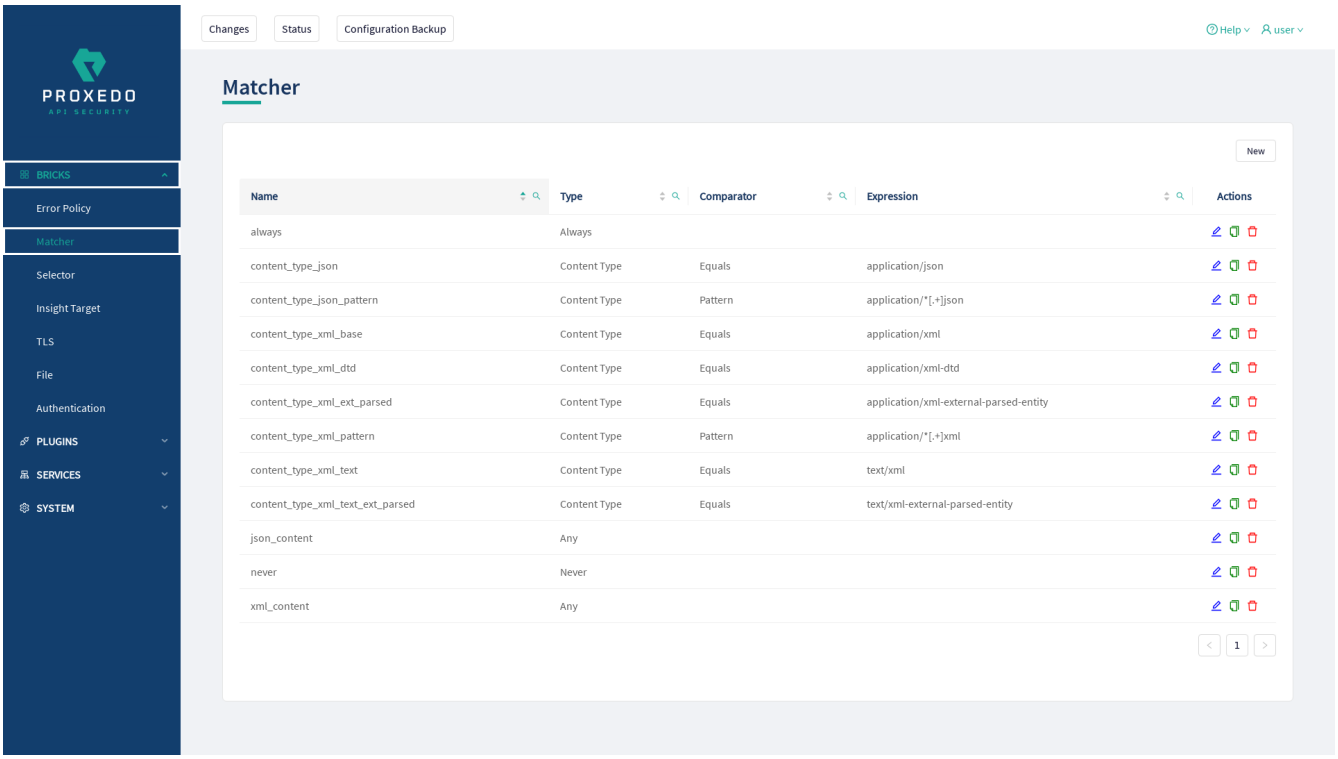


Figure 31. Matchers' main page in the Web User Interface

3. Click on the *New* navigation button to create a Matcher.
4. Provide the name of the matcher.
5. Choose the type of the matcher from the drop-down list.
6. Configure the necessary parameters with the help of the below tables.
7. Click the *Validate* button to check if the defined parameters are of the correct type and all required parameters have been filled out for configuring the component. If the configuration is erroneous or incomplete, the Web UI provides a warning that the 'Component validation failed'. Also a warning with information on the missing or faulty elements appears at the problematic field. If the configuration of the component is valid, after clicking the *Validate* button, a 'Component validation successful' notification is shown.
8. Save the component configuration by clicking the *Save* button.

The following values can be configured for the *Matcher Brick*:

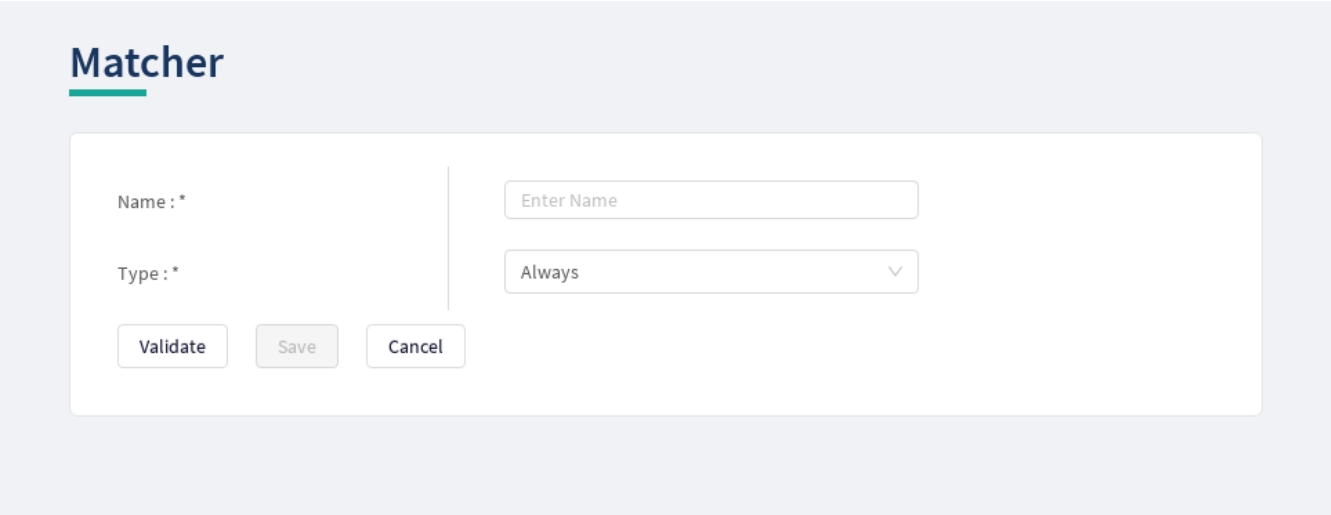


Figure 32. Configuring matchers in the Web User Interface

Table 17. Matcher configuration options

Key	Values	Default value	Description
Name*	Free text. Alphanumeric, may contain underscores, may not start with a number.	It can be defined in free text.	The **Name** of the matcher which can be referenced in Plugins.
Type*	For the available values, see Matcher types.		The preferred matcher type has to be selected from the drop-down list.

Matcher types

Depending on the choice of the matcher type, some more required configuration fields might appear on this page. The following tables describe the matcher types in details and provide the necessary information for the additional configuration fields, required for setting the matcher types:

- Matcher types and their settings - Simple matchers
- Matcher types and their settings - Compound matchers
- Matcher types and their settings - URI matchers
- Matcher types and their settings - SOAP matchers

Table 18. Matcher types and their settings - Simple matchers

Matcher type	Description
Always	This matcher always matches.
Never	This matcher never matches. It can be used to turn off a *Plugin*.
Call Direction	Matches the direction of the message (request or response).
Backend Response Time	Matches the time spent between the sending the request towards the server and receiving the response from the server, in milliseconds. Only matches in a response flow.

Matcher type	Description
Method	Matches the HTTP method of the request. Note that the standard and the practice differs regarding upper and lower casing, set case sensitivity according to needs. When choosing the *Method* matcher from the drop-down list, additional parameters appear. For more information on the configuration of these parameters, see Matcher types' additional configuration options.

Header	It matches the value of an HTTP header. Some HTTP headers can be present more than once in a call. To accommodate this, matching is completed against the value of **each** occurrence of the header. Matching occurs if there is any match. For example, if the *Accept* header was repeated as follows:
	
	`Accept: application/json`
	`Accept: application/xml`
	
	Consequently, in this example above both `header.accept: application/json` and `header.accept: application/xml` would match.
	
	To match against the header named **server** the key will be `header.server`, possibly completed with comparator specification, like `header.server.regex.ignorecase`.
	
	While the values are not, the HTTP header names are case insensitive, so you can write them all lowercase in the configuration key.
	
	The syntax of this matcher differs from the others because the name of the *Header* must be added.
	
	While the values are not, the HTTP header names are case insensitive, so you can write them all lowercase in the configuration key.

Cookie	Matches the value of a key in the Cookie HTTP header. A Cookie header key can be present more than once in a call. To accommodate this, matching is completed against the value of **each** occurrence of the key. Matching occurs if there is any match.
Content Type	Matches the content type of the message. It is a more robust solution than using the *Header* matcher on the *Content-Type* header because that can contain parameters as well. When choosing the *Content type* matcher from the drop-down list, additional parameters appear. For more information on the configuration of these parameters, see Matcher types' additional configuration options.
Status	Matches the status code of the response. See the default Status class comparator which allows convenient matching on HTTP status classes. When choosing the *Status* matcher from the drop-down list, additional parameters appear. For more information on the configuration of these parameters, see Matcher types' additional configuration options.

Matcher type	Description
Raw Content	Matches the raw bytes of the request or response. It requires an expression in the form of a hexadecimal string. For example, for matching a PNG image file, the expression shall be '89504e470d0a1a0a', which is equivalent to '89 50 4e 47 0d 0a 1a 0a', as whitespaces can also be used.
Text Content	Matches the request's or response's content as a decoded string.
Client Address	Matches the client's IP address (both IPv4 and IPv6). Use the *subnet* type comparator with that matcher type. The *subnet* comparator examines if the IP address of the Client is in the specified subnet. The format for the input of the subnet comparator is the CIDR notation for IPv4 (for example, 192.0.2.0/24) and canonical prefix notation for IPv6 (for example, 2001:db8::/32).
Client Port	Matches the client's port (TCP).
Server Address	Matches the server's IP address (both IPv4 and IPv6). Use the *subnet* type comparator with that matcher type. The *subnet* comparator examines if the IP address of the Server is in the specified subnet. The format for the input of the subnet comparator is the CIDR notation for IPv4 (for example, 192.0.2.0/24) and canonical prefix notation for IPv6 (for example, 2001:db8::/32).
Server Port	Matches the server's port (TCP).
Error Policy Action	Matches the *Request* or *Response* field of the *Error Policy* of the *Plugin* selected by the *plugin* field. If the *plugin* field contains the special value "Previous", the data will be extracted from the last evaluated plugin in the same HTTP request or response, if there is one. If the *plugin* field refers to a specific plugin instance, the data will be extracted from the last evaluation of the referred plugin instance. If the matcher is in the HTTP response, and the last evaluation is in the HTTP request, then the evaluation result from the HTTP request will be selected. Only plugins with a negative verdict will return data.
Error Policy Status Code	Works like the *Error Policy Action* matcher, but matches only the *Request Code* or *Response Code* field of the referred plugin's *Error Policy*.
Error Policy Silent	Works like the *Error Policy Action* matcher, but matches only the *Request Silent* or *Response Silent* field of the referred plugin's *Error Policy*.
Error Policy Message	Works like the *Error Policy Action* matcher, but matches only the *Request Message* or *Response Message* field of the referred plugin's *Error Policy*.
Plugin Name	Works like the *Error Policy Action* matcher, but matches the name of the referred plugin. Returns data regardless of verdict.
Plugin Verdict	Works like the *Error Policy Action* matcher, but matches the verdict of the referred plugin. Returns data regardless of verdict.
Plugin Error Message	Works like the *Error Policy Action* matcher, but matches the error message provided in the referred plugin's negative verdict, if there is one.

Matcher type	Description
XPath	Matches the data from the body of an XML call with the help of the XPath expression. XPath is a query language for XML. It is a very versatile tool for extracting the needed information from the body of the call, and organizing it according to needs. A complete explanation on how to write XPath expressions is not in the scope of this document. To learn more about it visit the main website. For more details on XPath configuration options, see XPath extractor configuration options.
JMESPath	Matches the data from the body of a JSON call with the help of the JMESPath expression. JMESPath is a query language for JSON. It is a very versatile tool for extracting the needed information from the body of the call, and for organizing it according to needs. A complete explanation on how to write JMESPath expressions is not in the scope of this document. To learn more about it visit the JMESPath website: • There is a tutorial. • There are examples. • There is also a formal specification. When choosing the *JMESPath* matcher from the drop-down list, additional parameters appear. For more information on the configuration of these parameters, see Matcher types' additional configuration options. The result of the JMESPath expression should be a string when using string comparators (*Equals*, *Starts with*, etc.), and number when using number comparators (*Min*, *Max*, *Range*). In case of boolean or complex types, convert to string in the JMESPath expression and use the string representation of the result. Example: instead of comparing the boolean result of `address != ''`, use `to_string(address != '')` with a string comparator.
Fraud Detector Score	Matches the score value provided by the *Fraud Detector* plugin.

Table 19. Matcher types and their settings - Compound matchers

Any	Any is a Compound matcher that matches if any of its sub-matchers matches. The sub-matcher can also be a compound matcher.
All	All is a Compound matcher that matches if all of its sub-matchers match. The sub-matcher can also be a compound matcher.
None	None is a Compound matcher that matches if none of its sub-matchers match. The sub-matcher can also be a compound matcher.
One	One is a Compound matcher that matches if exactly one of its sub-matchers matches. The sub-matcher can also be a compound matcher.

Table 20. Matcher types and their settings - URI matchers

Matcher type	Description

URI matchers	A range of matchers is available to match different parts of the URI.
	
	The structure of an URI looks as follows:
	
	`scheme://[username[:password]@]host[:port][/path][?query][#fragment]`
	
	That is, for example:
	
	`https://john.doe:secret123@example.com:8443/some/resource?foo=bar&baz=qux#some-anchor`
	
	The fragment part is used by the client locally, and is never sent in the HTTP requests, therefore PAS cannot do anything with it.
	
	These matchers use the URI extractors. It has an extensive list of examples of what each extractor extracts from the URI.

URI	Matches against the whole request URI as received from the client. When choosing the *URI* matcher from the drop-down list, additional parameters appear. For more information on the configuration of these parameters, see Matcher types' additional configuration options.
URI netloc	Matches the network location in the URI. It includes: • **username** and **password** if present • **host** • **port** if present unless scheme default If the port is the default port for the scheme - that is 80 and 443 for HTTP and HTTPS, respectively - the port will not be included even if explicitly sent by the client. Therefore if the client used `http://example.com:80/path` then the *netloc* would be `http://example.com`, **not** `http://example.com:80`. When choosing the *URI netloc* matcher from the drop-down list, additional parameters appear. For more information on the configuration of these parameters, see Matcher types' additional configuration options.

Matcher type	Description
URI origin	Matches the *origin* part of the URI. It includes: • **scheme** • **host** • **port** if present, unless the default port for the scheme is used  If the port is the default port for the scheme - that is 80 and 443 for HTTP and HTTPS, respectively - the port will not be included, even if explicitly sent by the client. Therefore if the client used `http://example.com:80/path`, then the *origin* would be `http://example.com`, **not** `http://example.com:80`. When choosing the *URI origin* matcher from the drop-down list, additional parameters appear. For more information on the configuration of these parameters, see Matcher types' additional configuration options.
URI scheme	Matches the scheme of request (http or https). Note that the scheme is case insensitive by definition, therefore the case will always be ignored. When choosing the *URI scheme* matcher from the drop-down list, additional parameters appear. For more information on the configuration of these parameters, see Matcher types' additional configuration options.
URI username	Matches the *username* in the request if present. When choosing the *URI username* matcher from the drop-down list, additional parameters appear. For more information on the configuration of these parameters, see Matcher types' additional configuration options.
URI password	Matches the *password* in the request if present. When choosing the *URI password* matcher from the drop-down list, additional parameters appear. For more information on the configuration of these parameters, see Matcher types' additional configuration options.
URI host	Matches the *host* in the request. When choosing the *URI host* matcher from the drop-down list, additional parameters appear. For more information on the configuration of these parameters, see Matcher types' additional configuration options.
URI port	Matches the *port* of the request. Note that this matches the default *port* — that is 80 and 443 for HTTP and HTTPS, respectively — even if it is not explicitly in the request. When choosing the *URI port* matcher from the drop-down list, additional parameters appear. For more information on the configuration of these parameters, see Matcher types' additional configuration options.

Matcher type	Description
URI path	Matches the *path* part of the URI. The path is normalized to allow more robust matching and cleaner reporting. This means that: • If the path is missing `/` it is extracted. • Repeating forward-slash (`/`) characters are replaced with a single one. • dot (`.`) and double-dot (`..`) path segments are resolved. Consequently, if the path present in the *URI* was `//some/./nothere/../resource///./somewhere` the *path* would be `/some/resource/somewhere`. If you need to match the *path* exactly as received, use URI raw path matcher. When choosing the *URI path* matcher from the drop-down list, additional parameters appear. For more information on the configuration of these parameters, see Matcher types' additional configuration options.
URI raw path	Matches the *path* part of the URI, without the normalization of URI path matcher carried out. If the *path* is missing, the match still runs against a single forward slash ("/"). It is recommended to use URI path matcher unless there is an explicit need for matching the raw path. One such example would be logging or filtering out badly formed requests. When choosing the *URI raw path* matcher from the drop-down list, additional parameters appear. For more information on the configuration of these parameters, see Matcher types' additional configuration options.
URI raw query	Matches the *query* part of the URI as a string. It is recommended to use URI query parameter matcher unless there is an explicit need for matching the raw string. An example on this might be if there is a match on `foo=barbar` or `tofoo=bar` as well, even though it was not intended. When choosing the *URI raw query* matcher from the drop-down list, additional parameters appear. For more information on the configuration of these parameters, see Matcher types' additional configuration options.
URI query parameter	Matches the value of a query parameter. It is also valid for URIs to include a query parameter more than once. That is, it could be *foo=bar&qux=quz&foo=baz*. To accommodate this, matching is done against the value of *each* occurrence of the parameter. Matching occurs if any value is matched. When choosing the *URI query parameter* matcher from the drop-down list, additional parameters appear. For more information on the configuration of these parameters, see Matcher types' additional configuration options.

Table 21. Matcher types and their settings - SOAP matchers

Matcher type	Description
SOAP Matchers	A range of matchers is available to match different parts of the SOAP message. These matchers extend the XPath matcher with predefined expressions. They use the SOAP extractors. It has an extensive list of examples of what each extractor extracts from the SOAP message. When choosing the *SOAP Matchers* matcher from the drop-down list, additional parameters appear. For more information on the configuration of these parameters, see Matcher types' additional configuration options.
SOAP version	Matches the SOAP message version. It identifies with the SOAP namespace. The possible values are: • **soapv1_1** - the message version is SOAP v1.1 • **soapv1_2** - the message version is SOAP v1.2 When choosing the *SOAP version* matcher from the drop-down list, additional parameters appear. For more information on the configuration of these parameters, see Matcher types' additional configuration options.
SOAP envelope	Matches the SOAP envelope. When choosing the *SOAP envelope* matcher from the drop-down list, additional parameters appear. For more information on the configuration of these parameters, see Matcher types' additional configuration options.
SOAP header	Matches the SOAP header. When choosing the *SOAP header* matcher from the drop-down list, additional parameters appear. For more information on the configuration of these parameters, see Matcher types' additional configuration options.
SOAP body	Matches the SOAP body. When choosing the *SOAP body* matcher from the drop-down list, additional parameters appear. For more information on the configuration of these parameters, see Matcher types' additional configuration options.
SOAP fault	Matches the SOAP fault. When choosing the *SOAP fault* matcher from the drop-down list, additional parameters appear. For more information on the configuration of these parameters, see Matcher types' additional configuration options.
SOAP fault code	Matches the SOAP fault 'code'. The expression depends on the SOAP version. • **faultcode** - it is the SOAP v1.1 node tag. • **Code** - it is the SOAP v1.2 node tag. When choosing the *SOAP fault code* matcher from the drop-down list, additional parameters appear. For more information on the configuration of these parameters, see Matcher types' additional configuration options.

Matcher type	Description
SOAP fault detail	Matches the SOAP fault 'detail'. The expression depends on the SOAP version. • **Detail** - it is the SOAP v1.1 node tag. • **Detail** - it is the SOAP v1.2 node tag. When choosing the *SOAP fault details* matcher from the drop-down list, additional parameters appear. For more information on the configuration of these parameters, see Matcher types' additional configuration options.
SOAP 1.1 fault faultstring	Matches the SOAP fault 'faultstring'. This matcher only works with SOAP version 1.1. When choosing the *SOAP 1.1 fault faultstring* matcher from the drop-down list, additional parameters appear. For more information on the configuration of these parameters, see Matcher types' additional configuration options.
SOAP 1.1 fault faultactor	Matches the SOAP fault 'faultactor'. This matcher only works with SOAP version 1.1. When choosing the *SOAP 1.1 fault faultactor* matcher from the drop-down list, additional parameters appear. For more information on the configuration of these parameters, see Matcher types' additional configuration options.
SOAP 1.2 fault reason	Matches the SOAP fault 'Reason'. This matcher only works with SOAP version 1.2. When choosing the *SOAP 1.2 fault reason* matcher from the drop-down list, additional parameters appear. For more information on the configuration of these parameters, see Matcher types' additional configuration options.
SOAP 1.2 fault node	Matches the SOAP fault 'Node'. This matcher only works with SOAP version 1.2. When choosing the *SOAP 1.2 fault node* matcher from the drop-down list, additional parameters appear. For more information on the configuration of these parameters, see Matcher types' additional configuration options.
SOAP 1.2 fault role	Matches the SOAP fault 'Role'. This matcher only works with SOAP version 1.2. When choosing the *SOAP 1.2 fault role* matcher from the drop-down list, additional parameters appear. For more information on the configuration of these parameters, see Matcher types' additional configuration options.

For details on comparator types, see Types of comparators.

Depending on the matcher type selected, the administrator might need to fill in further parameters. These parameters are described in the following table.

Table 22. Matcher types' additional configuration options

Key	Values	Default value	Description
Comparator			The matchers need the information on the Comparator, which shows by what means the collected value of the call is compared with the provided pattern.
Type*	The available comparator types can be checked from the drop-down list.	Equals	This configuration option has to be defined for the Comparator. For details on the comparator types, see Types of comparators.

Key	Values	Default value	Description
Ignorecase	True or False.	False	This configuration option has to be defined for the Comparator. It sets the IGNORECASE flag for the selected comparator type. For matcher types that work with numeric data type or with IP addresses, the 'Equals' and 'Not Equals' comparator types do not have ignorecase field.
Expression*			This configuration option has to be defined for the Comparator. A regular expression specifies a set of strings that match it.
JMESPath Expression*	A valid JMESPath expression in text.		A complete explanation on how to write JMESPath expressions is not in the scope of this document. To learn more about it visit the JMESPath website: • There is a tutorial. • There are examples. • There is also a formal specification.
Query Parameter			It is also valid for URIs to include a query parameter more than once. That is, it could be *foo=bar&qux=quz&foo=baz*. To accommodate this, matching is done against the value of *each* occurrence of the parameter. Matching occurs if any value is matched.
Header			Extracts the value of an HTTP header. It is valid for some HTTP headers to be present more than once in a call. In this case, all the values are extracted as a list. It provides the name of the header in the configuration.
Namespaces	A list of key and expression pairs, in text.		The namespaces to use during extraction.
XPath Expression	A valid XPath expression in text.		A complete explanation on how to write XPath expressions is not in the scope of this document. • There is a tutorial. • There are examples. • There is also a formal specification.
Multiline			Sets the Multiline flag for the *Regex* comparator.
Minimum*			Matches if the pattern is larger or equal to the value.
Maximum*			Matches if the pattern is smaller or equal to the value.

Key	Values	Default value	Description
Source Plugin	Reference to a *Fraud Detector Plugin* or "Last".	Last: In case there are more Fraud Detector plugins defined in the Security Flow, by using this default value, the selector will use the score value provided for the last run Fraud Detector plugin.	The Fraud Detector plugin to be used in case there are more than one defined.
Plugin	Reference to a *Plugin* or "Previous".	Previous: A special value that dynamically selects the plugin that was most recently evaluated in the security flow.	The plugin from which data will be extracted.

6.4.3. Selector

Selectors are responsible for collecting information from the call. They utilize Extractor bricks for this purpose.

Most extractors return simple string values. However, some (might) return dictionaries. For example, you can get all the HTTP headers, or all the URI query parameters.

They are used by Insight plugins and Fraud Detector plugins.

6.4.3.1. Configuring Selectors

1. Click on the *BRICKS* main navigation item in the Left navigation area. Alternatively you can also click on the  sign to open up the sub-navigation items of *BRICKS*.
2. Select *Selector*.

In the configuration window that appears, you can either see the empty parameter values that can be configured for the actual component or you can see already configured component(s) and their parameters. The already configured components with defined parameters can be default components available in the system by default, or can be components configured by the administrator:

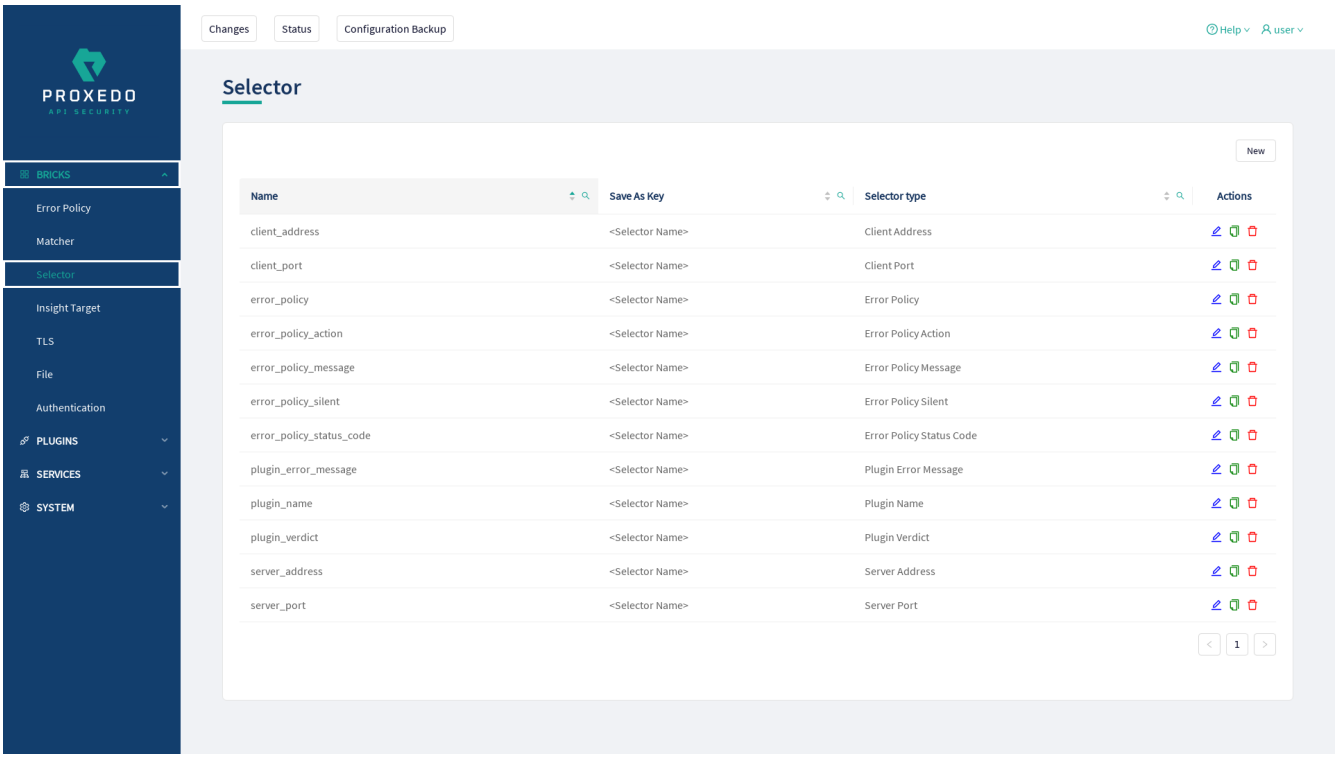


Figure 33. Selector main page in the Web User Interface

3. Click on the *New* navigation button to create a Selector.
4. Name the *Selector* key.
5. Fill in any more desired parameters.
6. Click the *Validate* button to check if the defined parameters are of the correct type and all required parameters have been filled out for configuring the component. If the configuration is erroneous or incomplete, the Web UI provides a warning that the 'Component validation failed'. Also a warning with information on the missing or faulty elements appears at the problematic field. If the configuration of the component is valid, after clicking the *Validate* button, a 'Component validation successful' notification is shown.
7. Save the component configuration by clicking the *Save* button.

The following values can be configured for the *Selector Brick*:

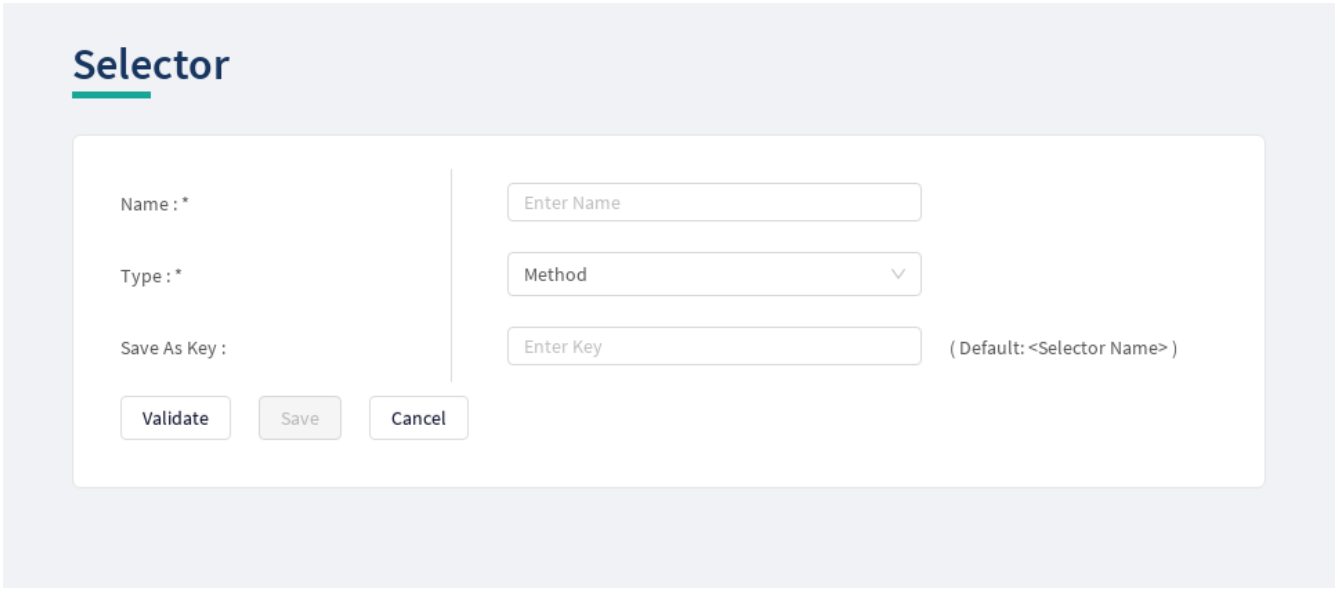


Figure 34. Configuring Selector in the Web User Interface

Table 23. Selector configuration options

Key	Values	Default value	Description
Name*	Free text. Alphanumeric, may contain underscores, may not start with a number.		The name of the parameter can be referenced.
Type*	Choose the selector type from the drop-down list. For more details on the values, see Extractor types.		Extractors are used to extract data from the call. They are utilized by Selector (and Matcher as well). Extractors are included by their type in Selectors, and are used by a special syntax in matchers. For details, see Extractors and Extractor types.
Save As Key	Text with ASCII characters. Space, '=', '"' and ']' are not allowed.	A special token, "<Selector Name>", which will use the Selector's name.	The key under which the results of a selector are saved in the *Insight* plugin's dictionary.

Depending on what value is selected for the *Type* parameter, additional parameters might appear for configuration. The following table provides details on these additional parameters.

Table 24. Additional Selector configuration options

Key	Values	Default value	Description
Save Under Key	True or False.	True	When set to False, returned dictionaries are merged into the *Insight* plugin's dictionary instead of being stored under a key. NOTE: this could lead to conflicting keys if multiple selectors would return the same key, in this case, keys could be overwritten.
Clear Text	True or False.	False	When turned on, whitespaces are stripped from the beginning and end of the result.
Namespaces	A list of key and expression pairs, in text.		The namespaces to use during extraction.
XPath Expression	A valid XPath expression in text.		A complete explanation on how to write XPath expressions is not in the scope of this document. • There is a tutorial. • There are examples. • There is also a formal specification.
JMESPath Expression*	A valid JMESPath expression in text.		A complete explanation on how to write JMESPath expressions is not in the scope of this document. To learn more about it visit the JMESPath website: • There is a tutorial. • There are examples. • There is also a formal specification.
Expression*			A regular expression specifies a set of strings that match it.

Key	Values	Default value	Description
Time Format	A valid time format string in text.	YYYY-MM-DDT HH:mm:ss.SSSSSSZZ (line breaks for display purposes only)	The time format to use, see: Timestamp format options.
Time Zone	A time zone specifier in text.	UTC	The name of the time zone, or the time zone offset. The time zone can be specified by using the name, for example, "Europe/Budapest", or as the time zone offset in +/-HH:MM format, for example, +01:00.
Source Plugin	Reference to a *Fraud Detector Plugin* or "Last".	Last: In case there are more Fraud Detector plugins defined in the Security Flow, by using this default value, the selector will use the score value provided for the last run Fraud Detector plugin.	The Fraud Detector plugin to be used in case there are more than one defined.
Plugin	Reference to a *Plugin* or "Previous".	Previous: A special value that dynamically selects the plugin that was most recently evaluated in the security flow.	The plugin from which data will be extracted.
Include Request Counter	True or False.	True	When turned on, the request counter is included in the Session ID. See [session-id] for details.

6.4.4. Insight Target

Insight Target bricks define where the data collected by the Insight will be sent to.

The **Insight Target** configuration tree contains named *Insight Targets* with their respective configuration.

See the *Insight Target* configuration options for the available *Insight Target* types and their configuration options.

6.4.4.1. Data flattening

To ensure compatibility with a wide range of *Insight Target* types, the results collected by the *Insight plugin* can be flattened. The path inside the complex data structure is encoded into the key for each value:

- The merged key describes the path to the value in the data structure as a string.
- The parts of the path will be separated by a separator character, forward slash by default ("/").
- Keys in nested dictionaries are added to the path by name.

- List items are added to the path by their index.

For example, take the following data structure:

```
{
  "a": 1,
  "b": 2,
  "c": [
    {
      "d": [2, 3, 4],
      "e": {
        "f": 5,
        "g": 6
      }
    }
  ]
}
```

This will be flattened to this:

```
{
  "a": 1,
  "b": 2,
  "c/0/d/0": 2,
  "c/0/d/1": 3,
  "c/0/d/2": 4,
  "c/0/e/f": 5,
  "c/0/e/g": 6
}
```

6.4.4.2. Configuring Insight Targets

1. Click on the *BRICKS* main navigation item in the Left navigation area. Alternatively you can also click on the  sign to open up the sub-navigation items of *BRICKS*.
2. Select *Insight Target*.

In the configuration window that appears, you can either see the empty parameter values that can be configured for the actual component or you can see already configured component(s) and their parameters. The already configured components with defined parameters can be default components available in the system by default, or can be components configured by the administrator:

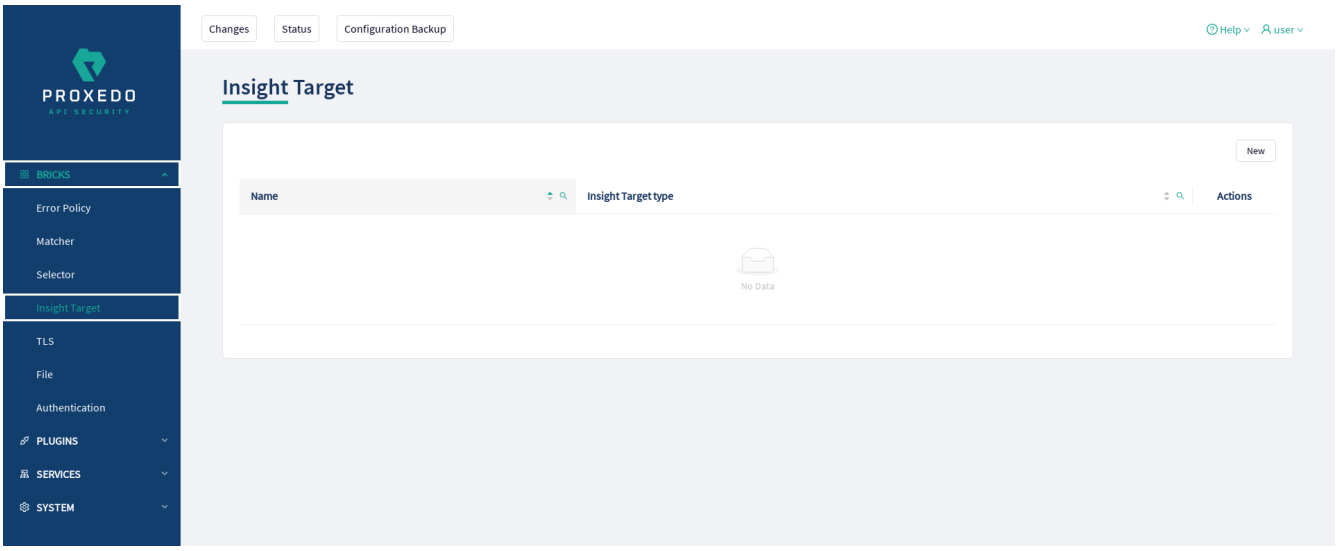


Figure 35. Insight Target main page in the Web User Interface

3. Click on the *New* navigation button to create an Insight Target.
4. Provide the name for your *Insight Target* configuration.
5. Select the *Insight Target* type.
6. Continue with the Syslog, Elastic and Local log configurations with the help of the following tables: Syslog *Insight Target* configuration parameters, Elastic *Insight Target* configuration parameters and Local log *Insight Target* configuration parameters.
7. Configure any more desired parameter details.
8. Click the *Validate* button to check if the defined parameters are of the correct type and all required parameters have been filled out for configuring the component. If the configuration is erroneous or incomplete, the Web UI provides a warning that the 'Component validation failed'. Also a warning with information on the missing or faulty elements appears at the problematic field. If the configuration of the component is valid, after clicking the *Validate* button, a 'Component validation successful' notification is shown.
9. Save the component configuration by clicking the *Save* button.

The following values can be configured for the *Insight Target Brick*:

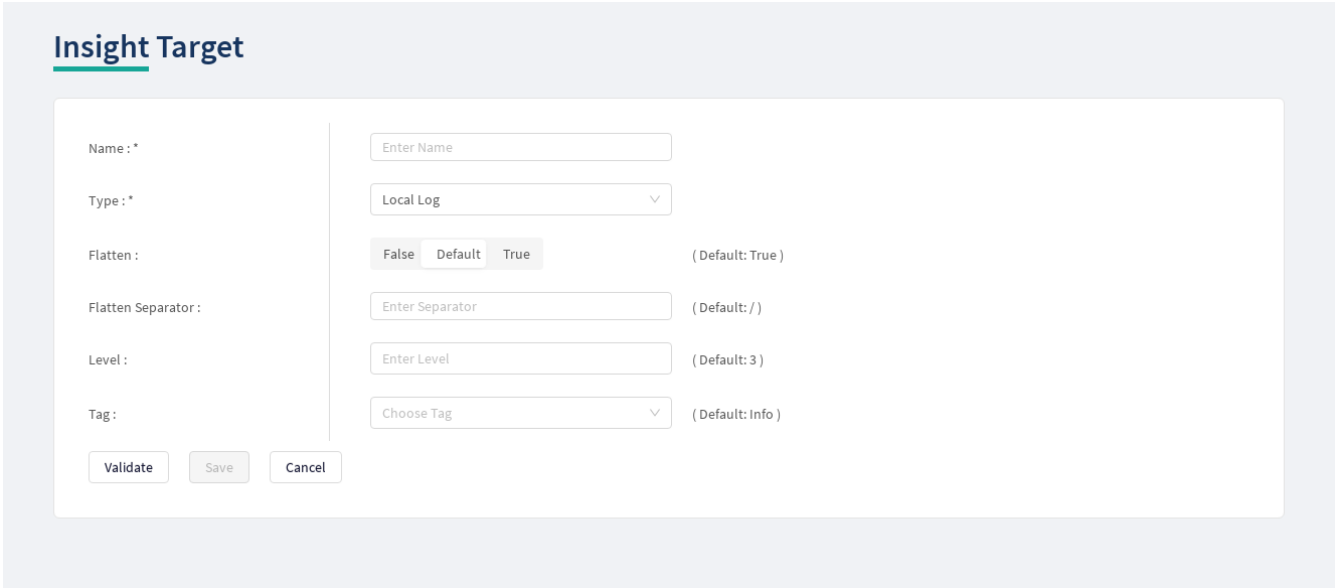


Figure 36. Configuring Insight Target in the Web User Interface

Table 25. Insight Target configuration options

Key	Values	Default value	Description
Name*	Free text. Alphanumeric, may contain underscores, may not start with a number.		The name identifying the *Insight Target*. This name of the *Insight Target* can be referenced from other parts of the configuration.
Type*	The values can be selected from the drop-down list. The available values are: • Local log • Syslog • Elastic		• Local log: Logs the result of the insight in the local system log. For more details on configuration settings with Local log, see Local log *Insight Target* configuration parameters. • Syslog: Sends the insight to a remote syslog server using the IETF syslog protocol defined in RFC5424. For more details on configuration settings with syslog, see table Syslog *Insight Target* configuration parameters. • Elastic: Sends the insight to an *Elasticsearch* engine in JSON. Since *OpenObserve* acts as a drop-in replacement for *Elasticsearch*, this target can be configured for *OpenObserve* instances. For more details on configuration settings, see Elastic *Insight Target* configuration parameters.
Flatten	True or False.	True	Flatten the *Insight Target* message.
Flatten Separator		/	The separator in the flattened message.
Level		3	The log level for the logged message.
Tag	The value can be selected from a drop-down list.	info	The log tag for the logged message.

The following table presents the configuration parameters for the Local log *Insight Target* type:

Table 26. Local log Insight Target configuration parameters

Key	Values	Default value	Description
Flatten	True or False.	True	Flatten the *Insight Target* message.
Flatten separator		/	The separator in the flattened message. Only visible when Flatten is True.
Level		3	The log level for the logged message.
Tag		info	The log tag for the logged message.

The following table presents the configuration parameters for the syslog *Insight Target* type:

Table 27. Syslog Insight Target configuration parameters

Key	Values	Default value	Description
Flatten	True or False.	True	Flattens the *Insight Target* message. Only visible when Data Format is JSON.
Flatten Separator		/	The separator in the flattened message. Only visible when Flatten is True.
Remote Connection	• Host: Hostname or IP address as text. • Port: The available values are integers. • Protocol: The available values are: TCP and UDP. • IP Protocol: The available values are: 4 and 6, corresponding to IPv4 and IPv6. • Use TLS: True or False. • **Syslog TLS***: Reference to a *TLS Brick* of type *Syslog TLS*.	• Protocol: TCP, Port: 601 (6514 if *Use TLS* is *True*.) • Protocol: UDP, Port: 514 • IP Protocol: 4 • Use TLS: False	• Host: The hostname or the IP address of the syslog server. • Port: Add the port number here to connect to the remote system. • Protocol: Add the transport protocol to send messages over. The available values are: TCP and UDP. • IP Protocol: The internet protocol version of the given driver. • Use TLS: It enables using TLS for the Syslog communication. • **Syslog TLS***: It is mandatory if the *Use TLS* option is set to True.
Flush Lines			It specifies how many lines are flushed to a destination at a time. The *Insights Director* waits for this number of lines to accumulate and sends them off in a single batch. Increasing this number increases the throughput, as more messages are sent in a single batch, but also increases the message latency.
Data Format	The possible values are: SData, JSON.	SData	This is the data format of the insight.
Include Message	True or False.	False	Whether to include the Insight plugin's Message field in the JSON output.
Message Key	Free text.	message	The key where the Insight plugin's Message field will appear in the JSON output.
Second Fraction Digits	Integer between 0 and 6 inclusive	3	The number of digits representing the fractions of seconds in the Syslog timestamp.
Time Zone	See table Time zones for time zone names.	GMT	The name of the time zone (for example, "Europe/Budapest") or the time zone offset in +/-HH:MM format (for example, +01:00).
Report Config Load	True or False.	False	It reports the event of a configuration being loaded with a cryptographic hash of the loaded configuration. This informs the *Insight Target* about changes in the configuration.

Key	Values	Default value	Description
Mask Credit Card Numbers	True or False.	False	It masks the middle section of recognised credit card numbers in any fields of the log message. Recognised credit cards are from one of the following issuers: American Express, Discover Card, Mastercard, VISA.
Enable Heartbeat	True or False.	False	It enables sending heartbeat (-- MARK --) messages to the *Insight Target*.
Heartbeat	• Frequency: A number greater than or equal to 1. • Mode: The possible values are: 'idle' (heartbeat messages are only sent when there is no traffic towards the *Insight Target*) and 'periodical' (heartbeat messages are sent regardless of activity).	• Frequency: 30 • Mode: 'periodical'	• Frequency: The number of seconds between heartbeat messages. • Mode: The operation mode of the heartbeat functionality.

The following table presents the configuration parameters for the Elastic *Insight Target* type:

Table 28. Elastic Insight Target configuration parameters

Key	Values	Default value	Description
Flatten	True or False.	True	It flattens the *Insight Target* message.
Flatten Separator		/	The separator in the flattened message. Only visible when Flatten is True.
Remote Connection			Settings related to the remote connection.
Include Message	True or False.	False	Whether to include the Insight plugin's Message field in the JSON output.
Message Key	Free text.	message	The key where the Insight plugin's Message field will appear in the JSON output.
Username*			The username to authenticate with on the servers.
Password*			The password to authenticate with on the servers.
Servers*	There are two values to be configured: • **Host***: The hostname or IP address of the Elasticsearch instance. • **Port**: The port on **host** to connect to. Defaults to 9200. (Add the values by clicking the '+' sign.)		The list of Elasticsearch servers. Messages will be load balanced between servers if multiple servers are given.

Key	Values	Default value	Description
Index*			The name of the index in the Elasticsearch instance.
Use TLS	True or False.	False	Enables using TLS in the connection towards the Elastic servers.
Elastic TLS*	Reference to a *TLS Brick* of type *Elastic TLS*.		The TLS configuration towards the Elastic servers. Mandatory if *Use TLS* is set to *True*.
Workers		4	The number of workers to use for communicating with the Elasticsearch servers. Should at least equal the number of servers.
Mask Credit Card Numbers	True or False.	False	It masks the middle section of recognised credit card numbers in any fields of the log message. Recognised credit cards are from one of the following issuers: American Express, Discover Card, Mastercard, VISA.

6.4.5. TLS

Transport Layer Security (TLS) is the cryptographic protocol that secures HTTPS communications. PAS can apply TLS encryption both when communicating with Clients and Backends. TLS encryption can also be used with *Syslog* and *Elastic Insight Targets*.

When HTTPS is used the *TLS* settings must be configured.

These parameters are used by the *Insight Director* and the *Transport Director*. For options that reference a file the path is relative to `/opt/balasys/var/persistent/` inside the *Transport Director* container. This directory is a docker volume and by default mounted from the `/opt/balasys/var/persistent/transport-director` directory in the host system.

6.4.5.1. Configuring TLS Bricks

1. Click on the *BRICKS* main navigation item in the Left navigation area. Alternatively you can also click on the  sign to open up the sub-navigation items of *BRICKS*.
2. Select *TLS*.

In the configuration window that appears, you can either see the empty parameter values that can be configured for the actual component or you can see already configured component(s) and their parameters. The already configured components with defined parameters can be default components available in the system by default, or can be components configured by the administrator:

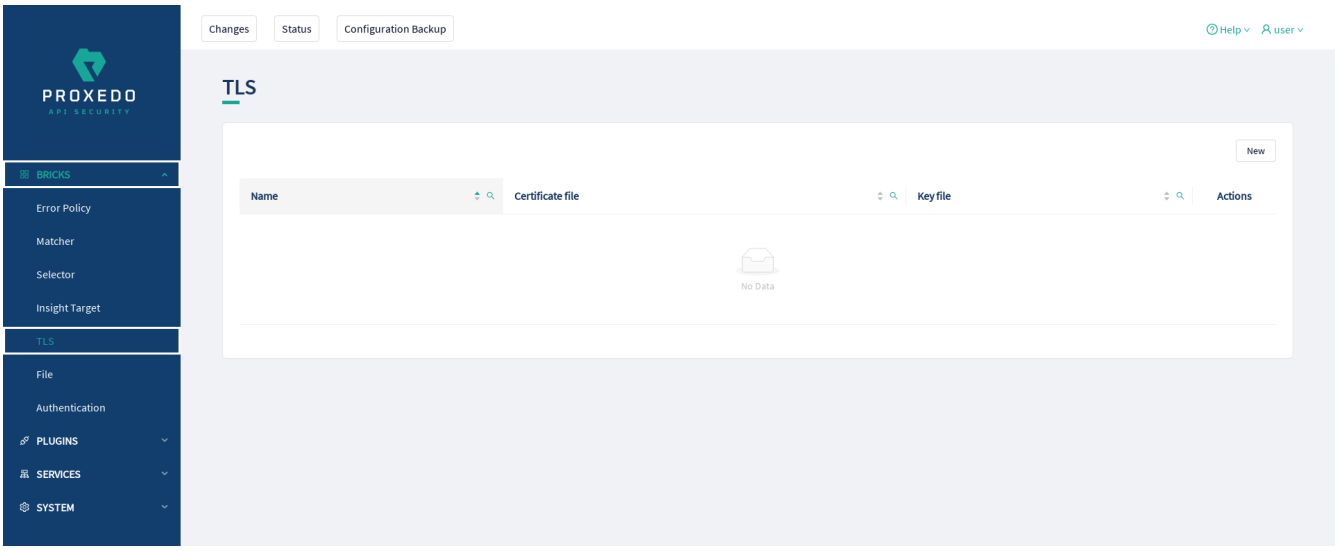


Figure 37. TLS main page in the Web User Interface

3. Click on the *New* navigation button to create a TLS.

The following values can be configured for the *TLS Brick*:

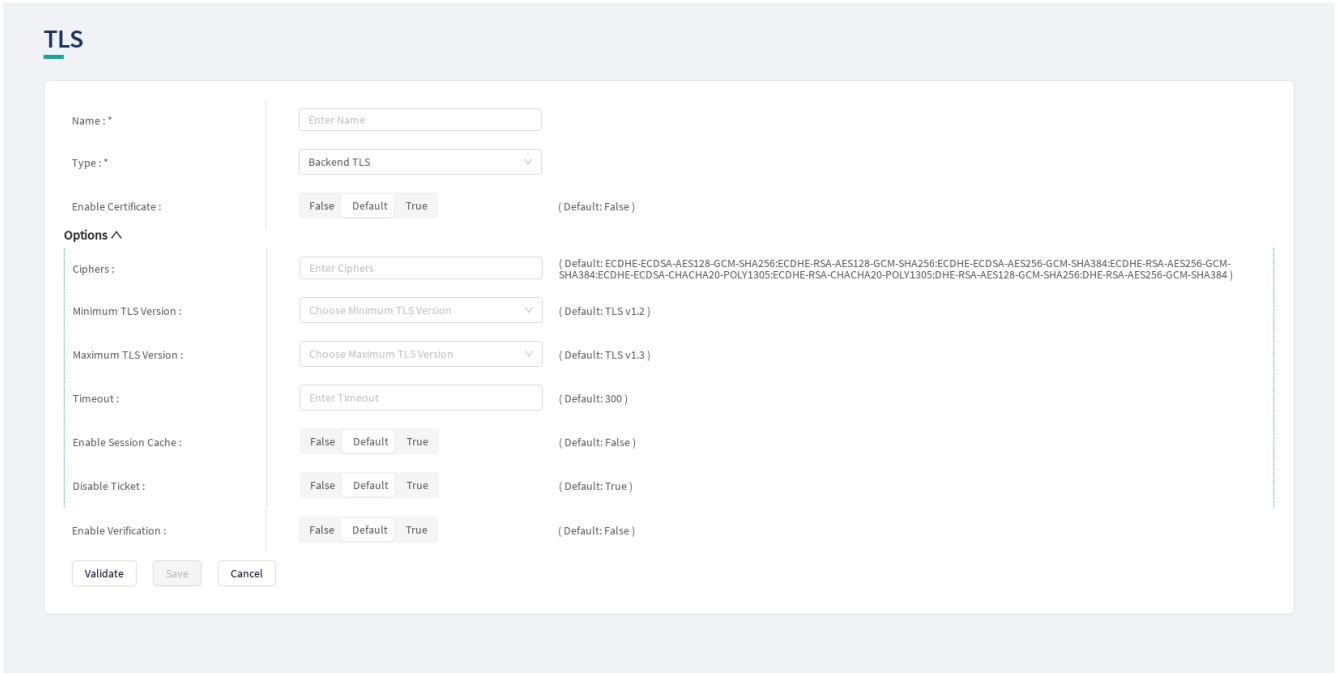


Figure 38. Configuring TLS in the Web User Interface

The configuration of the first two parameters determines the TLS type and from these two steps on, it is either a *Backend TLS* configuration, a *Client TLS* configuration, a *Syslog TLS* configuration or an *Elastic TLS* configuration.

6.4.5.1.1. Configuring Client TLS Bricks

The following parameters need to be configured for *Client TLS*:

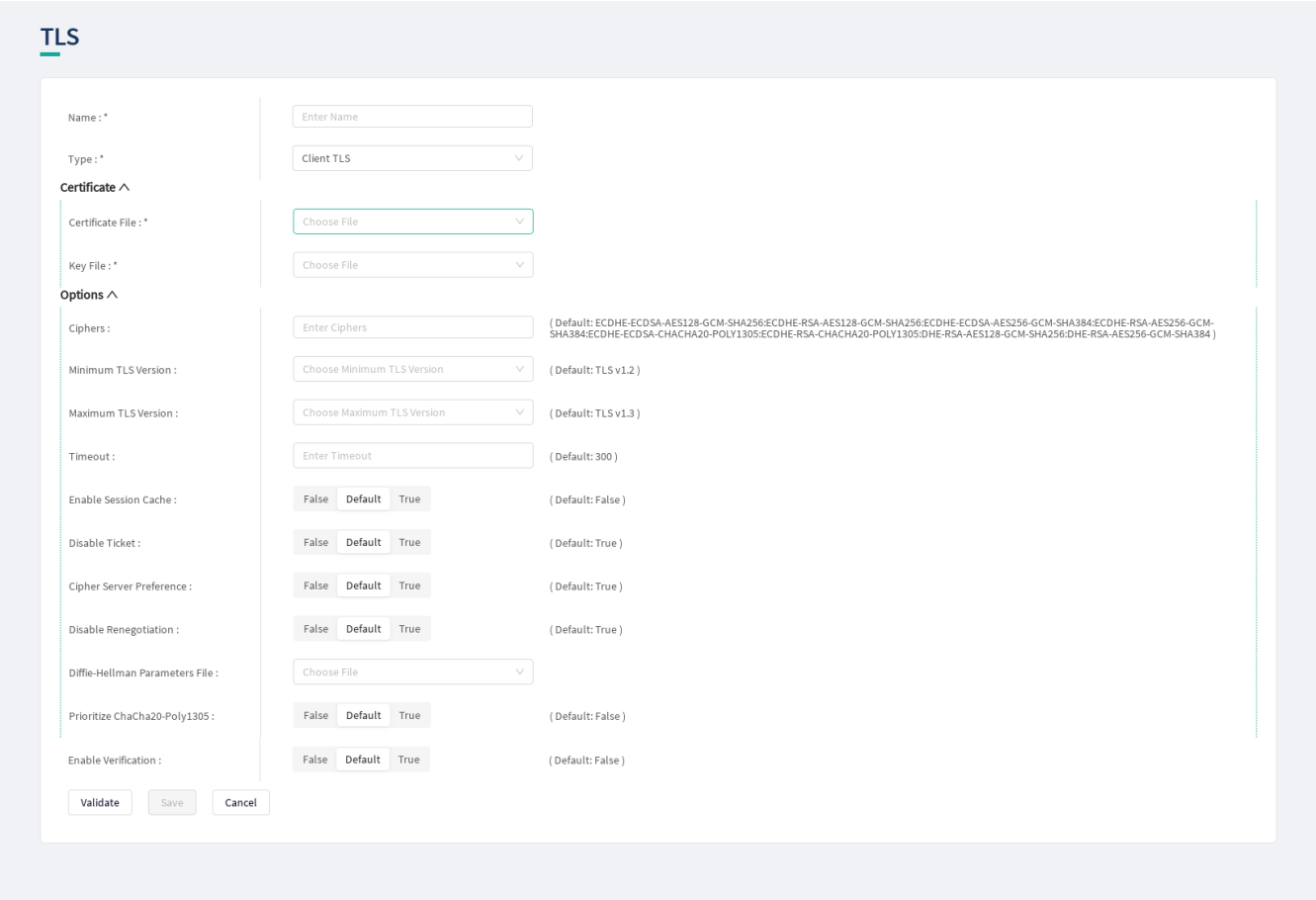


Figure 39. Configuring Client TLS in the Web User Interface, TLS options

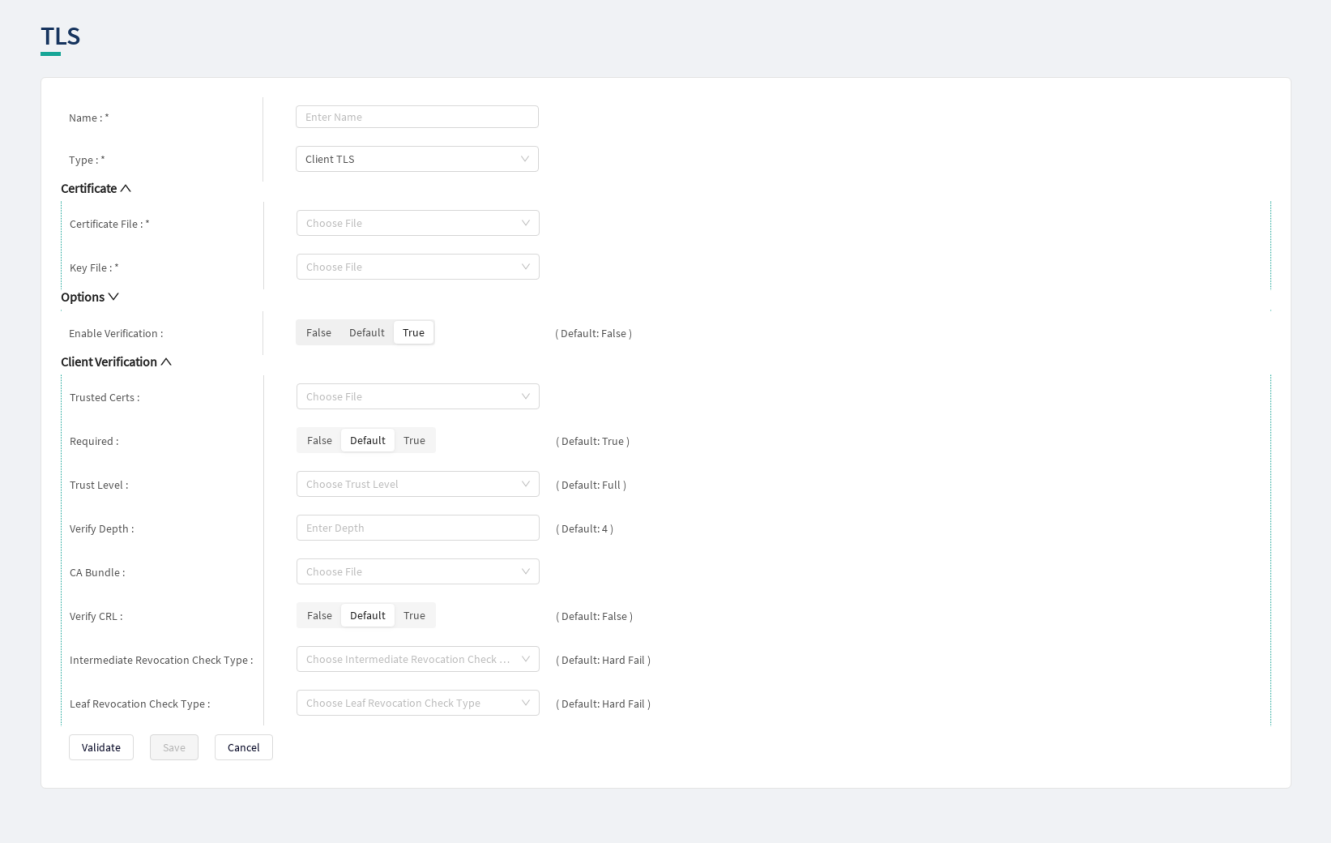


Figure 40. Configuring Client TLS in the Web User Interface, Certificate options

1. Name the Client TLS configuration.

2. Select the *Type* of the TLS, *Client TLS* in this case, from the drop-down list to configure TLS.

For details on these parameters, see the following table:

Table 29. TLS configuration

Key	Values	Default value	Description
Name*	Free text. Alphanumeric, may contain underscores, may not start with a number.		The name of the parameter can be referenced.
Type*	Choose the required value from the drop-down list.		Client TLS, Backend TLS, Syslog TLS and Elastic TLS configurations can be defined here.

3. Configure the mandatory parameters for *Client TLS*, based on the information provided in Table Client TLS configuration.

Table 30. Client TLS configuration

Key	Values	Default value	Description
Certificate			Configuration for the X.509 certificate used for TLS connections on the listener.
Certificate File*	Reference to a *File Brick* of type *Server Certificate*.		The certificate to be presented to clients.
Key File*	Reference to a *File Brick* of type *TLS Key*.		The private key corresponding to the certificate file.
Options			TLS protocol options used on the listener.
Ciphers		ECDHE-ECDSA-AES128-GCM-SHA256: ECDHE-RSA-AES128-GCM-SHA256: ECDHE-ECDSA-AES256-GCM-SHA384: ECDHE-RSA-AES256-GCM-SHA384: ECDHE-ECDSA-CHACHA20-POLY1305: ECDHE-RSA-CHACHA20-POLY1305: DHE-RSA-AES128-GCM-SHA256: DHE-RSA-AES256-GCM-SHA384	Specifies the allowed ciphers. Can be set to all, high, medium, low, or a string representation of the selected ciphers.

Key	Values	Default value	Description
Minimum TLS Version	Select one of the following options in the drop-down menu: • TLS v1.0 • TLS v1.1 • TLS v1.2 • TLS v1.3	TLS v1.2	The minimum version of TLS. Minimum TLS version must be less than or equal to the maximum TLS version.
Maximum TLS Version	Select one of the following options in the drop-down menu: • TLS v1.0 • TLS v1.1 • TLS v1.2 • TLS v1.3	TLS v1.3	The maximum version of TLS. Maximum TLS version must be greater than or equal to the minimum TLS version.
Timeout		300	It drops idle connection if the timeout value (in seconds) expires.
Enable Session Cache	True or False.	False	Store session information in the session cache. Set this option to 'On' to enable TLS session reuse.
Session Cache Size		20480	The number of sessions stored in the session cache for TLS session reuse.
Disable Ticket	True or False.	False	Session tickets are a method for TLS session reuse, described in RFC 5077. Set this option to 'On' to disable TLS session reuse using session tickets.
Cipher Server Preference	True or False.	True	Use server and not client preference order when determining which cipher suite, signature algorithm or elliptic curve to use for an incoming connection.
Disable Renegotiation	True or False.	True	Set this parameter *On* to disable client-initiated renegotiation.
Diffie-Hellman Parameters File	Reference to a *File Brick* of type *Diffie-Hellman Parameters*.		Contains the Diffie-Hellman parameters to be used by the TLS connection.
Prioritize ChaCha20-Poly1305	True or False.	False	Set this parameter *On* to prioritize using the ChaCha20-Poly1305 encryption.
Enable Verification	True or False.	False	It is an option for verifying client side X.509 certificates. By default no client verification takes place.
Client Verification			Client verification options

Key	Values	Default value	Description
Trusted Certs	Reference to a *File Brick* of type *Certificates*.		A directory where trusted IP addresses-certificate assignments are stored. When a peer from a specific IP address shows the certificate stored in this directory, it is accepted regardless of its expiration or issuer CA. Each file in the directory should contain a certificate in PEM format. The filename must be the IP address.
Required	True or False.	True	If it is set to True, PAS requires a certificate from the peer.
Trust Level	The values can be selected from the drop-down list. The available values are: • none • untrusted • full	full	It defines the trust level for certificate verification: • none: Accept even invalid certificates, for example self-signed certificates. • untrusted: Both trusted and untrusted certificates are accepted. • full: Only valid certificates signed by a trusted CA are accepted.
Verify Depth		4	The length of the longest accepted CA verification chain. PAS will automatically reject longer CA chains.
CA Bundle	Reference to a *File Brick* of type *CA Bundle*.		A bundle of trusted CA certificates and CRL files. CA certificates are loaded on-demand when PAS verifies the certificate of the peer.
Verify CRL	True or False.	False	If it is set to True, PAS checks the CRLs (Certificate Revocation Lists) associated with trusted CAs. CRLs will load automatically if PAS verifies the certificate of the peer.
Intermediate Revocation Check Type	The values can be selected from the drop-down list. The available values are: • none • soft_fail • hard_fail	hard_fail	The revocation check type for all certificates in the chain, except the Leaf Certificate: • none: Ignore the result certificate revocation status check • soft_fail: It fails if the check is successful and the certificate is revoked, it will pass otherwise • hard_fail: It passes only if the check is successful and the certificate is not revoked
Leaf Revocation Check Type	The values can be selected from the drop-down list. The available values are: • none • soft_fail • hard_fail	hard_fail	The revocation check types for the Leaf certificate are as follows: • none: With this option the result of the certificate revocation status check is ignored • soft_fail: It fails if the check is successful and the certificate is revoked, it passes otherwise • hard_fail: It passes only if the check is successful and the certificate is not revoked

4. Click the *Validate* button to check if the defined parameters are of the correct type and all required parameters have been filled out for configuring the component. If the configuration is erroneous or incomplete, the Web UI provides a warning that the 'Component validation failed'. Also a warning with information on the missing or faulty elements appears at the problematic field. If the configuration of the

component is valid, after clicking the *Validate* button, a 'Component validation successful' notification is shown.

5. Save the component configuration by clicking the *Save* button.

6.4.5.1.2. Configuring Backend TLS Bricks

The following parameters need to be configured for *Backend TLS*:

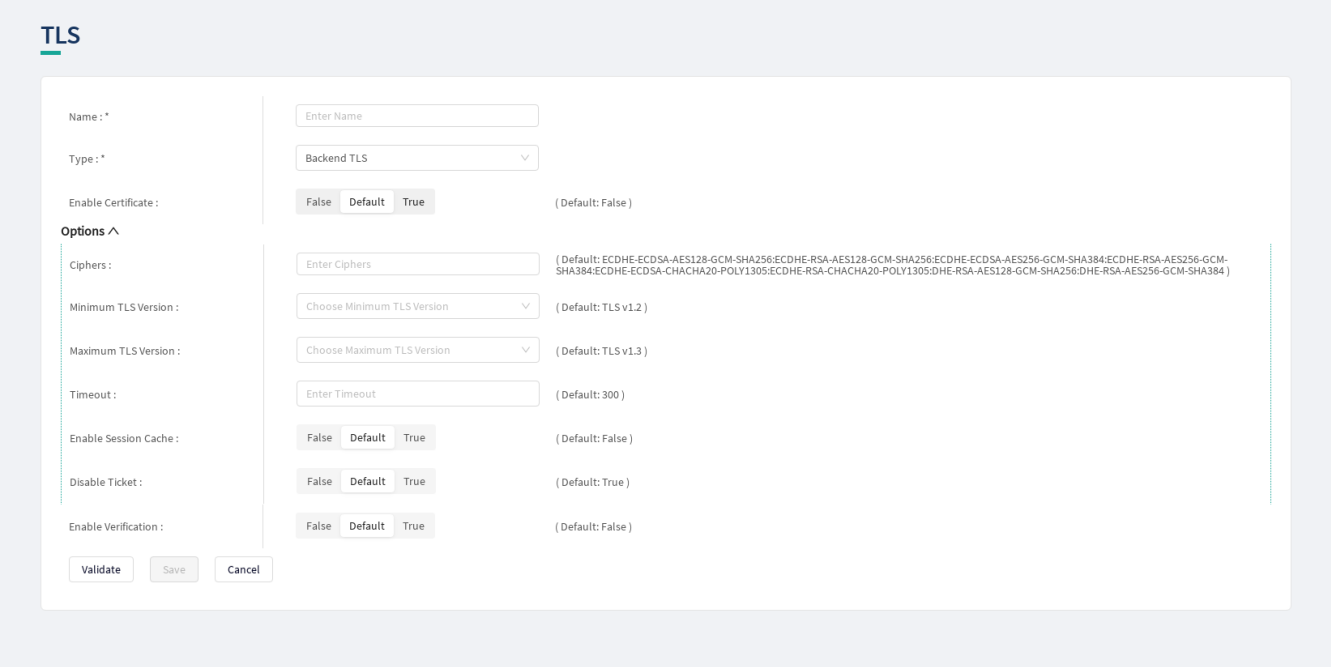


Figure 41. Configuring Backend TLS in the Web User Interface, TLS options

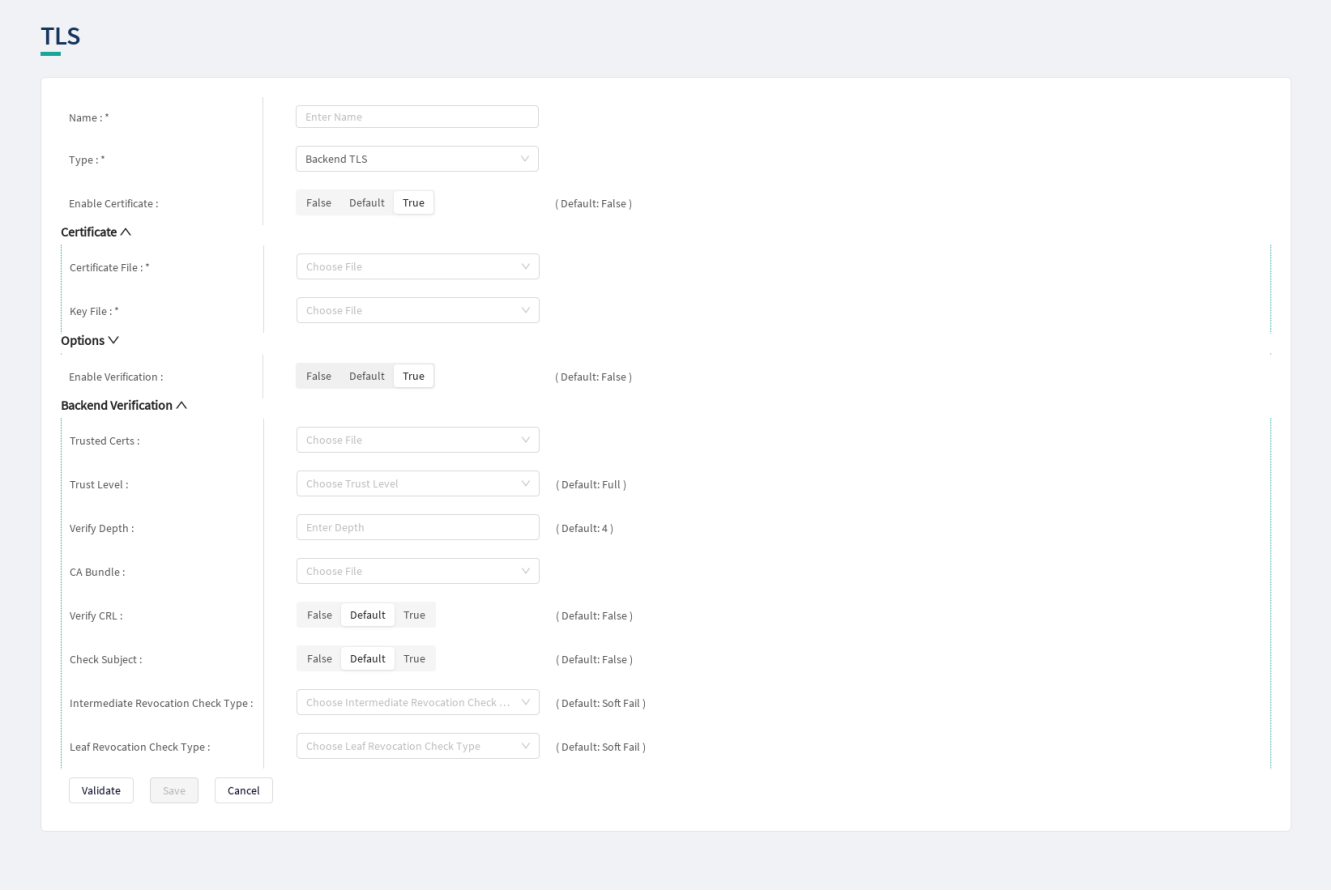


Figure 42. Configuring Backend TLS in the Web User Interface, Certificate options

1. Name the *Backend TLS* configuration.
2. Select *Backend TLS* from the drop-down list to configure *Backend TLS*.

For details on these parameters, see the following table:

Table 31. TLS configuration

Key	Values	Default value	Description
Name*	Free text. Alphanumeric, may contain underscores, may not start with a number.		The name of the parameter can be referenced.
Type*	Choose the required value from the drop-down list.		Client TLS, Backend TLS, Syslog TLS and Elastic TLS configurations can be defined here.

3. Configure the mandatory parameters for *Backend TLS*, based on the information provided in Table Backend TLS configuration.

The configuration parameters are described in details in the following table:

Table 32. Backend TLS configuration

Key	Values	Default value	Description
Enable Certificate	True or False.	False	It is an option for enabling backend side X.509 certificates. By default no backend verification takes place.
Certificate			Configuration for the X.509 certificate used for TLS connections on the listener.
Certificate File*	Reference to a *File Brick* of type *Client Certificate*.		The certificate to be presented to the backend.
Key File*	Reference to a *File Brick* of type *TLS Key*.		The private key corresponding to the certificate file.
Options			TLS protocol options used on the listener.

Key	Values	Default value	Description
Ciphers		ECDHE-ECDSA-AES128-GCM-SHA256: ECDHE-RSA-AES128-GCM-SHA256: ECDHE-ECDSA-AES256-GCM-SHA384: ECDHE-RSA-AES256-GCM-SHA384: ECDHE-ECDSA-CHACHA20-POLY1305: ECDHE-RSA-CHACHA20-POLY1305: DHE-RSA-AES128-GCM-SHA256: DHE-RSA-AES256-GCM-SHA384	Specifies the allowed ciphers. Can be set to all, high, medium, low, or a string representation of the selected ciphers.
Minimum TLS Version	Select one of the following options in the drop-down menu: • TLS v1.0 • TLS v1.1 • TLS v1.2 • TLS v1.3	TLS v1.2	The minimum version of TLS. Minimum TLS version must be less than or equal to the maximum TLS version.
Maximum TLS Version	Select one of the following options in the drop-down menu: • TLS v1.0 • TLS v1.1 • TLS v1.2 • TLS v1.3	TLS v1.3	The maximum version of TLS. Maximum TLS version must be greater than or equal to the minimum TLS version.
Timeout		300	It drops idle connection if the timeout value (in seconds) expires.
Enable Session Cache	True or False.	False	Store session information in the session cache. Set this option to 'On' to enable TLS session reuse.
Session Cache Size		20480	The number of sessions stored in the session cache for TLS session reuse.
Disable Ticket	True or False.	False	Session tickets are a method for TLS session reuse, described in RFC 5077. Set this option to 'On' to disable TLS session reuse using session tickets.

Key	Values	Default value	Description
Enable Verification	True or False.	False	It is an option for verifying Backend side X.509 certificates. By default no backend verification takes place.
Backend verification			Backend verification options
Trusted Certs	Reference to a *File Brick* of type *Certificates*.		A directory where trusted IP addresses-certificate assignments are stored. When a peer from a specific IP address shows the certificate stored in this directory, it is accepted regardless of its expiration or issuer CA. Each file in the directory should contain a certificate in PEM format. The filename must be the IP address.
Trust Level	The values can be selected from the drop-down list. The available values are: • none • untrusted • full	full	The trust level for certificate verification: • none: Accept even invalid certificates, for example self-signed certificates. • untrusted: Both trusted and untrusted certificates are accepted. • full: Only valid certificates signed by a trusted CA are accepted.
Verify Depth		4	It defines the length of the longest accepted CA verification chain. PAS will automatically reject longer CA chains.
CA Bundle	Reference to a *File Brick* of type *CA Bundle*.		A bundle of trusted CA certificates and CRL files. CA certificates are loaded on-demand when PAS verifies the certificate of the peer.
Verify CRL	True or False.	False	If it is set to True PAS checks the CRLs (Certificate Revocation Lists) associated with trusted CAs. CRLs will load automatically if PAS verifies the certificate of the peer.
Check Subject	True or False.	False	If it is set to, PAS compares the subject of the server-side certificate with application-layer information (for example, it checks whether the Subject matches the hostname in the URL).
Intermediate Revocation Check Type	The values can be selected from the drop-down list. The available values are: • none • soft_fail • hard_fail	soft_fail	The revocation check types for all certificates in the chain, except for the Leaf Certificate are as follows: • none: If this options is set, the certificate revocation status check results are ignored • soft_fail: If this option is set, the certificate revocation check fails, if the check is successful and the certificate is revoked. The check passes otherwise. • hard_fail: If this option is set, the check passes only if the check is successful, and the certificate is not revoked.

Key	Values	Default value	Description
Leaf Revocation Check Type	The values can be selected from the drop-down list. The available values are: • none • soft_fail • hard_fail	soft_fail	The revocation check type for the Leaf Certificate. • none: The result of the Certificate Revocation Status Check is ignored. • soft_fail: If this option is set, the certificate revocation check fails, if the check is successful and the certificate is revoked. The check passes otherwise. • hard_fail: If this option is set, the check passes only if the check is successful, and the certificate is not revoked.

4. Click the *Validate* button to check if the defined parameters are of the correct type and all required parameters have been filled out for configuring the component. If the configuration is erroneous or incomplete, the Web UI provides a warning that the 'Component validation failed'. Also a warning with information on the missing or faulty elements appears at the problematic field. If the configuration of the component is valid, after clicking the *Validate* button, a 'Component validation successful' notification is shown.
5. Save the component configuration by clicking the *Save* button.

6.4.5.1.3. Revocation checks for certificates

PAS tries to complete both CRL and OCSP-stapling checks for certificates.

The result for a certificate, according to the revocation check types is as follows:

Table 33. Certificate revocation checks

CRL check	OCSP stapling check	Soft fail result	Hard fail result
PASS	PASS	PASS	PASS
PASS	unsuccessful	PASS	PASS
unsuccessful	PASS	PASS	PASS
unsuccessful	unsuccessful	PASS	FAIL
PASS	FAIL	FAIL	FAIL
FAIL	PASS	FAIL	FAIL
unsuccessful	FAIL	FAIL	FAIL
FAIL	unsuccessful	FAIL	FAIL
FAIL	FAIL	FAIL	FAIL

6.4.5.1.4. Configuring Syslog TLS Bricks

The following parameters need to be configured for *Syslog TLS*:

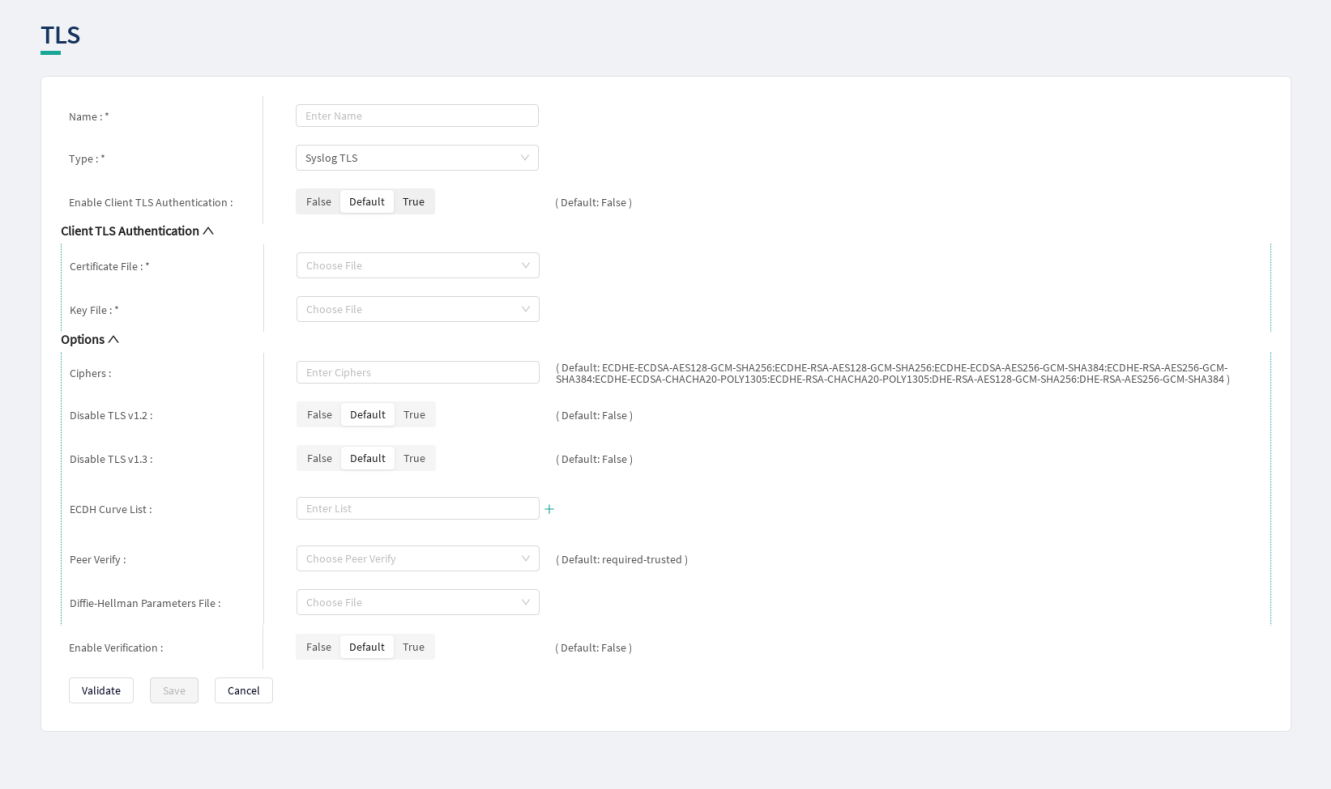


Figure 43. Configuring Syslog TLS in the Web User Interface

1. Name the Syslog TLS configuration.
2. Select the *Type* of the TLS, *Syslog TLS* in this case, from the drop-down list to configure TLS.

For details on these parameters, see the following table:

Table 34. TLS configuration

Key	Values	Default value	Description
Name*	Free text. Alphanumeric, may contain underscores, may not start with a number.		The name of the parameter can be referenced.
Type*	Choose the required value from the drop-down list.		Client TLS, Backend TLS, Syslog TLS and Elastic TLS configurations can be defined here.

3. Configure the mandatory parameters for *Syslog TLS*, based on the information provided in Table Syslog TLS configuration.

Table 35. Syslog TLS configuration

Key	Values	Default value	Description
Enable Client TLS Authentication	True or False.	False	Option for enabling TLS authentication towards the server.
Client TLS Authentication			Configuration for the X.509 certificate used for TLS connections on the *Insight Target*.
Certificate File*	Reference to a *File Brick* of type *Client Certificate*.		The certificate to be used for the connection.

Key	Values	Default value	Description
Key File*	Reference to a *File Brick* of type *TLS Key*.		The private key corresponding to the certificate file. The private key file must not be encrypted.
Options			TLS protocol options used on the *Syslog Insight target*.
Ciphers	Colon-separated list of ciphers from the list supported by OpenSSL 3.0.2.	ECDH+AESGCM: DH+AESGCM:EC DH+AES256: DH+AES256:ECD H+AES128: DH+AES:!aNULL :!MD5: !DSS!aNULL: !MD5: !DSS	Specifies the allowed ciphers.
Disable TLS v1.2	True or False.	False	Disables the usage of TLSv1.2 in the connection.
Disable TLS v1.3	True or False.	False	Disables the usage of TLSv1.3 in the connection.
ECDH Curve List	Add the names of one or more ECDH curves. The possible values are the ones supported by OpenSSL 3.0.2.	empty list	A list of curves permitted in the connection when using Elliptic Curve Cryptography (ECC).
Peer Verify	Select one of the following options in the drop-down menu: optional-trusted, optional-untrusted, required-trusted, required-untrusted	required-trusted	Defines the verification method of the peer. The four possible values are a combination of two properties of validation: whether the peer is required to provide a certificate (required or optional prefix), and whether the certificate provided needs to be valid (trusted or untrusted suffix).
Diffie-Hellman Parameters File	Reference to a *File Brick* of type *Diffie-Hellman Parameters*.		Contains the Diffie-Hellman parameters to be used by the TLS connection.
Enable Verification	True or False.	False	Option for enabling the verification of server side X.509 certificates.
Server Verification*			Server verification options are mandatory if *Enable Verification* is set to *True*.
CA Bundle	Reference to a *File Brick* of type *CA Bundle*.		A bundle of trusted CA certificates and CRL files. CA certificates are loaded on-demand when PAS verifies the certificate of the peer.
Verify CRL	True or False.	False	Verifies that certificates used in the connection are not revoked by any CRLs in the CA directory.

4. Click the *Validate* button to check if the defined parameters are of the correct type and all required parameters have been filled out for configuring the component. If the configuration is erroneous or incomplete, the Web UI provides a warning that the 'Component validation failed'. Also a warning with information on the missing or faulty elements appears at the problematic field. If the configuration of the component is valid, after clicking the *Validate* button, a 'Component validation successful' notification is shown.
5. Save the component configuration by clicking the *Save* button.

6.4.5.1.5. Configuring Elastic TLS Bricks

The following parameters need to be configured for *Elastic TLS*:

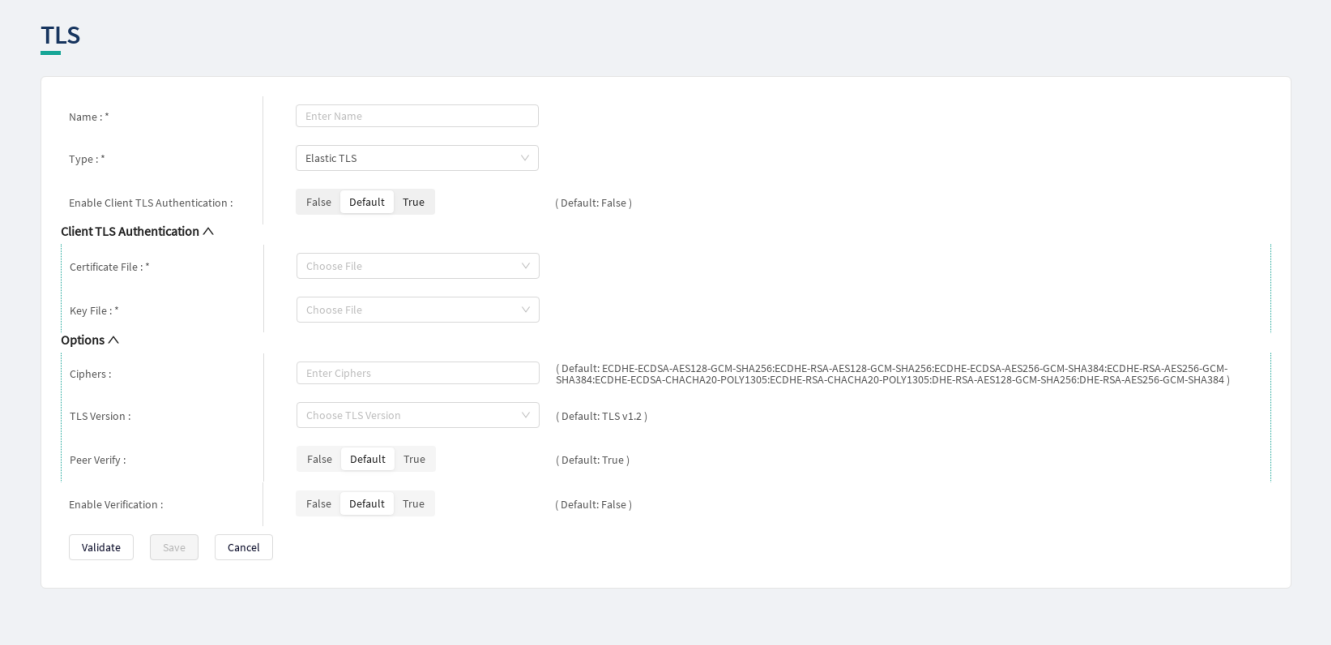


Figure 44. Configuring Elastic TLS in the Web User Interface

1. Name the Elastic TLS configuration.
2. Select the *Type* of the *TLS brick*, *Elastic TLS* in this case, from the drop-down list to configure the encryption used with the Elastic server.

For details on these parameters, see the following table:

Table 36. TLS configuration

Key	Values	Default value	Description
Name*	Free text. Alphanumeric, may contain underscores, may not start with a number.		The name of the parameter can be referenced.
Type*	Choose the required value from the drop-down list.		Client TLS, Backend TLS, Syslog TLS and Elastic TLS configurations can be defined here.

3. Configure the mandatory parameters for *Elastic TLS*, based on the information provided in Table Elastic TLS configuration.

Table 37. Elastic TLS configuration

Key	Values	Default value	Description
Enable Client TLS Authentication	True or False.	False	Option for enabling TLS authentication towards the server.
Client TLS Authentication			Configuration for the X.509 certificate used for TLS connections on the *Insight Target*.
Certificate File*	Reference to a *File Brick* of type *Client Certificate*.		The certificate to be used for the connection.

Key	Values	Default value	Description
Key File*	Reference to a *File Brick* of type *TLS Key*.		The private key corresponding to the certificate file. The private key file must not be encrypted.
Options			TLS protocol options used on the *Elastic Insight target*.
Ciphers	Colon-separated list of ciphers from the list supported by OpenSSL 3.0.2.	ECDH+AESGCM:DH+AESGCM:ECDH+AES256:DH+AES256:ECDH+AES128:DH+AES:!aNULL:!MD5:!DSS!aNULL:!MD5: !DSS	Specifies the allowed ciphers.
TLS Version	Select one of the following options in the drop-down menu: • TLS v1.0 • TLS v1.1 • TLS v1.2 • TLS v1.3	TLS v1.2	Defines the TLS version used in the connection.
Peer Verify	True or False.	True	Defines whether the peer is verified. If set to true, the peer is required to provide a certificate, and the certificate provided needs to be valid.
Enable Verification	True or False.	False	Option for enabling the verification of the X.509 certificate presented by the Elastic server.
CA Bundle	Reference to a *File Brick* of type *CA Bundle*.		A bundle of trusted CA certificates and CRL files. CA certificates are loaded on-demand when PAS verifies the certificate of the peer.

4. Click the *Validate* button to check if the defined parameters are of the correct type and all required parameters have been filled out for configuring the component. If the configuration is erroneous or incomplete, the Web UI provides a warning that the 'Component validation failed'. Also a warning with information on the missing or faulty elements appears at the problematic field. If the configuration of the component is valid, after clicking the *Validate* button, a 'Component validation successful' notification is shown.
5. Save the component configuration by clicking the *Save* button.

6.4.6. File

The *File* configuration element enables the administrator to upload files used by various plugins.

6.4.6.1. Configuring File Bricks

1. Click on the *BRICKS* main navigation item in the Left navigation area. Alternatively you can also click on the  sign to open up the sub-navigation items of *BRICKS*.
2. Select *File*.

In the configuration window that appears, you can either see the empty parameter values that can be configured for the actual component or you can see already configured component(s) and their parameters. The already

configured components with defined parameters can be default components available in the system by default, or can be components configured by the administrator:

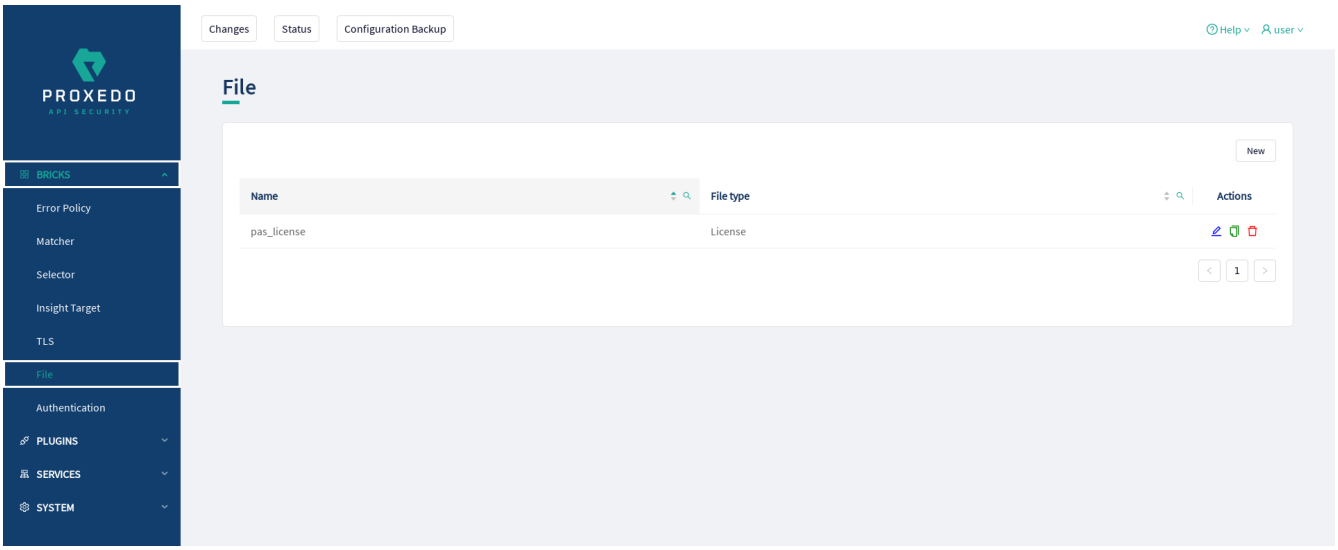


Figure 45. File main page in the Web User Interface

3. Click on the *New* navigation button to create a File Brick.
4. Choose the type of the file brick from the drop-down list.
5. Upload a file according to the selected type.
6. Click the *Validate* button to check if the defined parameters are of the correct type and all required parameters have been filled out for configuring the component. If the configuration is erroneous or incomplete, the Web UI provides a warning that the 'Component validation failed'. Also a warning with information on the missing or faulty elements appears at the problematic field. If the configuration of the component is valid, after clicking the *Validate* button, a 'Component validation successful' notification is shown.
7. Save the component configuration by clicking the *Save* button.

The following values can be configured for the *File Brick*:

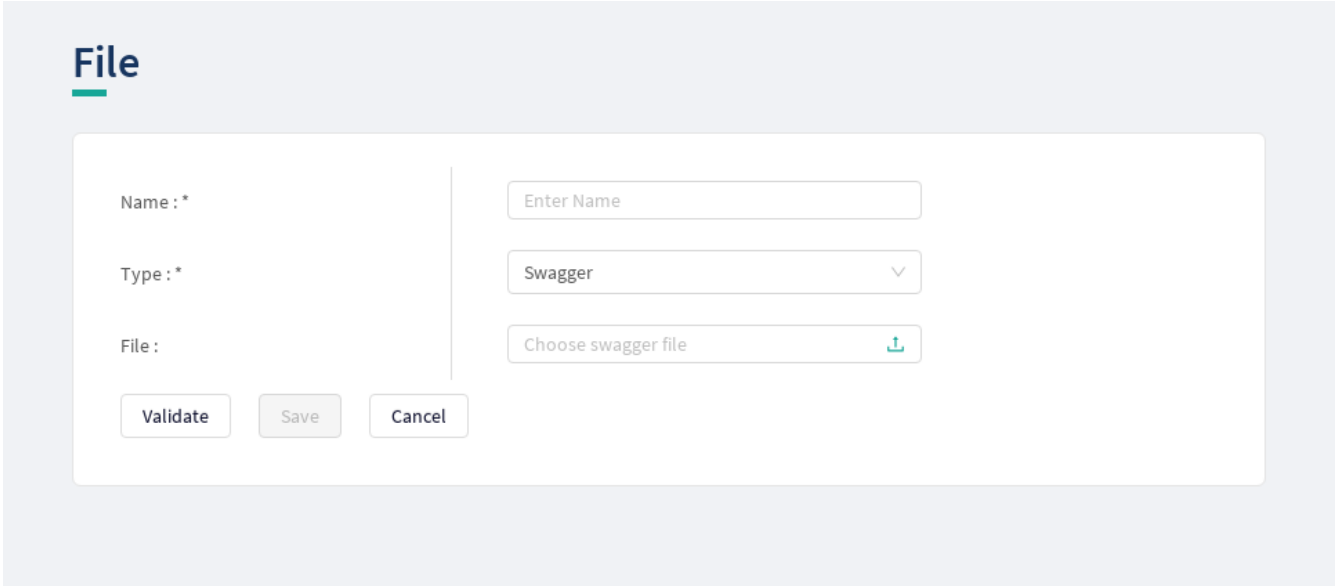


Figure 46. Configuring File in the Web User Interface

Table 38. File configuration parameters

Key	Values	Default	Description
Name*	Free text. Alphanumeric, may contain underscores, may not start with a number.		It defines the file-related configuration.
Type*	The available values are: • Swagger • OpenAPI 3.0 • OpenAPI 3.1 • XSD • WSDL • CA Bundle • Certificates • Diffie-Hellman Parameters • TLS Key • Client Certificate • Server Certificate • License See table Requirements for specific file types for specific requirements for each type.		The type selected here defines by which *PLUGINS* it can be used. The file uploaded here with the *Type Swagger*, for example, can be used by *Swagger* Plugins.
File*	The required file can be uploaded here.		
Passphrase	String value, could be empty.		*Only available for TLS Key files.* The passphrase to access an encrypted private key. Leave empty if the private key is unencrypted.

Table 39. Requirements for specific file types

File type	Requirements
CA Bundle	1. The file must be a flat ZIP file with the CA certificates inside. 2. It can contain copies of the certificates named following the `<hash>.0` format. The value of the `<hash>` part can be produced with the following command: `openssl x509 -noout -hash -in /path/to/cert/file`. **These copies will be generated automatically after saving if they are not present already, and the original File brick will be overwritten.** 3. It can contain CRL files, and it also can contain the copies of them following the `<hash_of_the_related_ca_file>.r0` format. The hash can be produced as described above. **These copies will be generated automatically after saving if they are not present already, and the original File brick will be overwritten.**

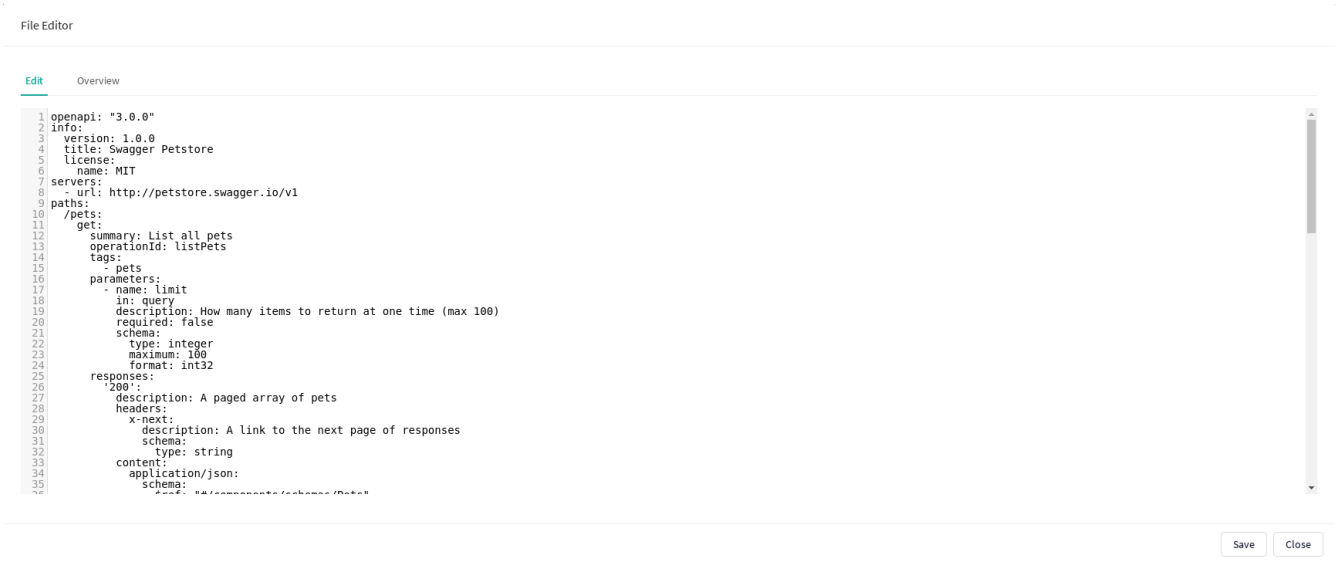
File type	Requirements
Certificates	1. The file must be a flat ZIP file with the certificates inside. 2. The certificates must be named after IPv4 or IPv6 addresses.
Diffie-Hellman Parameters	1. Must be in PEM format. 2. Must be a parameters file, such as one generated by the `openssl dhparam` utility.
TLS Key	1. Must be in PEM format. 2. Must be a private key file. 3. Could be encrypted or unencrypted. If the file is encrypted, the passphrase must be provided in the Passphrase field.
Client Certificate	1. Must be in PEM format. 2. Must be a certificate file. 3. Must have a Common Name attribute, and have the CLIENT_AUTH ExtendedKeyUsage.
Server Certificate	1. Must be in PEM format. 2. Must be a certificate file. 3. Must have a Common Name attribute, and have the SERVER_AUTH ExtendedKeyUsage.
License	1. Must be a ZIP file with a single `pas` directory, with a single `license.txt` inside. 2. Must be a valid PAS license. 3. Expirations are not validated. Limits are validated in integrity checks based on the *License File* brick selected in the *License* system.
Swagger	The file must be a Swagger schema as described in the OpenAPI 2.0 specification.
OpenAPI 3.0	The file must be an OpenAPI 3.0 schema as described in the OpenAPI 3.0 specification.
OpenAPI 3.1	The file must be an OpenAPI 3.1 schema as described in the OpenAPI 3.1 specification.
XSD	The file must be an XML Schema Definition as described in XML Schema Part 1: Structures, XML Schema Part 2: Datatypes, XSD 1.1 Part 1: Structures and XSD 1.1 Part 2: Datatypes.
WSDL	The file must be a WSDL service descriptor as described in the Web Services Description Language 1.1 specification or in the Web Services Description Language 2.0 specification.

File editor

Files in certain *File* brick types are editable when configuring the *File* brick. A *File editor* is available for the following types:

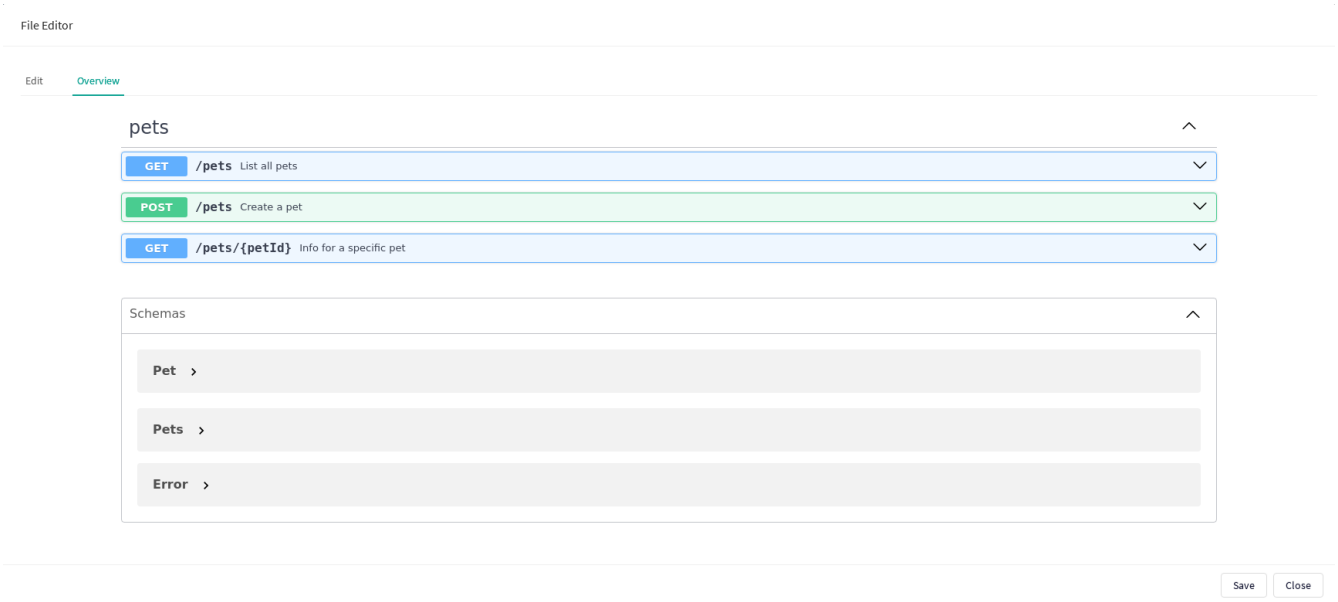
- Swagger
- OpenAPI 3.0
- OpenAPI 3.1

The uploaded file can be opened and edited by clicking the *Edit* button. The contents of the file open inside a new window, with the *Edit* tab selected:



The editor can be closed without saving any changes to the file with the *Close* button. The changes are saved and the editor is closed with the *Save* button.

The *Overview* tab shows errors if there are any, and the structure of the schema that the file describes:



6.4.7. Authentication

Authentications contain settings for ways to authenticate incoming calls.

The *Authentication* brick only defines the authentication method used for the call, but not the scope. For example, if two *Endpoints* share an *Authentication* brick, a user who authenticates on one *Endpoint* will still have to authenticate separately on the other *Endpoint*.

Authentication is based on OpenID Connect (OIDC), an authentication protocol based on OAuth 2.0. It requires an external OpenID Identity Provider. See the OpenID Foundation website for details. See the Authentication configuration options for the available *Authentication* configuration options.

6.4.7.1. Configuring Authentications

1. Click on the *BRICKS* main navigation item in the Left navigation area. Alternatively you can also click on the

sign to open up the sub-navigation items of *BRICKS*.

2. Select *Authentication*.

In the configuration window, you will either see empty parameter fields for the component you are configuring or a list of components with preconfigured parameters. These components may be system defaults or may have been configured by an administrator:

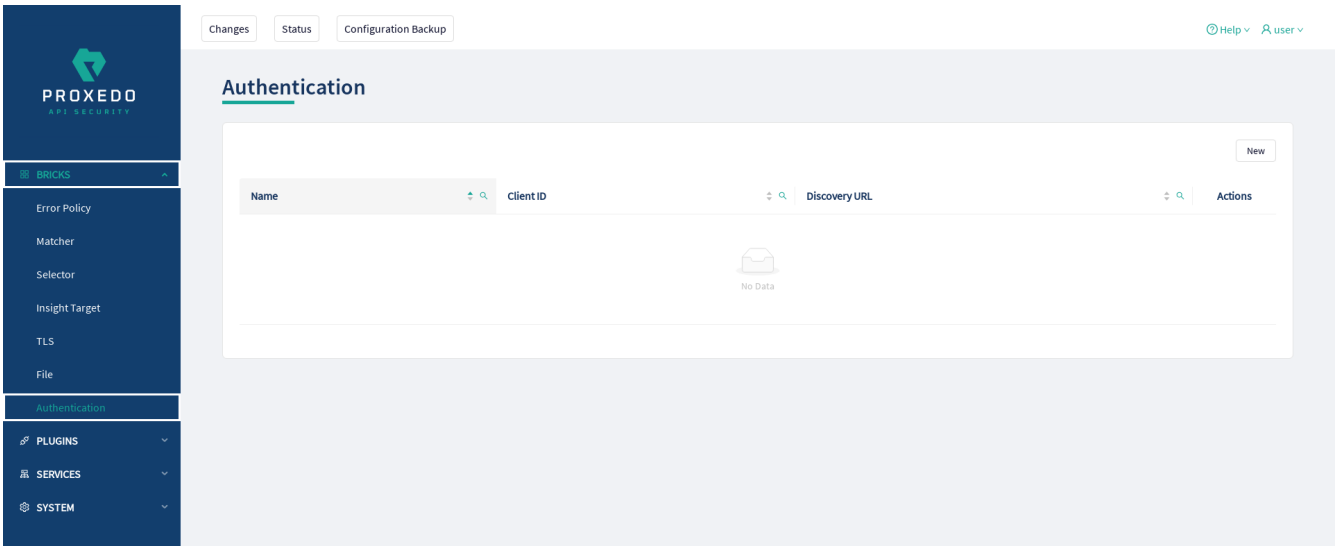


Figure 47. Authentication main page in the Web User Interface

3. Click on the *New* navigation button to create an Authentication.
4. Provide the name for your *Authentication* configuration.
5. Proceed with configuring OpenID Connect parameters. The list of available configuration options is described the following table: OpenID Connect *Authentication* configuration parameters.
6. Set up any optional parameters if needed.
7. Click the *Validate* button to check if the defined parameters are of the correct type and all required parameters have been filled out for configuring the component. If the configuration is erroneous or incomplete, the Web UI provides a warning that the 'Component validation failed'. Also a warning with information on the missing or faulty elements appears at the problematic field. If the configuration of the component is valid, after clicking the *Validate* button, a 'Component validation successful' notification is shown.
8. Save the component configuration by clicking the *Save* button.

The following values can be configured for the *Authentication Brick*:

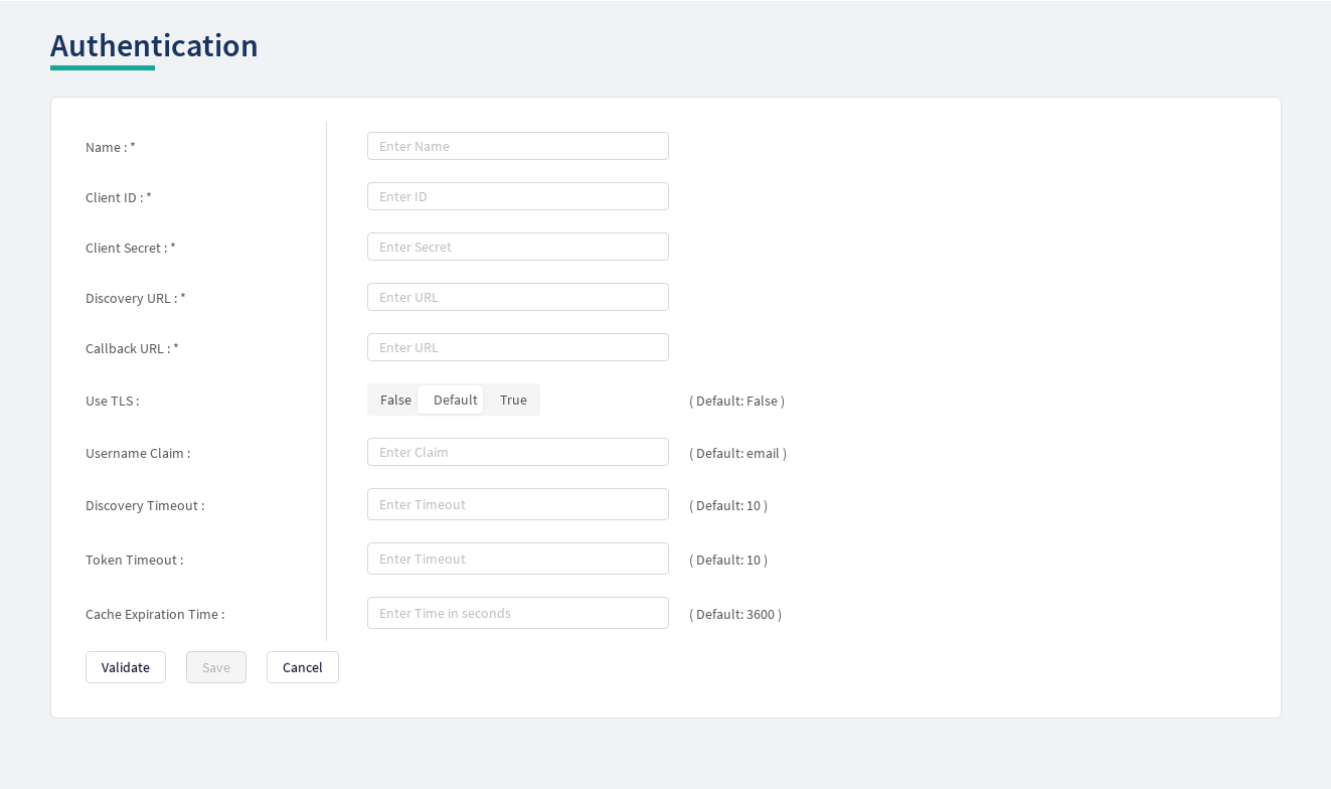


Figure 48. Configuring Authentication in the Web User Interface

Table 40. Authentication configuration options

Key	Values	Default value	Description
Name*	Free text. Alphanumeric, may contain underscores, may not start with a number.		The name identifying the *Authentication*. This name of the *Authentication* can be referenced from other parts of the configuration.

The following table describes the configuration parameters for the OpenID Connect *Authentication* type:

Table 41. OpenID Connect Authentication configuration parameters

Key	Values	Default value	Description
Client ID*	String.		The ID used to register PAS at the identity provider.
Client Secret*	String.		A secret provided by the identity provider.
Discovery URL*	URL.		The identity provider's configuration endpoint, usually ending in "/.well-known/openid-configuration".
Callback URL*	URL.		The Redirection Endpoint the identity provider should redirect to after successful authentication. NOTE: as this call contains authentication tokens, it is highly recommended to use an URL with HTTPS for this.
Use TLS	True or False.	False	Enables using TLS encryption when connecting to the identity provider.
CA Bundle*	Reference to a *File Brick* of type *CA Bundle*.		The CA bundle to use in validating TLS connections towards the provider servers. Mandatory if *Use TLS* is set to *True*.

Key	Values	Default value	Description
Username Claim	String.	email	The name of the claim in the ID token that contains the username used for authorization. In OpenID Connect, a claim is a piece of information asserted about a user, such as their email address, name, or unique identifier.
Discovery Timeout	Integer.	10	The maximum time, in seconds, to wait for a response from the configuration endpoint.
Token Timeout	Integer.	10	The maximum time, in seconds, to wait for a response from the token endpoint.
Cache Expiration Time	Integer.	3600	The duration, in seconds, for which authentication information is cached.

6.4.8. Common configuration elements for *BRICKS*

6.4.8.1. Extractors

Extractors are used to extract data from the call.

Extractors are not independent configuration components, but common configuration elements that are utilized by Matchers and Selectors. In fact, when configuring matchers and selectors, it is extractors that are listed at their type fields. Extractors are configured and used as part of matchers and selectors. There are no named extractors.

Most extractors return simple string values. However, some (might) return dictionaries. For example, you can get all the HTTP headers, or all the URI query parameters.

See the Extractor types for more details on extractors and their configuration options.

The following table provides details on extractor types:

Table 42. Extractor types

Key	Description
Method	Extracts the HTTP method of the request. It does not require configuration.
Status	Extracts the status code of the response. It does not require configuration.
JMESPath	Extracts data from the body of a JSON call with the help of a JMESPath expression. JMESPath is a query language for JSON. It is a very versatile tool for extracting the needed information from the body of the call, and organizing it according to requirements. A complete explanation on how to write JMESPath expressions is not in the scope of this document. To learn more about it visit the JMESPath website: • There is a tutorial. • There are examples. • There is also a formal specification.
Header	Extracts the value of an HTTP header. It is valid for some HTTP headers to be present more than once in a call. In this case, all the values are extracted as a list. It provides the name of the header in the configuration.
Header Force List	A *Header* extractor that returns a list even if there is only a single extracted value.

Key	Description
Header First	A *Header* extractor that only returns the **first** extracted value even if there is a list of extracted values.
Headers	The *Headers* extractor returns all the headers from the call. The results are stored as a dictionary, therefore it is recommended to set 'Save Under Key' to False if you use this from a Selector. It is valid for some HTTP headers to be present more than once in a call. In such cases all the values are stored under the header's key as a list. It does not require configuration.
Fraud Detector Score	Extracts the score value provided by the *Fraud Detector* plugin.
URI	Extracts the whole request URI as received from the client. It does not require configuration.
URI Netloc	Extracts the *network location* in the URI. It does not require configuration. It includes: • **username** and **password** if present • **host** • **port** if present unless scheme default  If the port is the default port for the scheme - that is 80 and 443 for HTTP and HTTPS, respectively - the port will not be included even if explicitly sent by the client. Therefore if the client used `http://example.com:80/path` then the *netloc* would be `http://example.com`, **not** `http://example.com:80`.
URI Origin	Extracts the *origin* part of the URI. It does not require configuration. It includes: • **scheme** • **host** • **port** if present, unless the default port for the scheme is used  If the port is the default port for the scheme - that is 80 and 443 for HTTP and HTTPS, respectively - the port will not be included, even if explicitly sent by the client. Therefore if the client used `http://example.com:80/path`, then the *origin* would be `http://example.com`, **not** `http://example.com:80`.
URI Scheme	Extracts the *scheme* of the request (http or https). It does not require configuration.
URI Username	Extracts the *username* in the request if present. It does not require configuration.
URI Password	Extracts the *password* in the request if present. It does not require configuration.
URI Host	Extracts the host in the request. It does not require configuration.
URI Port	Extracts the port of the request, the default port — that is 80 and 443 for HTTP and HTTPS, respectively — even if it is not not displayed explicitly in the request. It does not require configuration.

Key	Description
URI Path	Extracts the *path* part of the URI. It does not require configuration. The path is normalized to allow more robust matching and cleaner reporting. This means that: • If the path is missing `/` it is extracted. • Repeating forward-slash (`/`) characters are replaced with a single one. • dot (`.`) and double-dot (`..`) path segments are resolved. Consequently, if the path present in the *URI* was `//some/./nothere/../resource///./somewhere` the *path* would be `/some/resource/somewhere`. If you need to extract the *path* exactly as received, use URI raw path parameter.
URI Raw Path	Extracts the path part of the URI, without the normalization of URI path carried out. NOTE: If the *path* is missing a single forward slash ("/") is extracted. It does not require configuration.
URI Raw Query	Extracts the query part of the URI as a string. It does not require configuration.
URI Query	Extracts the query part of the URI. The results are stored as a dictionary, therefore it is recommended to set 'Save Under Key' to False if you use this from a Selector.
URI Query Param	Extracts the value of a query parameter. It is also valid for URIs to include a query parameter more than once. That is, it could be 'foo=bar&qux=quz&foo=baz'. In this case both values are extracted as a list. Provide the name of the parameter in the configuration.
URI Query Param Force List	An *URI Query Param* extractor that returns a list even if there is only a single extracted value.
URI Query Param First	An *URI Query Param* extractor that only returns the **first** extracted value even if there is a list of extracted values.
Client Address	Extracts the client's IP address.
Client Port	Extracts the client's port (TCP).
Server Address	Extracts the server's IP address.
Server Port	Extracts the server's port (TCP).
Parsed Content	Extracts the content. It does not require configuration.
Raw Content	Extracts the raw bytes of the request or response. It saves the results as a base64 encoded string.
Text Content	Extracts the request's or response's content as a decoded string.
Cookie	Extracts the values for a given key from the Cookie HTTP header. It is valid for multiple key-value pairs to be present in a Cookie header for the same key. In this case, all the values are extracted as a list. It requires the name of the Cookie key in the configuration.
Cookie Force List	A *Cookie* extractor that returns a list even if there is only a single extracted value.
Cookie First	A *Cookie* extractor that only returns the **first** extracted value even if there is a list of extracted values.

Key	Description
Cookies	The *Cookies* extractor returns all the key-value pairs from the Cookie header. The results are stored as a dictionary, therefore it is recommended to set 'Save Under Key' to False if you use this from a Selector. It is valid for multiple key-value pairs to be present in a Cookie header for the same key. In such cases, all the values are stored under the Cookie's key as a list. It does not require configuration.
Content Type	Extracts the content type from the HTTP header. It does not require configuration.
Content Type Charset	Extracts the charset from the content type HTTP header. It does not require configuration.
Call Direction	Extracts the call direction (request, response). It does not require configuration.
Session Id	Extracts the internal identifier of the HTTP session in keep-alive HTTP connections. Its 'Include request counter' option enables adding a request counter representing the number of requests in the session. See [session-id] for details.
Backend Response Time	Extracts the time spent between the sending the request towards the server and receiving the response from the server, in milliseconds. Only returns a value in a response flow.
Backend Name	Extracts the name of the *Backend Service* component handling the call.
Endpoint Name	Extracts the name of the *Endpoint Service* component handling the call.
Static	Extracts a string, integer, number, object, array, boolean as string from the configuration.
Timestamp	Extracts the current time. Also see the tables on Configuring timestamps and Timestamp format options.
Error Policy	Extracts *Request* or *Response* field of the *Error Policy* of the *Plugin* selected by the *plugin* field. If the *plugin* field contains the special value "Previous", the data will be extracted from the last evaluated plugin in the same HTTP request or response, if there is one. If the *plugin* field refers to a specific plugin instance, the data will be extracted from the last evaluation of the referred plugin instance. If the extractor is in the HTTP response, and the last evaluation is in the HTTP request, then the evaluation result from the HTTP request will be selected. Only plugins with a negative verdict will return data.
Error Policy Action	Works like the *Error Policy* extractor, but extracts only the *Request* or *Response* field of the referred plugin's *Error Policy*.
Error Policy Status Code	Works like the *Error Policy* extractor, but extracts only the *Request Code* or *Response Code* field of the referred plugin's *Error Policy*.
Error Policy Silent	Works like the *Error Policy* extractor, but extracts only the *Request Silent* or *Response Silent* field of the referred plugin's *Error Policy*.
Error Policy Message	Works like the *Error Policy* extractor, but extracts only the *Request Message* or *Response Message* field of the referred plugin's *Error Policy*.
Plugin Name	Works like the *Error Policy* extractor, but extracts the name of the referred plugin. Returns data regardless of verdict.
Plugin Verdict	Works like the *Error Policy* extractor, but extracts the verdict of the referred plugin. Returns data regardless of verdict.
Plugin Error Message	Works like the *Error Policy* extractor, but extracts the error message provided in the referred plugin's negative verdict, if there is one.

Key	Description
XPath	Extracts data from the body of an XML call with the help of a XPath expression. XPath is a query language for XML. It is a very versatile tool for extracting the needed information from the body of the call, and organizing it according to needs. A complete explanation on how to write XPath expressions is not in the scope of this document. To learn more about it visit the main website. Also see table XPath extractor configuration options. Provide the XPath expression in the configuration. Depending on the expression, the return value is a single node or a list of nodes. If you want a single value or a list independent from the expression, use *XPath First* or *XPath Force List*.
XPath Force List	Works like *XPath* but it returns a list even if there is only a single extracted value.
XPath First	Works like *XPath* but it only returns the first extracted value even if there is a list of extracted values.
SOAP Version	Extends the XPath extractor with predefined expressions. Extracts the SOAP message version. It identify with the SOAP namespace. Possible values: • soapv1_1 - the message version is SOAP v1.1 • soapv1_2 - the message version is SOAP v1.2
SOAP Envelope	Extends the XPath extractor with predefined expressions. Extracts the SOAP envelope.
SOAP Header	Extracts the SOAP header. Extends the XPath extractor with predefined expressions.
SOAP Body	Extracts the SOAP body. Extends the XPath extractor with predefined expressions.
SOAP Fault	Extracts the SOAP fault. Extends the XPath extractor with predefined expressions.
SOAP Fault Code	Extracts the SOAP fault 'code'. Extends the XPath extractor with predefined expressions. This extractor expression depends on the SOAP version. • **faultcode** - the SOAP v1.1 node tag • **Code** - the SOAP v1.2 node tag

Key	Description
SOAP Fault Detail	Extends the XPath extractor with predefined expressions. Extracts the SOAP fault 'detail'. This matcher expression depends on the SOAP version. • **Detail** - the SOAP v1.1 node tag • **Detail** - the SOAP v1.2 node tag
SOAP 1.1 Fault Faultstring	Extends the XPath extractor with predefined expressions. Extracts the SOAP fault 'faultstring'. This extractor only works with SOAP version 1.1.
SOAP 1.1 Fault Faultactor	Extends the XPath extractor with predefined expressions. Extracts the SOAP fault 'faultactor'. This extractor only works with SOAP version 1.1.
SOAP 1.2 Fault Reason	Extends the XPath extractor with predefined expressions. Extracts the SOAP fault 'Reason'. This extractor only works with SOAP version 1.2.
SOAP 1.2 Fault Node	Extends the XPath extractor with predefined expressions. Extracts the SOAP fault 'Node'. This extractor only works with SOAP version 1.2.
SOAP 1.2 Fault Role	Extends the XPath extractor with predefined expressions. Extracts the SOAP fault 'Role'. This extractor only works with SOAP version 1.2.

You can still use **Save As Key** for extractors returning dictionaries. For example, you can save all the headers under the `headers` key and the URI query parameters under the `parameters` key.

Timestamp extractors can be configured as follows:

Table 43. Configuring timestamps

Name	Default	Description
Time Zone	'UTC'	Set the time zone. • An *str* describing a time zone, similar to 'US/Pacific', or 'Europe/Berlin'. See: Time zones • An *str* in ISO 8601 style, as in '+07:00'. • An *str*, one of the following: 'local', 'utc', 'UTC'.
Time Format	YYYY-MM-DDT HH:mm:ss.SSSSSSZZ (line breaks for display purposes only)	Set the format. See: Timestamp format options
Use Request Start Time	True	If set to True, uses the request's start time. This value is set once for each call. If set to False, uses the time when the selector is processed during a session. This value can change every time the selector's value is queried during a call.

Table 44. Timestamp format options

Name	Token	Output
Year	YYYY YY	2000, 2001, 2002 … 2012, 2013 00, 01, 02 … 12, 13
Month	MMMM MMM MM M	January, February, March Jan, Feb, Mar 01, 02, 03 … 11, 12 1, 2, 3 … 11, 12
Day of Year	DDDD DDD	001, 002, 003 … 364, 365 1, 2, 3 … 364, 365
Day of Month	DD D Do	01, 02, 03 … 30, 31 1, 2, 3 … 30, 31 1st, 2nd, 3rd … 30th, 31st
Day of Week	dddd ddd d	Monday, Tuesday, Wednesday Mon, Tue, Wed 1, 2, 3 … 6, 7
Hour	HH H hh h	00, 01, 02 … 23, 24 0, 1, 2 … 23, 24 01, 02, 03 … 11, 12 1, 2, 3 … 11, 12
AM / PM	A a	AM, PM, am, pm am, pm
Minute	mm m	00, 01, 02 … 58, 59 0, 1, 2 … 58, 59
Second	ss s	00, 01, 02 … 58, 59 0, 1, 2 … 58, 59
Sub-second	S…	0, 02, 003, 000006, 123123123123 the result is truncated to microseconds, with half-to-even rounding
Time zone	ZZZ ZZ Z	Asia/Baku, Europe/Warsaw, GMT -07:00, -06:00 … +06:00, +07:00, +08, Z -0700, -0600 … +0600, +0700, +08, Z
Seconds Timestamp	X	1381685817, 1381685817.915482
ms or µs Timestamp	x	1569980330813, 1569980330813221

Table 45. XPath extractor configuration options

Key	Default	Description
xpath_expression		The expression to extract the node from the call to match against.
namespaces		Defines the XML namespaces.
clear_text	False	Whether to remove white spaces at the beginning and at the end of the string.

6.4.8.2. Comparators

Comparators are used for comparing the pattern with the result of the XPath expression.

Table 46. Types of comparators

Key	Description	Parameters
Equals	Matches if the parameter is exactly the same as the value matched. For matchers that work with numeric data type or with IP addresses it validates if the input is a valid number or IP address.	Ignorecase: Case differences (lower case, upper case) are ignored. When the present **VaLuE** would match **value**. For matcher types that work with numeric data type or with IP addresses, the 'Equals' and 'Not Equals' comparator types do not have ignorecase field.
Not equals	Matches if the parameter is not exactly the same as the value matched. For matchers that work with numeric data type or with IP addresses it validates if the input is a valid number or IP address.	Ignorecase: Case differences are ignored. When the present **VaLuE** would not match **vAlUe**. For matcher types that work with numeric data type or with IP addresses, the 'Equals' and 'Not Equals' comparator types do not have ignorecase field.
Starts with	Matches if the value starts exactly with the pattern.	Ignore case: Case differences are ignored. When the present **VaLuE** would match **value_given**.
Ends with	Matches if the value ends exactly with the pattern.	Ignore case: Case differences are ignored. When the present **VaLuE** would match **given_value**.
Contains	Matches if the exact pattern is found somewhere in the value.	Ignore case: Case differences are ignored. When the present **VaLuE** would match **some-value-given**.
Pattern	The Pattern treats the input as Unix shell-style wildcards. There are special characters used in shell-style wildcards: • '*' Matches everything. • '?' Matches a single character. • [seq] Matches any character in *seq*.  For a literal match, wrap the meta-characters in brackets. For example, `[?]` matches a literal question-mark (*?*).	Ignore case: Case differences are ignored. When the present **VaLuE** would match **some-value-given**.
Regex	*Regex* treats input as a regular expression for matching. Consult Python's regular expression documentation and their Regular Expression HOWTO.	• Ignore case: It sets the IGNORECASE flag for the regex. • Multiline: It sets the MULTILINE flag for the regex.
Minimum	Matches if the pattern is larger or equal to the value.	
Maximum	Matches if the pattern is smaller or equal to the value.	
Range	Matches if the value is between the limits in the pattern, including boundaries. The format of the pattern must be minimum..maximum.	

Key	Description	Parameters
Status class	Status class is a special comparator for conveniently matching HTTP status code classes in a *Status* matcher. It takes the name of the class and checks if the status code is in the given range as stated in Checking status code range.	
Subnet	The *subnet* comparator examines if an extracted IP address is in the specified subnet. The format for the input of the subnet comparator is the CIDR notation for IPv4 (for example, 192.0.2.0/24) and canonical prefix notation for IPv6 (for example, 2001:db8::/32).	
Error Policy Action	Matches the value in the *Request* or *Response* fields of an *Error Policy* brick.	
Plugin Verdict	Matches the verdict of a *Plugin* evaluation.	

Table 47. Checking status code range

Pattern	Status code range	Description
Info	1xx	Informational response
Success	2xx	Successful response
Redirect	3xx	Redirects
Client Error	4xx	Client Errors
Server Error	5xx	Server Errors

6.5. PLUGINS - Configuration units

A plugin is an element of the security flow that applies a specific security function. Plugins have different types based on the role they do:

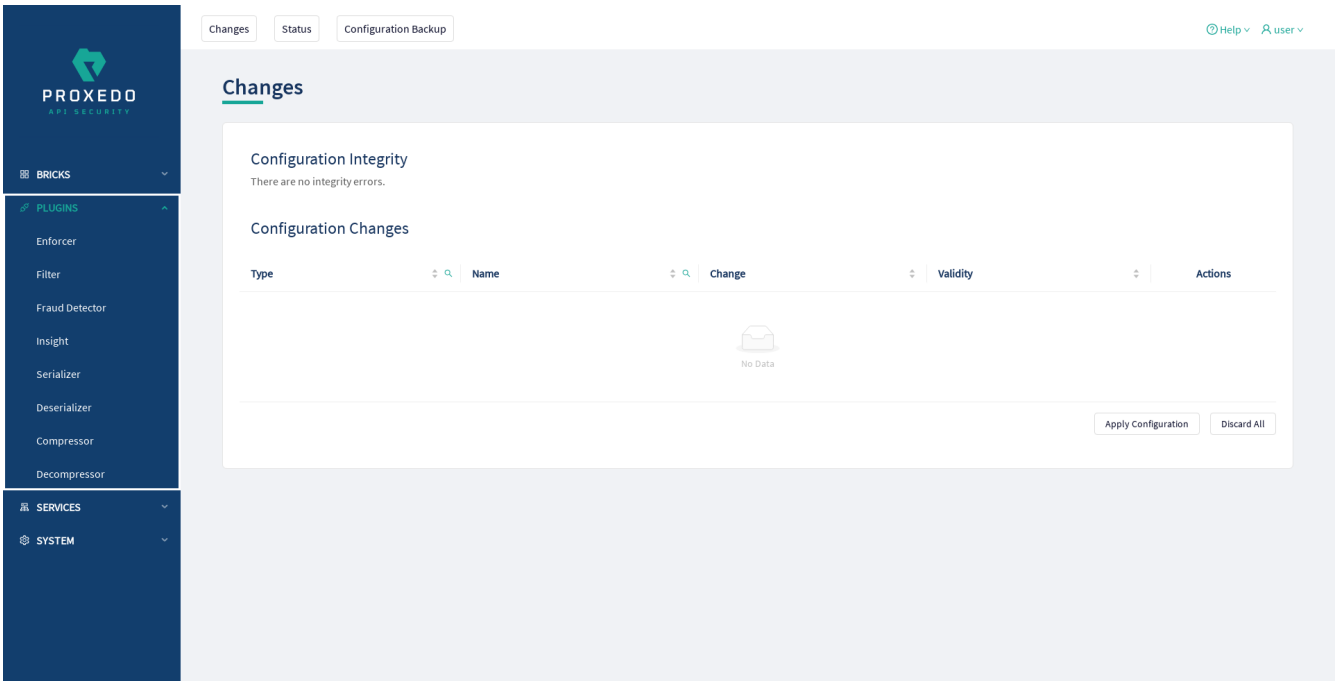


Figure 49. The PLUGINS main page in the Web User Interface

Plugins are named, so that they can be referenced in other parts of the configuration.

This means that *Plugin* configurations are reusable.

Certain Plugins are so called *default* objects, which are in 'read-only' state and cannot be configured or modified. Such default objects are listed in the following table:

Table 48. Default objects - PLUGINS

Default object name	Key
default_json	Serializer
default_xml	Serializer
default_json	Deserializer
default_xml	Deserializer
default	Compressor
default	Decompressor

6.5.1. Common Plugin parameters

Regardless of what plugins do, all plugins share some common parameters.

Table 49. Plugins' common parameters

Key	Values	Default value	Description
Matcher	Reference to a *Matcher Brick*.	Always: If the value is not defined, the plugin is always executed.	Optional parameter. It decides if the Plugin should be executed based on the call's details. If no matcher is configured the Plugin is always executed. For more details, see Matcher.

Key	Values	Default value	Description
Error Policy	Reference to an *Error Policy Brick*.		Optional parameter. It defines a custom error policy to be applied if the Plugin reports an error. The settings of the Error policy here override the Security Flow's default error policy. If no error policy is configured, the plugin type's default error policy is applied. For more details, see Error Policy.

Plugins are always named so that their names refer to a *Plugin* that represents a certain configuration. The names themselves are referenced from the Security Flow.

6.5.2. Enforcer

An *Enforcer Plugin* validates calls against externally defined schemas.

The *Plugin* supports validation against OpenAPI (Swagger) schemas, XSD schemas, WSDL schema or WAF ruleset.

Understanding the format of these schemas is not in the scope of this document. Further information is available at:

- The OpenAPI 2.0 format
- The OpenAPI 2.0 Specification
- The OpenAPI 3.0 format
- The OpenAPI 3.0 Specification
- The OpenAPI 3.1 Specification
- XSD 1.1 Specification
- XSD Tutorial
- WSDL Tutorial
- WSDL 1.1 Specification
- WSDL 2.0 Specification

The Enforcer Plugin uses its own default error policy, that is, the 'enforcer_default' error policy. The Plugin overrides the following fields of the default error policy:

Table 50. Default Enforcer Error Policy

Policy Setting	Default
request_code	422
request_message	Request Error

Problems are considered errors that lead to the termination of the call. Problems in the request are reported back to the client, while errors in the response are suppressed to avoid information leak.

See Error Policy to understand how defaults are applied.

6.5.2.1. Configuring Enforcer Plugins

1. Click on the *PLUGINS* main navigation item in the Left navigation area. Alternatively you can also click on the  sign to open up the sub-navigation items of *PLUGINS*.
2. Select *Enforcer* plugin.

In the configuration window that appears, you can either see the empty parameter values that can be configured

for the actual component or you can see already configured component(s) and their parameters. The already configured components with defined parameters can be default components available in the system by default, or can be components configured by the administrator:

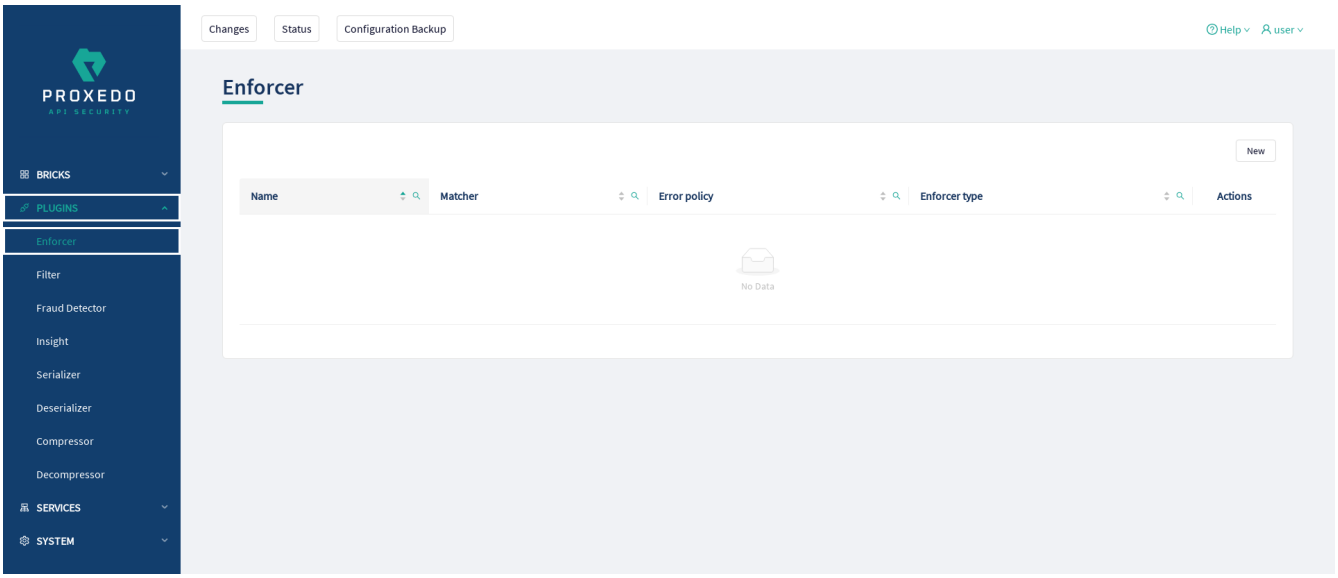


Figure 50. Enforcer Plugin's main page in the Web User Interface

3. Click on the *New* navigation button to create an Enforcer.
4. Name the *Enforcer* Plugin.
5. Choose the type of the *Enforcer* plugin.
6. Choose an *Error policy* from the drop-down list. The drop-down list will offer the error policy options configured under *BRICKS*.
7. Choose a *Matcher* from the drop-down list. The drop-down list will offer the matcher options configured under *BRICKS*.
8. Depending on the choice of the *Enforcer plugin* type selected earlier, different fields appear here for further configuration:
    - Swagger - Choose an uploaded Swagger file if the Enforcer type selected at the *Type* field was Swagger.
    - OpenAPI 3.0 - Choose an uploaded OpenAPI 3.0 file if the Enforcer type selected at the *Type* field was OpenAPI 3.0.
    - OpenAPI 3.1 - Choose an uploaded OpenAPI 3.1 file if the Enforcer type selected at the *Type* field was OpenAPI 3.1.
    - WSDL - Choose an uploaded WSDL file if the Enforcer type selected at the *Type* field was WSDL.
    - Operations - Fill in the *Operations* fields according to XSD enforcer plugin configuration options for Operations if the Enforcer type selected at the *Type* field was XSD.
    - Request Limit in Kilobytes - Fill in a number if you want to overwrite the default setting.
    - Harden Additional Properties Defaults - Choose the desired setting.
9. Click the *Validate* button to check if the defined parameters are of the correct type and all required parameters have been filled out for configuring the component. If the configuration is erroneous or incomplete, the Web UI provides a warning that the 'Component validation failed'. Also a warning with information on the missing or faulty elements appears at the problematic field. If the configuration of the component is valid, after clicking the *Validate* button, a 'Component validation successful' notification is shown.
10. Save the component configuration by clicking the *Save* button.

6.5.2.2. Swagger

The Swagger enforcer *Plugin* validates against OpenAPI 2.0 schemas.

The *Plugin* needs the schema definition file of the API Endpoint. This file must be in JSON or YML format.

6.5.2.3. OpenAPI 3.0

The OpenAPI 3.0 enforcer *Plugin* validates against OpenAPI 3.0 schemas.

The *Plugin* needs the schema definition file of the API Endpoint. This file must be in JSON or YML format.

6.5.2.4. OpenAPI 3.1

The OpenAPI 3.1 enforcer *Plugin* validates against OpenAPI 3.1 schemas.

The *Plugin* needs the schema definition file of the API Endpoint. This file must be in JSON or YML format.

6.5.2.5. XSD

XSD enforcer *Plugin* validates against XSD schemas. Both XSD 1.0 and 1.1 are supported.

As XSD enforcer requires parsed XML content an xml deserializer plugin needs to be included before XSD enforcer.

In the XSD enforcer you can define operations. Each operation contains criteria for identifying the call, and path of an XSD schema. If the HTTP message meets all criteria, its content will be validated using the schema.

XSD enforcer schema must contain at least one operation.

6.5.2.6. WSDL

WSDL enforcer *Plugin* validates against WSDL 1.0-2.0 schemas.

As WSDL enforcer requires parsed XML content, an xml deserializer plugin needs to be included before WSDL enforcer.

The Enforcer Plugin uses its own default error policy, that is, the 'enforcer_default' error policy. The Plugin overrides the following fields of the default error policy:

Table 51. Default Enforcer Error Policy

Policy Setting	Default
request_code	422
request_message	Request Error

Problems are considered errors that lead to the termination of the call. Problems in the request are reported back to the client, while errors in the response are suppressed to avoid information leak.

See Error Policy to understand how defaults are applied.

The plugin needs the schema definition file. This file must be in XML format.

WSDL schema validates request and response as well. Make sure that wsdl enforcer included in request and response flow as well.

In simple cases — when the listener/endpoint is serving a single version of a single API endpoint — a matcher is usually not needed as the schemas define all known URLs in the API. If however multiple API endpoints are consolidated under a single endpoint definition, you can define multiple enforcers each matching on a sub-path by using an URI path matcher and putting them all in the Security Flow.

6.5.2.7. WAF

The Web Application Firewall (WAF) enforcer *Plugin* protects against known attacks. The list of known attacks is updated by the WAF Ruleset Updater.

The following values can be configured for the *Enforcer Plugin*:

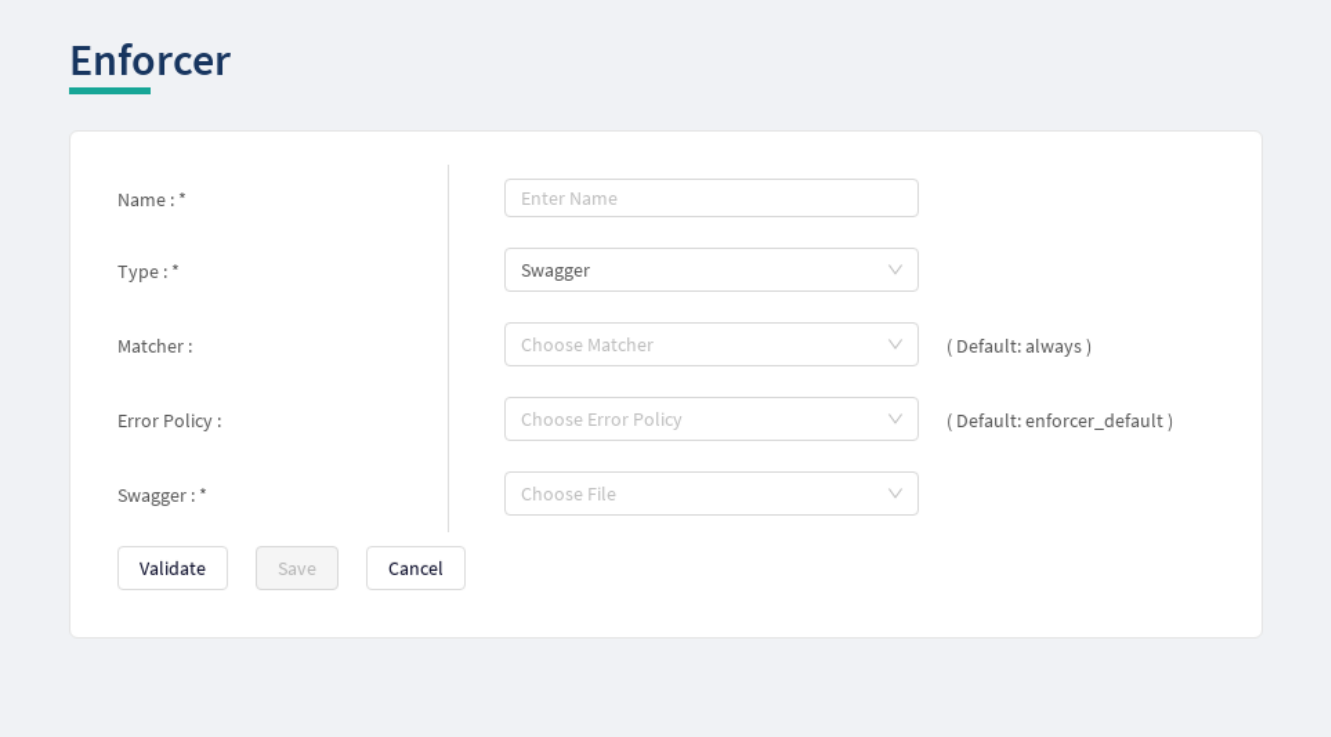


Figure 51. Configuring an enforcer plugin in the Web User Interface

Table 52. Enforcer Plugin's configuration options

Key	Values	Default value	Description
Name*	Free text. Alphanumeric, may contain underscores, may not start with a number.		This name identifies the Enforcer Plugin. The name of the plugin can be referenced from other parts of the configuration.

Key	Values	Default value	Description
Type*	Can be selected from the drop-down list. The available values are: • Swagger • OpenAPI 3.0 • OpenAPI 3.1 • XSD • WSDL • WAF		The type of the *Enforcer* plugin.
Matcher	Reference to a *Matcher Brick*.	Always: If the value is not defined the plugin is always executed.	It decides if the Plugin should be executed based on the call's details. For details see Matcher. If omitted the Plugin is always executed.
Error Policy	Reference to an *Error Policy Brick*.	enforcer_default	It defines a custom error policy to be applied if the Plugin reports an error. The settings of the Error policy here override the Security Flow's default error policy. For details see Error Policy.
Swagger*/OpenAPI 3.0*/OpenAPI 3.1*/WSDL*/Operations*	Depending on which type of the component was selected above, the following values are available: • For Swagger, OpenAPI 3.0, OpenAPI 3.1, WSDL, and XSD a reference to a *File Brick* of the appropriate type. • For XSD the configuration options for Operations can also be selected here. For details on parameters for Operations, see XSD enforcer plugin configuration options for Operations.		The Swagger enforcer *Plugin* validates against OpenAPI 2.0 schemas. The OpenAPI 3.0 enforcer *Plugin* validates against OpenAPI 3.0 schemas. The OpenAPI 3.1 enforcer *Plugin* validates against OpenAPI 3.1 schemas. WSDL enforcer *Plugin* validates against WSDL 1.0-2.0 schemas. XSD enforcer *Plugin* validates against XSD schemas.
Harden Additional Properties Defaults	True or False.	False	*Only available for OpenAPI 3.0 and OpenAPI 3.1 enforcers.* If set to True, the Enforcer will check calls as if the default value of `additionalProperties` would be False for Schema Objects, triggering the error policy if a non-specified property is present in the call, unless `additionalProperties=True` is explicitly set on the object. If set to False, the original behavior of OpenAPI where `additionalProperties` defaults to True is retained.

Key	Values	Default value	Description
Request Limit in Kilobytes		100000	*Only available for WAF enforcers.* It defines the size limit for requests in kilobytes.

XSD has the following configuration options for the *Operations* parameters:

Table 53. XSD enforcer plugin configuration options for Operations

Key	Default	Description
URI Path	*	The pattern for uri_path.
Choose Method		The method of the HTTP message. The following values are available for *Method*: • get • head • post • put • delete • connect • options • trace • patch
Status		The status of the HTTP message.
Choose Call Direction		The direction of the message, which must be either request or response.
Choose File		The XSD schema.

6.5.3. Filter

Filter Plugins are lightweight alternatives of *Enforcer Plugins* for filtering unwanted traffic. They only consist of a matcher and an error policy. If the matcher matches, the error policy is applied. This way you can use matchers inline, instead of creating a whole schema-based *Enforcer Plugin* for the simple use cases.

6.5.3.1. Configuring Filter Plugins

1. Click on the *PLUGINS* main navigation item in the Left navigation area. Alternatively you can also click on the  sign to open up the sub-navigation items of *PLUGINS*.
2. Select *Filter* plugin.

In the configuration window that appears, you can either see the empty parameter values that can be configured for the actual component or you can see already configured component(s) and their parameters. The already configured components with defined parameters can be default components available in the system by default, or can be components configured by the administrator:

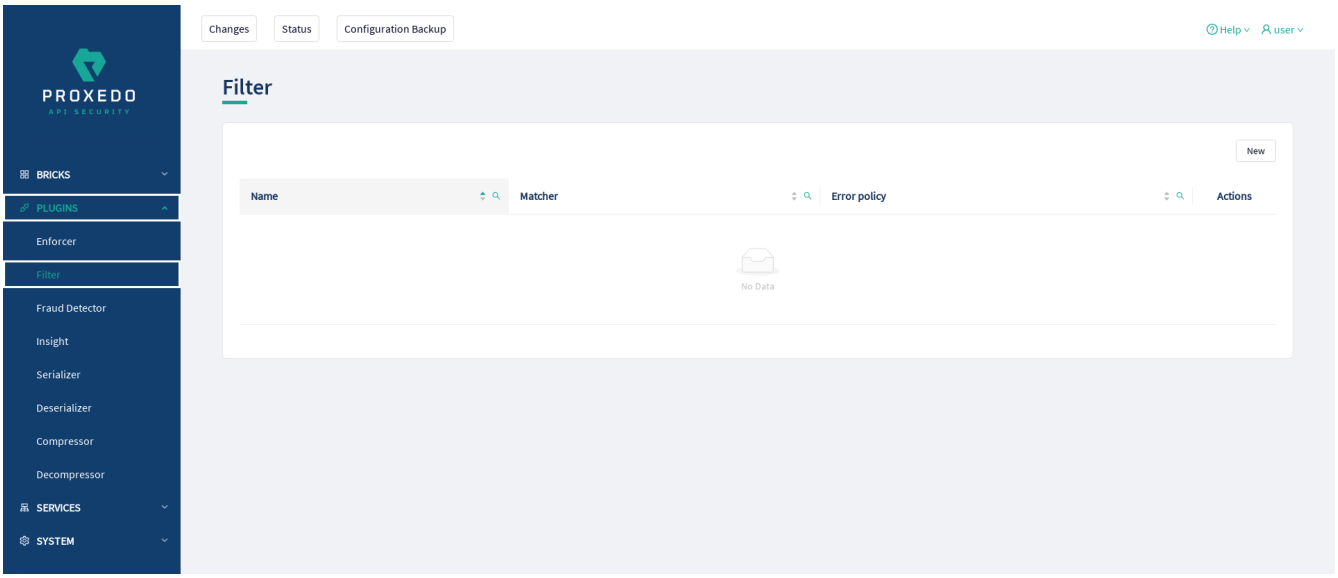


Figure 52. Filter Plugin's main page in the Web User Interface

Make sure that any component referenced in the configuration of this component, for example an Error policy or a Matcher selected from the drop-down lists, must remain part of the configuration later as well. Removing any of the referenced components might lead to invalid configuration.

3. Click on the *New* navigation button to create a Filter.
4. Add the name of the Filter Plugin.
5. Add the Body content for the error policy. (Optional)
6. Define the Content type.
7. Choose an error policy from the drop-down list. (Optional)
8. Choose a matcher from the drop-down list. (Optional)
9. Click the *Validate* button to check if the defined parameters are of the correct type and all required parameters have been filled out for configuring the component. If the configuration is erroneous or incomplete, the Web UI provides a warning that the 'Component validation failed'. Also a warning with information on the missing or faulty elements appears at the problematic field. If the configuration of the component is valid, after clicking the *Validate* button, a 'Component validation successful' notification is shown.
10. Save the component configuration by clicking the *Save* button.

The Plugin does not override any of the default error policy options.

Problems are considered errors that lead to the termination of the call. Problems in the request are reported back to the client, while errors in the response are suppressed to avoid information leak.

See Error Policy to understand how defaults are applied.

If you omit the matcher, the *Plugin* will always be executed. For *Filter plugins* this means aborting **all** calls.

The following values can be configured for the *Filter Plugin*:

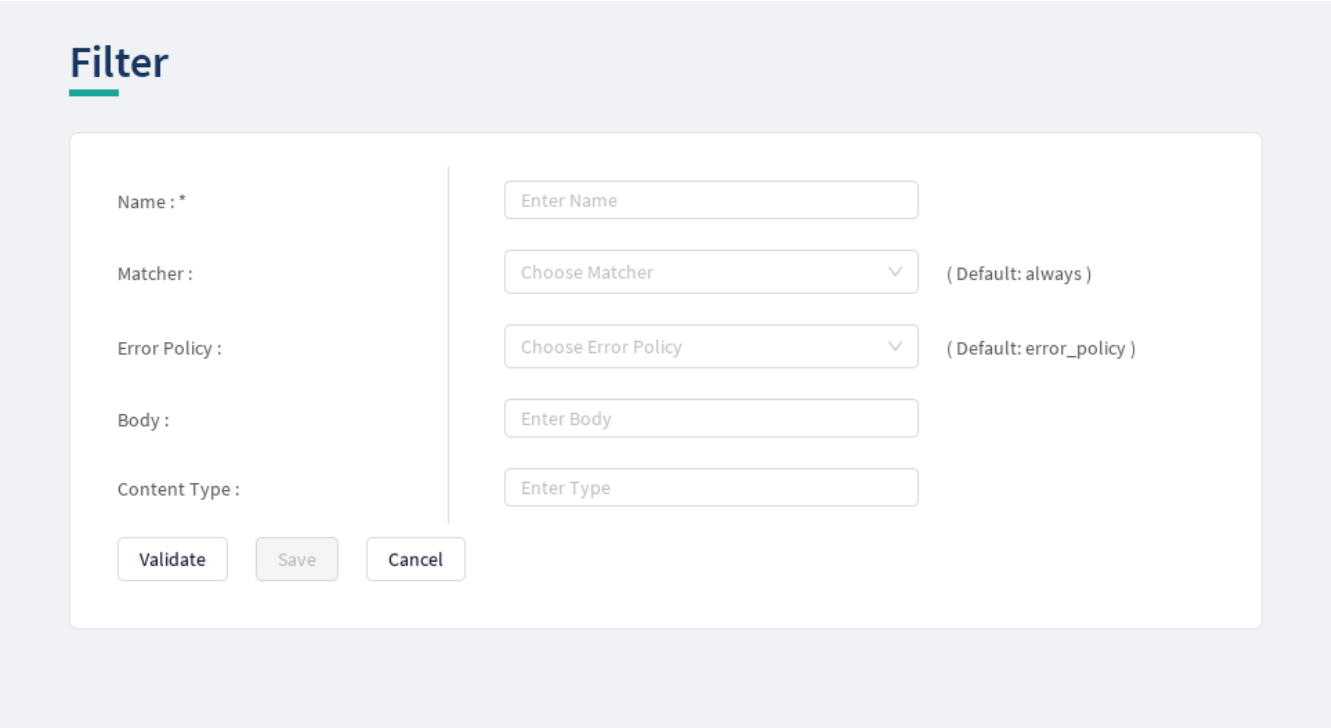


Figure 53. Configuring a filter plugin in the Web User Interface

Table 54. Filter Plugin's configuration options

Key	Values	Default value	Description
Name*	Free text. Alphanumeric, may contain underscores, may not start with a number.		The name identifying the Filter Plugin. This name of the plugin can be referenced from other parts of the configuration.
Matcher	Reference to a *Matcher Brick*.	Always: If the value is not defined the plugin is always executed.	It decides if the Plugin should be executed based on the call's details. For details see Matcher. If omitted the Plugin is always executed.
Error Policy	Reference to an *Error Policy Brick*.	error_policy	It defines a custom error policy to be applied if the Plugin reports an error. The settings of the Error policy here override the Security Flow's default error policy. For details see Error Policy.
Body	Can be defined in free text.		The body of the message sent in case an error policy is applied.
Content Type			The content type of HTTP error request sent, if the filter stops the call. It can be referenced by its name.

6.5.4. Fraud Detector

The Fraud Detector Plugin, leveraging the data collected from the calls by the selectors, evaluates the level of risk with regards to the call. The risk calculated by the Fraud Detector plugin is translated to a score between *0.0* and *100.0*. The lower the score is, the more secure and trustworthy the actual call is. Consequently, the value *0.0* means that the call is perfectly secure, until the value *100.0* identifies a malicious act with the call.

6.5.4.1. Configuring Fraud Detector Plugins

1. Click on the *PLUGINS* main navigation item in the Left navigation area. Alternatively you can also click on the sign to open up the sub-navigation items of *PLUGINS*.
2. Select *Fraud Detector* plugin.

In the configuration window that appears, you can either see the empty parameter values that can be configured for the actual component or you can see already configured component(s) and their parameters. The already configured components with defined parameters can be default components available in the system by default, or can be components configured by the administrator:

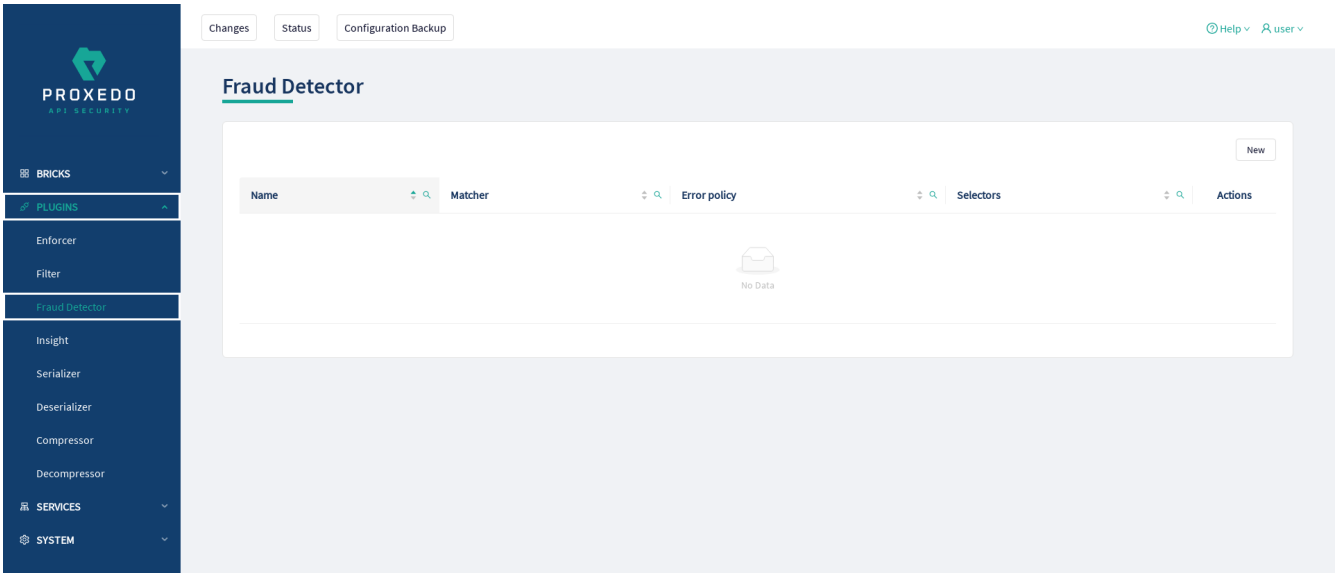


Figure 54. Fraud Detector's main page in the Web User Interface

Make sure that any component referenced in the configuration of this component, for example an Error policy or a Matcher selected from the drop-down lists, must remain part of the configuration later as well. Removing any of the referenced components might lead to invalid configuration.

3. Click on the *New* navigation button to create a Fraud Detector.
4. Add the name of the Fraud Detector.
5. Choose an error policy from the drop-down list. (Optional)
6. Choose a matcher from the drop-down list. (Optional)
7. Choose a *Selector* from the drop-down list. When it is selected click on the plus sign to add it to the configuration.
8. Click the *Validate* button to check if the defined parameters are of the correct type and all required parameters have been filled out for configuring the component. If the configuration is erroneous or incomplete, the Web UI provides a warning that the 'Component validation failed'. Also a warning with information on the missing or faulty elements appears at the problematic field. If the configuration of the component is valid, after clicking the *Validate* button, a 'Component validation successful' notification is shown.
9. Save the component configuration by clicking the *Save* button.

See Error Policy to understand how they shall be applied here.

The following values can be configured for the *Fraud Detector Plugin*:

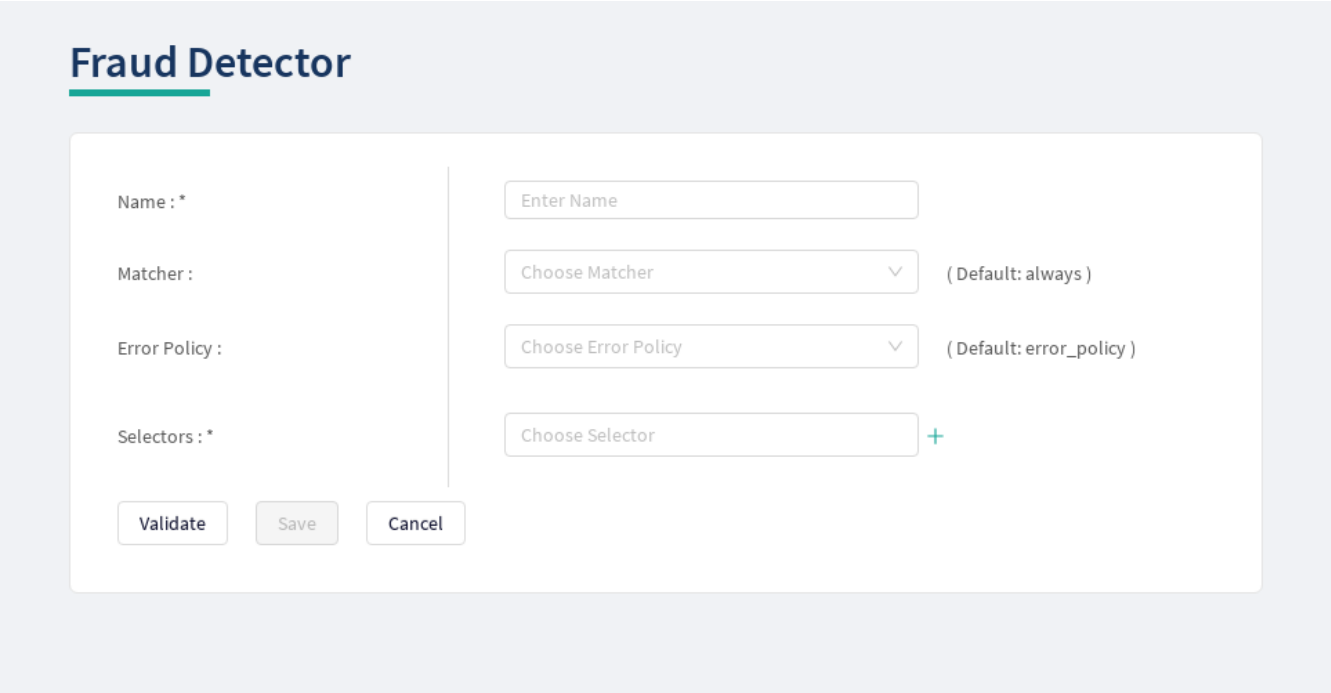


Figure 55. Configuring the Fraud Detector plugin in the Web User Interface

Table 55. Fraud Detector Plugin's configuration options

Key	Values	Default value	Description
Name*	Free text. Alphanumeric, may contain underscores, may not start with a number.		The name identifying the Fraud Detector. This name of the plugin can be referenced from other parts of the configuration.
Matcher	Reference to a *Matcher Brick*.	Always: If the value is not defined the plugin is always executed.	It decides if the Plugin should be executed based on the call's details. For details see Matcher. If omitted the Plugin is always executed.
Error Policy	Reference to an *Error Policy Brick*.	error_policy	It defines a custom error policy to be applied if the Plugin reports an error. The settings of the Error policy here override the Security Flow's default error policy. For details see Error Policy.
Selectors*	A list of references to *Selector Bricks*.		A list of Selector that collect information from the call. Selectors can be configured as listed in Selector configuration for the Fraud Detector Plugin. It is possible to add more data from the selectors to the Fraud Detector Plugin using custom fields, apart from the list in section Selector configuration for the Fraud Detector Plugin. In such cases contact the Balasys Support team.

6.5.5. Insight

It is a Plugin that extracts various data from the call and sends it to external systems (log servers, SIEMs, and other data analysis tools).

6.5.5.1. Configuring Insight Plugins

1. Click on the *PLUGINS* main navigation item in the Left navigation area. Alternatively you can also click on the sign to open up the sub-navigation items of *PLUGINS*.
2. Select *Insight* plugin.

In the configuration window that appears, you can either see the empty parameter values that can be configured for the actual component or you can see already configured component(s) and their parameters. The already configured components with defined parameters can be default components available in the system by default, or can be components configured by the administrator:

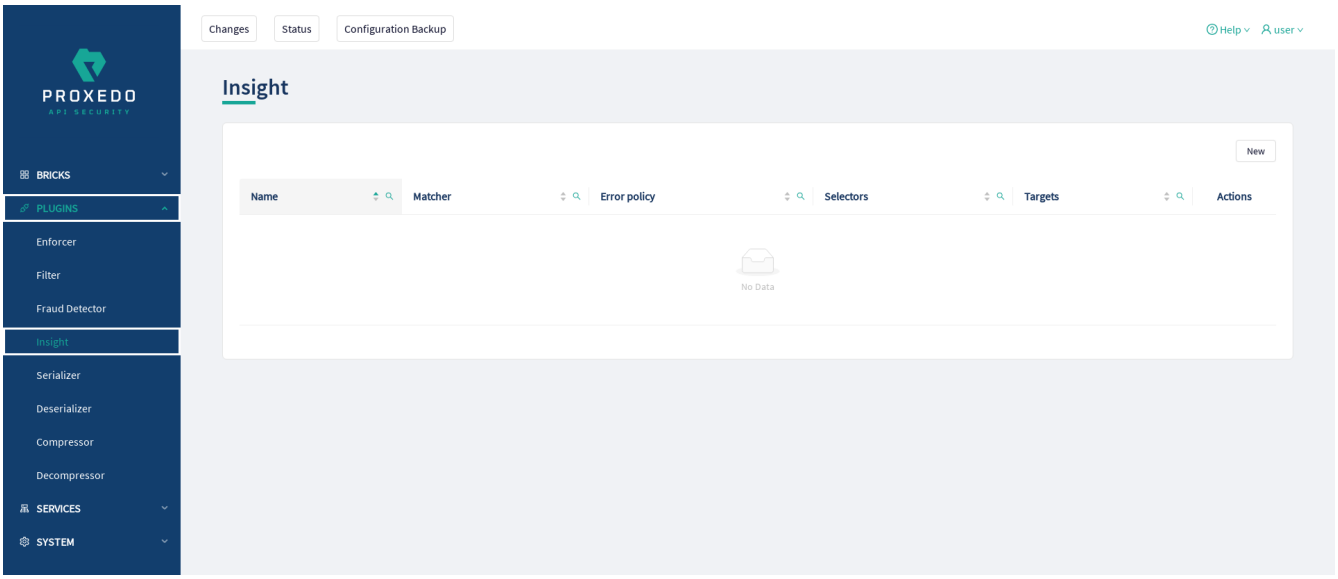


Figure 56. Insight Plugin's main page in the Web User Interface

3. Click on the *New* navigation button to create an Insight.

The Plugin uses the default *Error policy* by default, that is, the 'insight_default'.

The Plugin overrides the following fields of the default error policy:

Table 56. Default Insight Error Policy

Policy Setting	Default
request	log
response	log

Problems are considered errors that only need to be logged. If that is overridden then problems in the request are reported back to the client, while errors in the response are suppressed to avoid information leak.

See Error Policy to understand how defaults are applied.

The *Plugin* collects the information from all the selectors and sends them to all the targets.

The collected information from all the selectors is arranged into a dictionary: a list of *key — value* pairs. The key can be configured in each selector. Certain selectors might return complex data structures, that are made up of other dictionaries and/or lists. To ensure compatibility with a wide range of *Insight Target* types, such results are flattened. The path inside the complex data structure is encoded into the key for each value. More details are available on this in Data flattening.

4. Add the name of the Insight Plugin.
5. Choose an error policy from the drop-down list. (optional)
6. Choose a matcher from the drop-down list. (optional)
7. Add the message content for the error policy. (optional)
8. Choose a selector from the drop-down list.
9. Select the *Insight Target*.
10. Click the *Validate* button to check if the defined parameters are of the correct type and all required parameters have been filled out for configuring the component. If the configuration is erroneous or incomplete, the Web UI provides a warning that the 'Component validation failed'. Also a warning with information on the missing or faulty elements appears at the problematic field. If the configuration of the component is valid, after clicking the *Validate* button, a 'Component validation successful' notification is shown.
11. Save the component configuration by clicking the *Save* button.

The following values can be configured for the *Insight Plugin*:

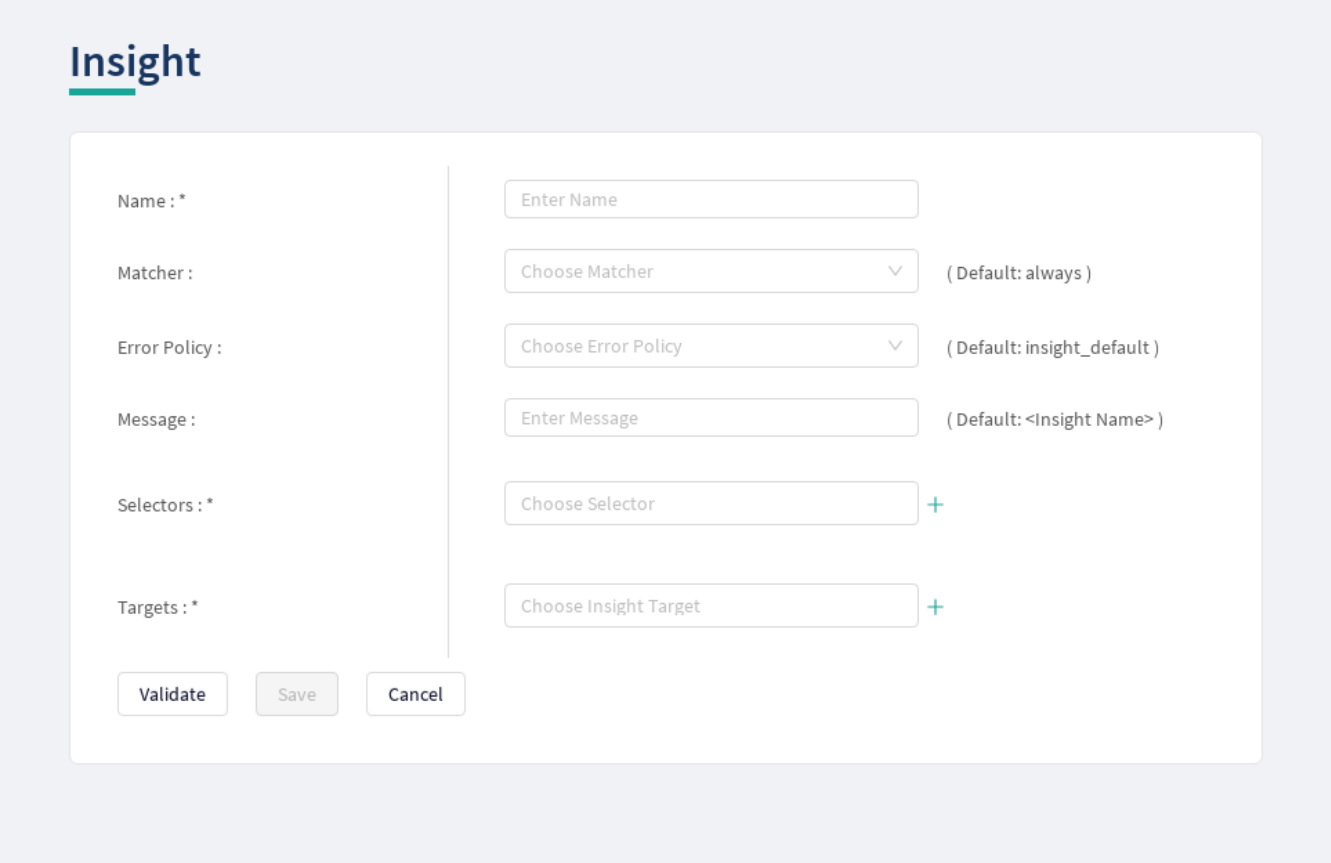


Figure 57. Configuring an insight plugin in the Web User Interface

Table 57. Insight Plugin's configuration options

Key	Values	Default value	Description
Name*	Free text. Alphanumeric, may contain underscores, may not start with a number.		The name identifying the insight. This name of the insight can be referenced from other parts of the configuration.

Key	Values	Default value	Description
Matcher	Reference to a *Matcher Brick*.	Always: If the value is not defined the plugin is always executed.	It decides if the Plugin should be executed based on the call's details. For details see Matcher. If omitted the Plugin is always executed.
Error Policy	Reference to an *Error Policy Brick*.	insight_default	It defines a custom error policy to be applied if the Plugin reports an error. The settings of the Error policy here override the Security Flow's default error policy. For details see Error Policy.
Message	Can be defined in free text.	The name of the plugin.	The message part of the log message.
Selectors*	A list of references to *Selector Bricks*.		A list of Selectors that collect information from the call. It is possible to multiselect more than one selector in this list by clicking on them. The multiple selected elements can then be added to the configuration by clicking on the plus sign.
Targets*	A list of references to *Insight Target Bricks*.		A list of Insight Targets where the collected information will be sent to.

6.5.6. Serializer

The *Serializer Plugin* is responsible for serializing the structured data to the format of the HTTP message's body.

Serialization needs to be done before compression. A typical Security Flow configuration starts with a *Decompressor* followed by a *Deserializer* and finishes with a *Serializer* followed by a *Compressor*. This ensures that transferred HTTP bodies are syntactically correct and that they are reconstructed to avoid transferring potentially crafted content.

The Serializer Plugin understands the Content-Type HTTP header and can work with JSON and XML content.

6.5.6.1. Configuring Serializer Plugins

1. Click on the *PLUGINS* main navigation item in the Left navigation area. Alternatively you can also click on the  sign to open up the sub-navigation items of *PLUGINS*.
2. Select *Serializer*.

The configuration window that appears presents the default Serializers, as listed in Default objects - PLUGINS and the configuration values already set by the user:

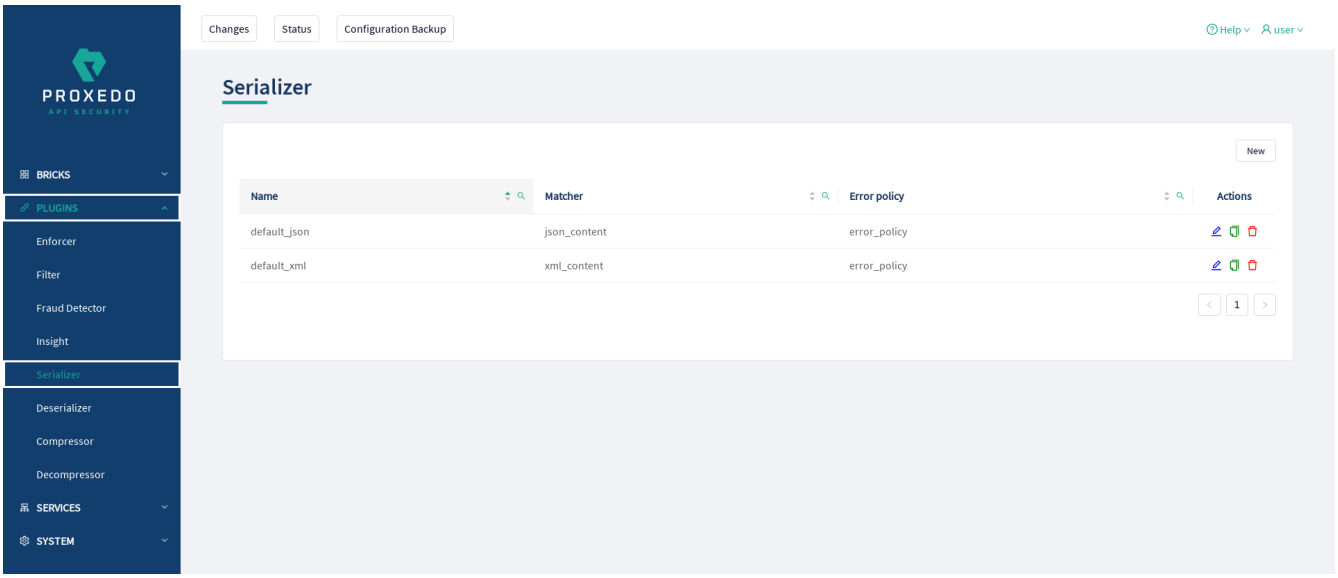


Figure 58. The serializer main page in the Web User Interface

3. Click on the *New* navigation button to create a Serializer.

The Plugin does not override any of the default error policy options.

Problems are considered errors that lead to the termination of the call. Problems in the request are reported back to the client, while errors in the response are suppressed to avoid information leak.

See Error Policy to understand how defaults are applied.

Continue configuring the serializer with the following steps:

4. Add the name of the serializer.
5. Select the type of the *Serializer*.
6. Choose an Error policy from the drop-down list.
7. Choose a Matcher from the drop-down list.
8. Click the *Validate* button to check if the defined parameters are of the correct type and all required parameters have been filled out for configuring the component. If the configuration is erroneous or incomplete, the Web UI provides a warning that the 'Component validation failed'. Also a warning with information on the missing or faulty elements appears at the problematic field. If the configuration of the component is valid, after clicking the *Validate* button, a 'Component validation successful' notification is shown.
9. Save the component configuration by clicking the *Save* button.

The following values can be configured for the *Serializer Plugin*:

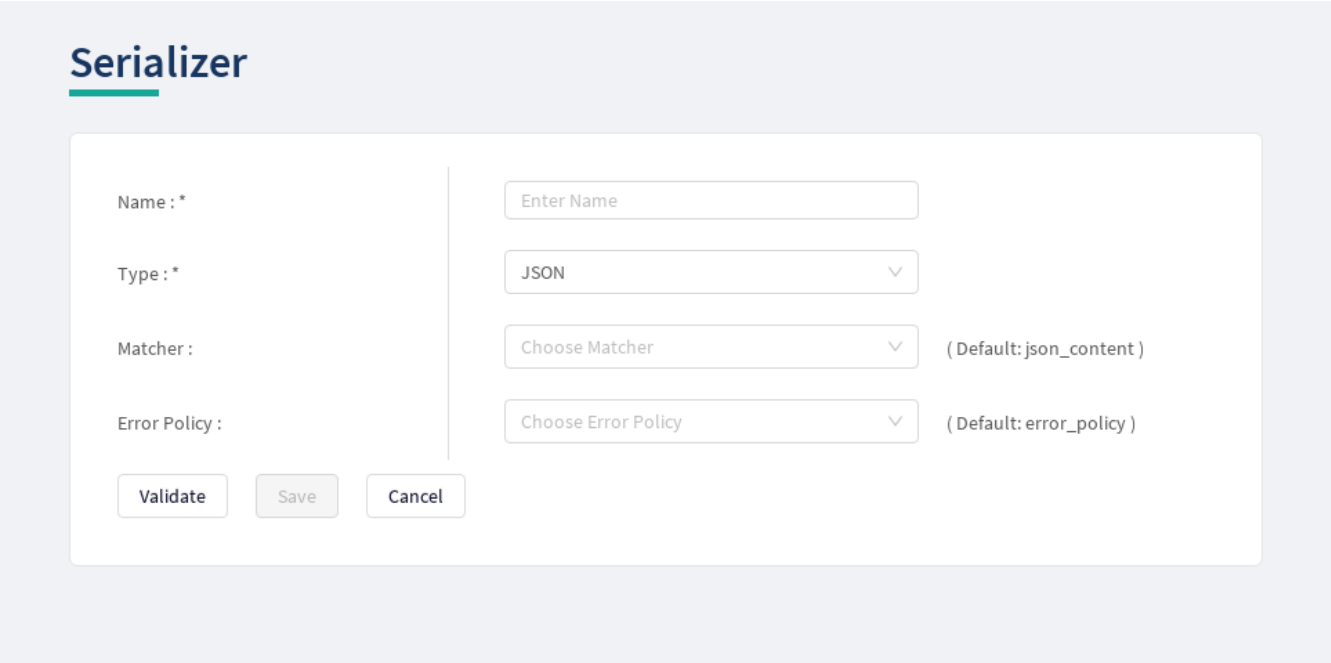


Figure 59. Configuring a serializer in the Web User Interface

Table 58. Serializers' configuration options

Key	Values	Default value	Description
Name*	Free text. Alphanumeric, may contain underscores, may not start with a number.		The name identifying the serializer. This name of the serializer can be referenced from other parts of the configuration, that is, the Plugin is reusable.
Type*	The value can be selected from a drop-down list. The value can be: • JSON • XML		There are two types of predefined (de)serializer plugins.
Matcher	Reference to a *Matcher Brick*.	Depending on which 'Type' was selected for the *Serializer*, the default value can be: json_content or xml_content.	It decides if the Plugin should be executed based on the call's details. For details see Matcher. If no matcher is configured the Plugin is always executed.
Error Policy	Reference to an *Error Policy Brick*.	error_policy	It defines a custom error policy to be applied if the Plugin reports an error. The settings of the Error policy here override the Security Flow's default error policy. If no error policy is configured, the plugin type's default error policy is applied. For details see Error Policy.

6.5.7. Deserializer

It is a Plugin responsible for parsing the HTTP message's body to structured data. This ensures that a message is well-formed. The structured data will also be consumed by other Plugins that operate on the body of the message.

A typical Security Flow configuration starts with a *Decompressor* followed by a *Deserializer* and finishes with a *Serializer* followed by a *Compressor*. This ensures that transferred HTTP bodies are syntactically correct and that they are reconstructed to avoid transferring potentially crafted content.

6.5.7.1. Configuring Deserializer Plugins

1. Click on the *PLUGINS* main navigation item in the Left navigation area. Alternatively you can also click on the sign to open up the sub-navigation items of *PLUGINS*.
2. Select *Deserializer* plugin.

The configuration window that appears presents the default Deserializers, as listed in Default objects - PLUGINS and the configuration values already set by the user:

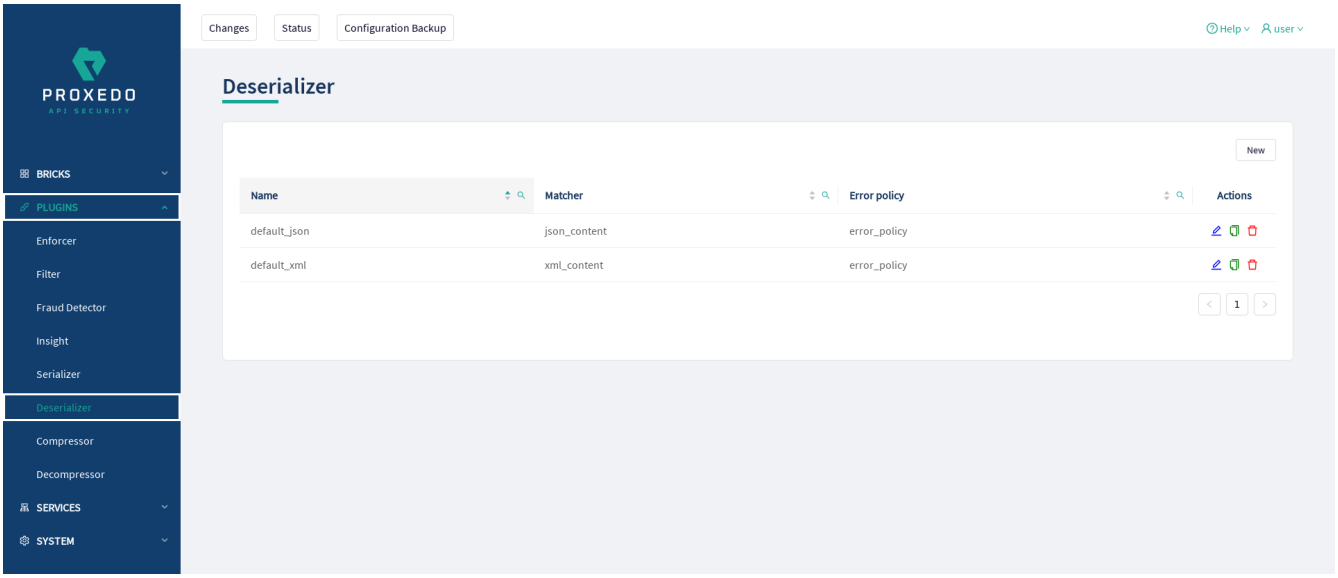


Figure 60. The deserializer's main page in the Web User Interface

2. Click on the *New* navigation button to create a Deserializer.

The Plugin does not override any of the default error policy options.

Problems are considered errors that lead to the termination of the call. Problems in the request are reported back to the client, while errors in the response are suppressed to avoid information leak.

See Error Policy to understand how defaults are applied.

3. Add the name of the deserializer.
4. Select the Type of the Deserializer.
5. Choose an Error policy from the drop-down list.
6. Choose a Matcher from the drop-down list.
7. Click the *Validate* button to check if the defined parameters are of the correct type and all required parameters have been filled out for configuring the component. If the configuration is erroneous or incomplete, the Web UI provides a warning that the 'Component validation failed'. Also a warning with information on the missing or faulty elements appears at the problematic field. If the configuration of the component is valid, after clicking the *Validate* button, a 'Component validation successful' notification is shown.
8. Save the component configuration by clicking the *Save* button.

The following values can be configured for the *Deserializer Plugin*:

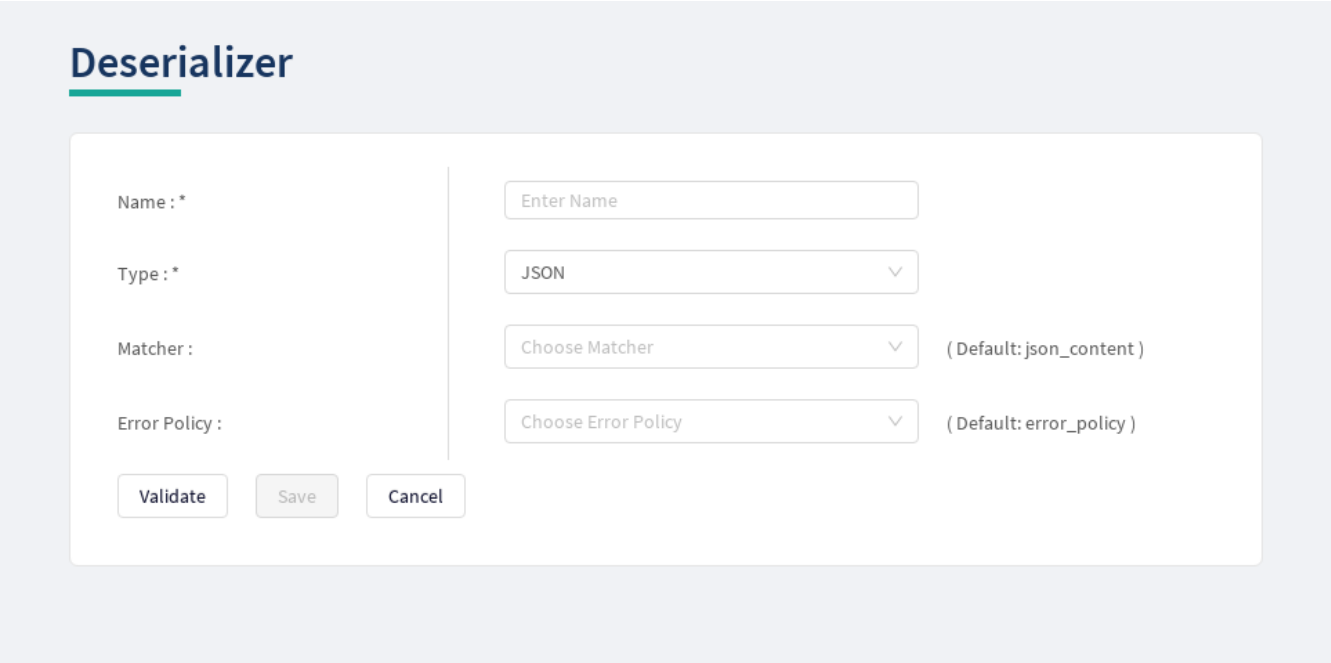


Figure 61. Configuring a deserializer in the Web User Interface

Table 59. Deserializers' configuration options

Key	Values	Default value	Description
Name*	Free text. Alphanumeric, may contain underscores, may not start with a number.		The name identifying the deserializer. This name of the deserializer can be referenced from other parts of the configuration.
Type*	The value can be selected from a drop-down list. The value can be: • JSON • XML		There are two types of predefined (de)serializer plugins.
Matcher	Reference to a *Matcher Brick*.	Depending on which 'Type' was selected for the *Deserializer*, the default value can be: json_content or xml_content.	It decides if the Plugin should be executed based on the call's details. For details see Matcher. If omitted the Plugin is always executed.
Error Policy	Reference to an *Error Policy Brick*.	error_policy	It defines a custom error policy to be applied if the Plugin reports an error. The settings of the Error policy here override the Security Flow's default error policy. For details see Error Policy.

Key	Values	Default value	Description
Charset Conflict	• drop: If this parameter is set to 'drop', the configuration instructs to drop the call in case there is conflict for the character set in the message's header. • log: If the value is set to 'log', the system will use either type of the character set defined and will log the error.	drop	This parameter needs to be configured in case the 'Type' of the Deserializer is set to XML. In XML messages, there might be a conflict in the definition of the character set. The XML and the HTTP headers might instruct to use different character sets. The conflicting information on the character set can be configured to be handled in two different ways, that is the call dropped, or the call maintained and the error logged, depending on the settings of this parameter.

6.5.8. Compressor

The *Compressor Plugin* compresses the body of the HTTP message.

Compressors understand the *Transfer-Encoding* HTTP header and compress data by using the *gzip*, *deflate* and *brotli* algorithms.

6.5.8.1. Configuring Compressor Plugins

1. Click on the *PLUGINS* main navigation item in the Left navigation area. Alternatively you can also click on the  sign to open up the sub-navigation items of *PLUGINS*.
2. Select *Compressor*.

The configuration window that appears presents the default Compressor, as listed in Default objects - PLUGINS and the configuration values already set by the user:

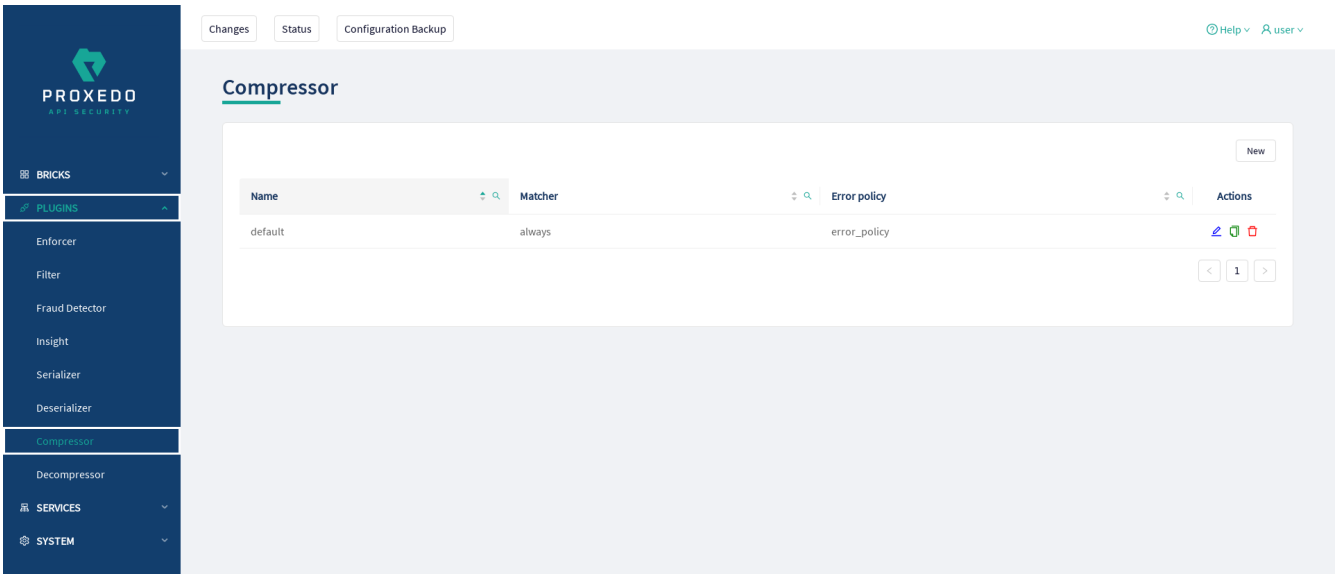


Figure 62. The compressor main page in the Web User Interface

3. Click on the *New* navigation button to create a Compressor.
4. Add the name of the compressor.
5. Choose an Error policy from the drop-down list.

6. Choose a Matcher from the drop-down list.
7. Click the *Validate* button to check if the defined parameters are of the correct type and all required parameters have been filled out for configuring the component. If the configuration is erroneous or incomplete, the Web UI provides a warning that the 'Component validation failed'. Also a warning with information on the missing or faulty elements appears at the problematic field. If the configuration of the component is valid, after clicking the *Validate* button, a 'Component validation successful' notification is shown.
8. Save the component configuration by clicking the *Save* button.

The following values can be configured for the *Compressor Plugin*:

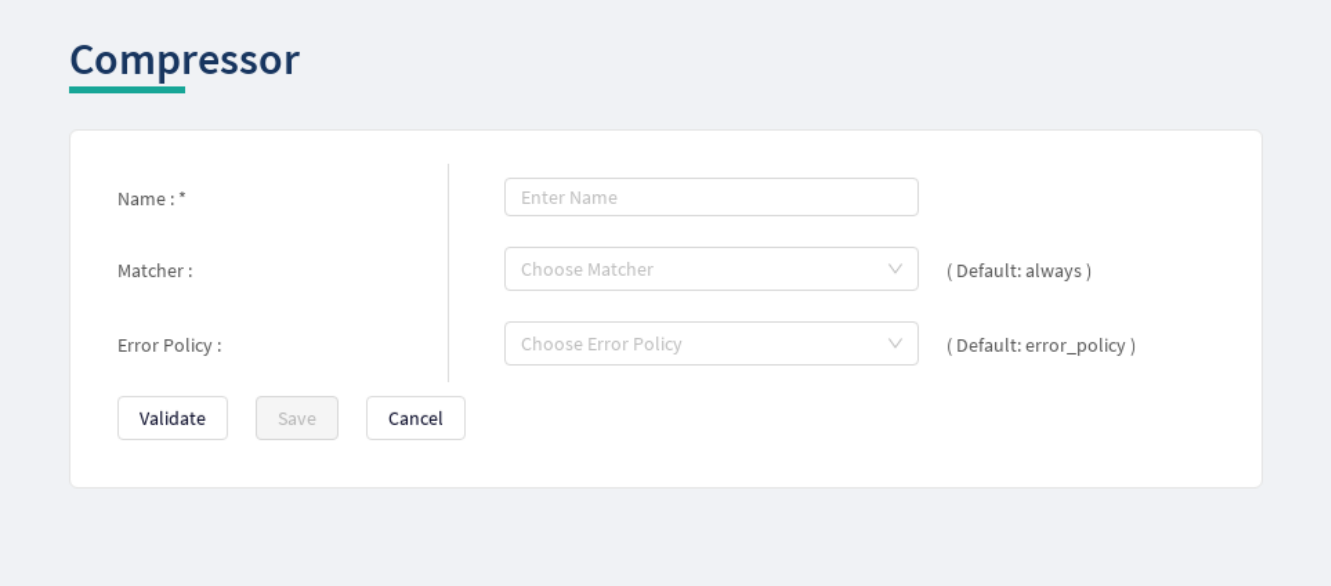


Figure 63. Configuring a compressor in the Web User Interface

Table 60. The Compressors' configuration options

Key	Values	Default value	Description
Name*	Free text. Alphanumeric, may contain underscores, may not start with a number.		The name identifying the compressor. This name of the compressor can be referenced from other parts of the configuration, that is, the Plugin is reusable.
Matcher	Reference to a *Matcher Brick*.	Always: If the value is not defined the plugin is always executed.	It decides if the Plugin should be executed based on the call's details. For details see Matcher. If no matcher is configured the Plugin is always executed.
Error Policy	Reference to an *Error Policy Brick*.	The Plugin has a default error policy.	It defines a custom error policy to be applied if the Plugin reports an error. The settings of the Error policy here override the Security Flow's default error policy. If no error policy is configured, the plugin type's default error policy is applied. For details see Error Policy.

6.5.9. Decompressor

The *Decompressor Plugin* decompresses the body of the HTTP message.

Decompressors understand the *Transfer-Encoding* HTTP header and can work with content optionally compressed by the *gzip*, *deflate* and *brotli* algorithms.

6.5.9.1. Configuring Decompressor Plugins

1. Click on the *PLUGINS* main navigation item in the Left navigation area. Alternatively you can also click on the sign to open up the sub-navigation items of *PLUGINS*.
2. Select *Decompressor*.

The configuration window that appears presents the default Decompressor, as listed in Default objects - PLUGINS and the configuration values already set by the user:

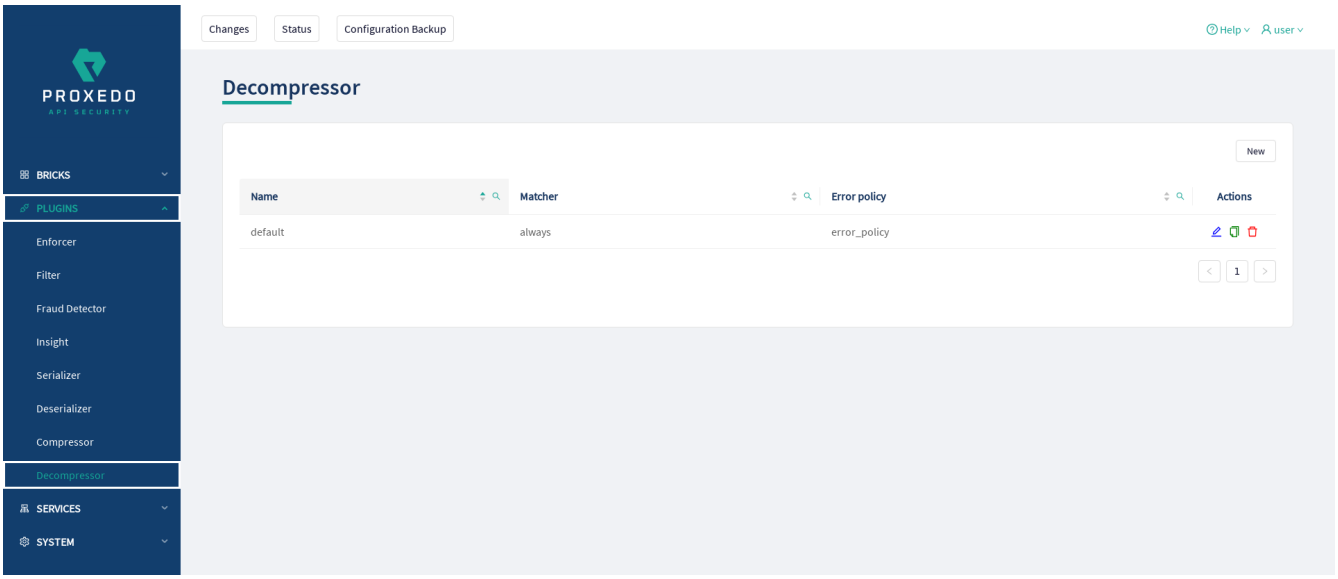


Figure 64. The Decompressor's main page in the Web User Interface

3. Click on the *New* navigation button to create a Deserializer.
4. Add the name of the decompressor.
5. Choose an Error policy from the drop-down list.
6. Choose a Matcher from the drop-down list.
7. Click the *Validate* button to check if the defined parameters are of the correct type and all required parameters have been filled out for configuring the component. If the configuration is erroneous or incomplete, the Web UI provides a warning that the 'Component validation failed'. Also a warning with information on the missing or faulty elements appears at the problematic field. If the configuration of the component is valid, after clicking the *Validate* button, a 'Component validation successful' notification is shown.
8. Save the component configuration by clicking the *Save* button.

The following values can be configured for the *Decompressor Plugin*:

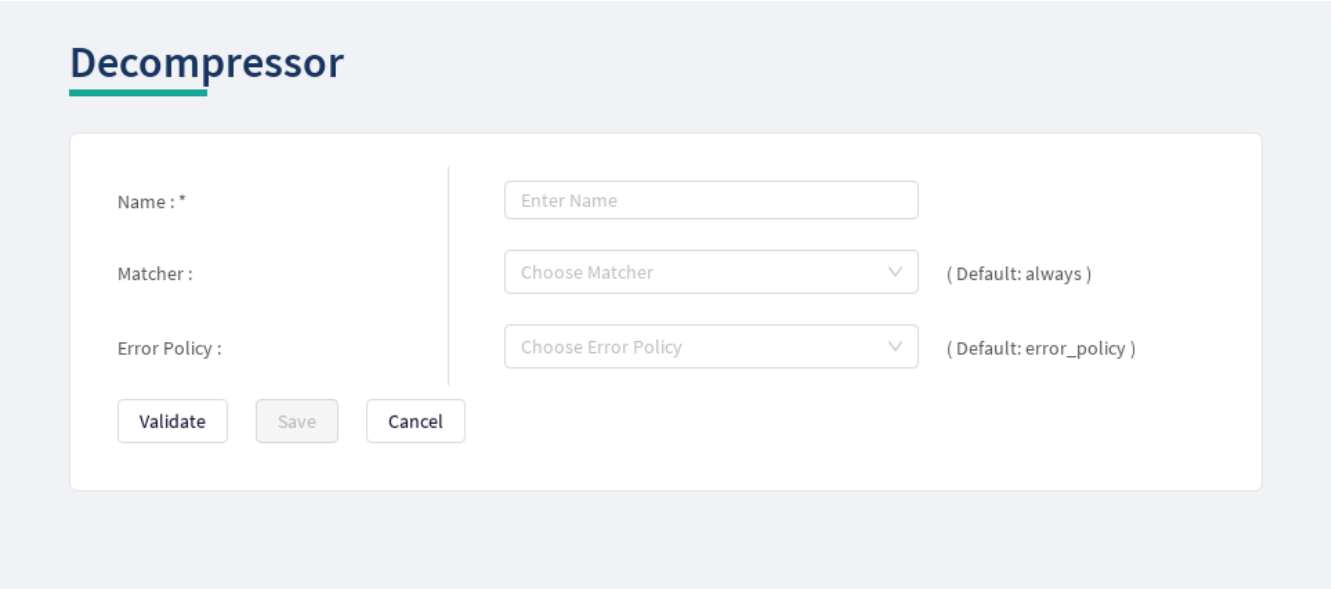


Figure 65. Configuring a decompressor in the Web User Interface

Table 61. The Decompressors' configuration options

Key	Values	Default value	Description
Name*	Free text. Alphanumeric, may contain underscores, may not start with a number.		The name identifying the decompressor. This name of the decompressor can be referenced from other parts of the configuration, that is, the Plugin is reusable.
Matcher	Reference to a *Matcher Brick*.	Always: If the value is not defined the plugin is always executed.	It decides if the Plugin should be executed based on the call's details. For details see Matcher. If no matcher is configured the Plugin is always executed.
Error Policy	Reference to an *Error Policy Brick*.	The Plugin has a default error policy.	It defines a custom error policy to be applied if the Plugin reports an error. The settings of the Error policy here override the Security Flow's default error policy. If no error policy is configured, the plugin type's default error policy is applied. For details see Error Policy.

6.6. SERVICES - Configuration units

Proxedo API Security is based on a micro-services architecture.

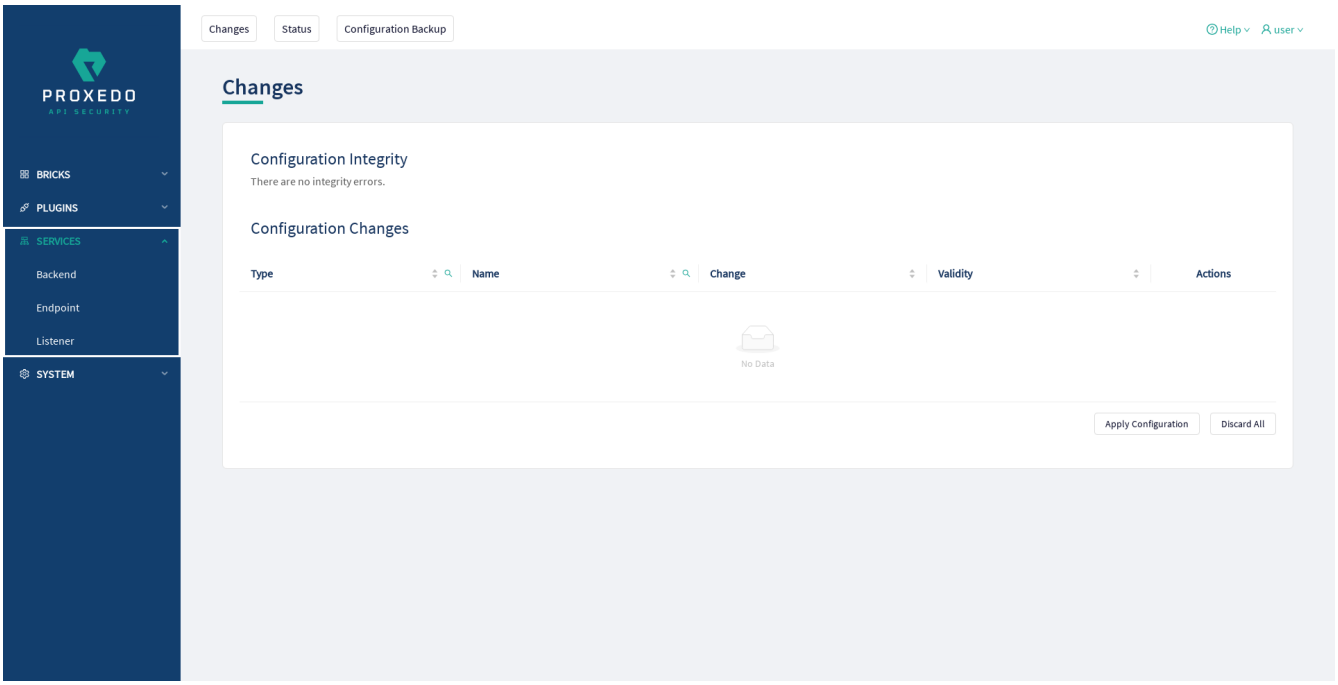


Figure 66. The SERVICES main page in the Web User Interface

6.6.1. Backend

Backends are a set of servers for a given API endpoint.

Their configuration is made up of two main parts:

- a list of servers: port pairs and how to route traffic to them
- TLS configuration for talking to the servers

6.6.1.1. Configuring Backends

1. Click on the *SERVICES* main navigation item in the Left navigation area. Alternatively you can also click on the sign to open up the sub-navigation items of *SERVICES*.
2. Select *Backend*.

In the configuration window that appears, you can either see the empty parameter values that can be configured for the actual component or you can see already configured component(s) and their parameters. The already configured components with defined parameters can be default components available in the system by default, or can be components configured by the administrator:

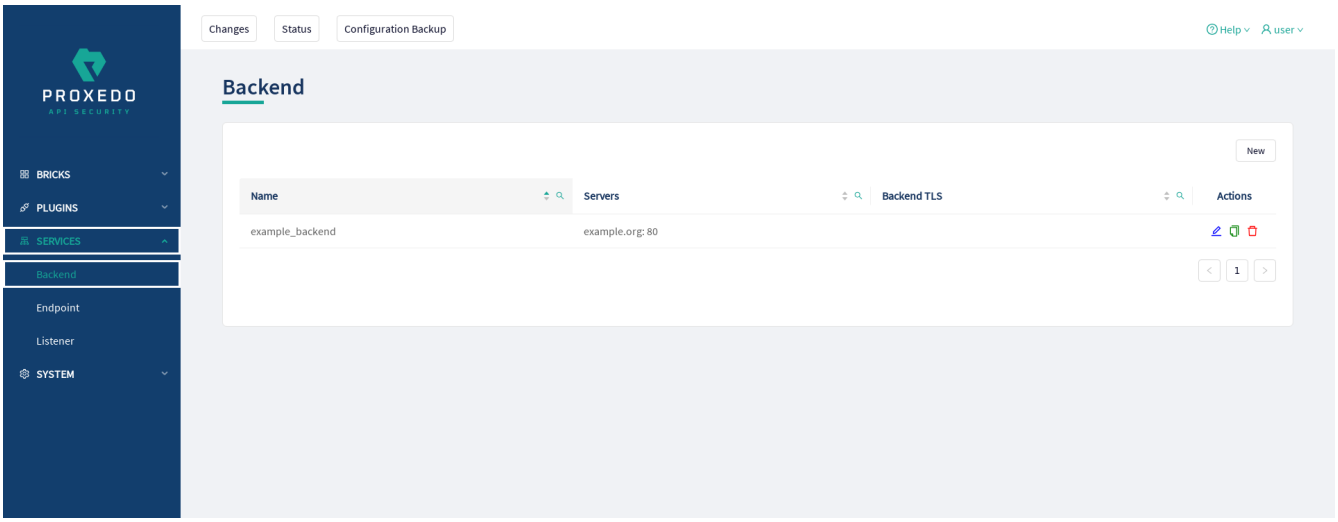


Figure 67. The main page for Backend

3. Click on the *New* navigation button to create a Backend.
4. Name the *Backend* configuration.
5. Provide the values for the Servers parameter: *Host* and *Port*.
6. Click the *Validate* button to check if the defined parameters are of the correct type and all required parameters have been filled out for configuring the component. If the configuration is erroneous or incomplete, the Web UI provides a warning that the 'Component validation failed'. Also a warning with information on the missing or faulty elements appears at the problematic field. If the configuration of the component is valid, after clicking the *Validate* button, a 'Component validation successful' notification is shown.
7. Save the component configuration by clicking the *Save* button.

The following values can be configured for the *Backend Service*:

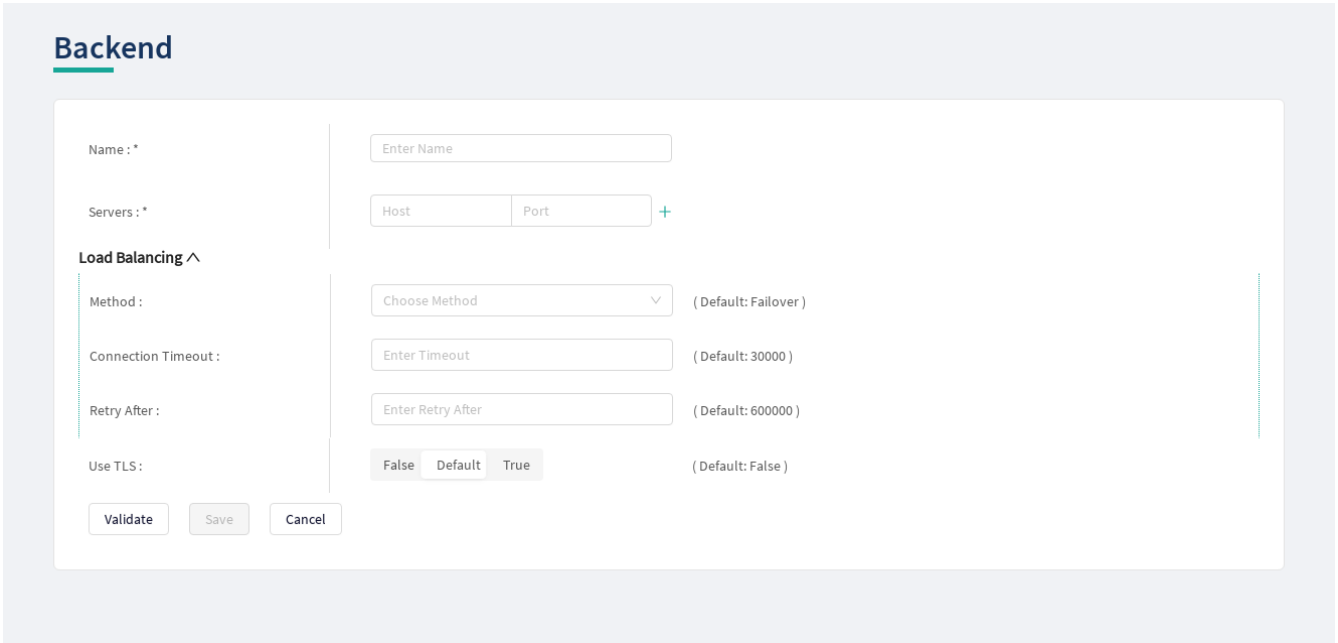


Figure 68. Configuring backend in the Web User Interface

Table 62. Backend configuration

Key	Values	Default value	Description
Name*	Free text. Alphanumeric, may contain underscores, may not start with a number.		The name identifying the backend. This name of the backend can be referenced from other parts of the configuration.
Servers*	There are two values to be configured: • **Host**: The name or IP address of the host to connect to. • **Port**: The port on **host** to connect to. (You can add the values by clicking the '+' sign.)		The list of servers that serve API endpoint(s). See Backend servers' configuration for details.
Load Balancing / Method	One of the following methods can be used: • Failover: use the first server while available, then fail over to the next • Round Robin: use all servers in a round-robin fashion If the value is not configured the default value will be added.	Failover	Load balancing method to use.
Load Balancing / Connection Timeout	If the value is not configured the default value will be added.	30000	The connection timeout in milliseconds towards the backend server.
Load Balancing / Retry After	If the value is not configured the default value will be added.	600000	The time in milliseconds before connection towards the backend server is started again in case of a connection failure.
Use TLS	True or False.	False	Enables using TLS in the connection towards the backend servers.
Backend TLS*	Reference to a *TLS Brick* of type *Backend TLS*.		The TLS configuration towards the backend servers. See Configuring Backend TLS Bricks for details. Mandatory if *Use TLS* is set to *True*.

6.6.2. Endpoint

An endpoint holds together all the policies that apply to a certain API endpoint:

- List of URLs
- The default error policy for the endpoint
- The backend to which requests will be forwarded
- The authentication method
- The security flow that will be applied to the traffic

6.6.2.1. Security Flow

The Security Flow definition in an endpoint lists what happens to the traffic on a given endpoint.

To understand how requests flow through PAS, see Understanding processing flow. The Security Flow starts when the Transport Director has already set up client connection and routed the request to the Flow Director. At this point the TLS and HTTP layers are already processed, but the content in the body of the request is available only in raw format and has not been parsed yet.

At this stage, the configuration security flow decides on what happens to the traffic by applying a list of *Plugins* one by one. *Plugin* is a collective name for Enforcers, Insights, Filters, etc. Once, all the *plugins* have processed the request, the control is handed back to the *Transport Director* which routes the request to a backend server, and comes back with the response after handling TLS and HTTP. At this point, the *Flow Director* applies another list of *Plugins* to response, and once done, it hands back the response to the *Transport Director* which in turn returns that to the client.

If at any point an error occurs, the error policy is applied — which might either mean to lead to logging the error or to terminating processing and returning an error indication to the client.

Plugins can override the endpoint's error policy.

Also note that different *Plugins* need different data. An Insight that applies a JMESPath query needs parsed JSON, while one that extracts value from an HTTP header field does not. Other *Plugins* provide these required values, like a JSON deserializer *Plugin*. It is important that the *Plugins* are configured in such an order that the required data is made available beforehand.

6.6.2.2. Configuring Endpoints

1. Click on the *SERVICES* main navigation item in the Left navigation area. Alternatively you can also click on the sign to open up the sub-navigation items of *SERVICES*.
2. Select *Endpoint*.

In the configuration window that appears, you can either see the empty parameter values that can be configured for the actual component or you can see already configured component(s) and their parameters. The already configured components with defined parameters can be default components available in the system by default, or can be components configured by the administrator:

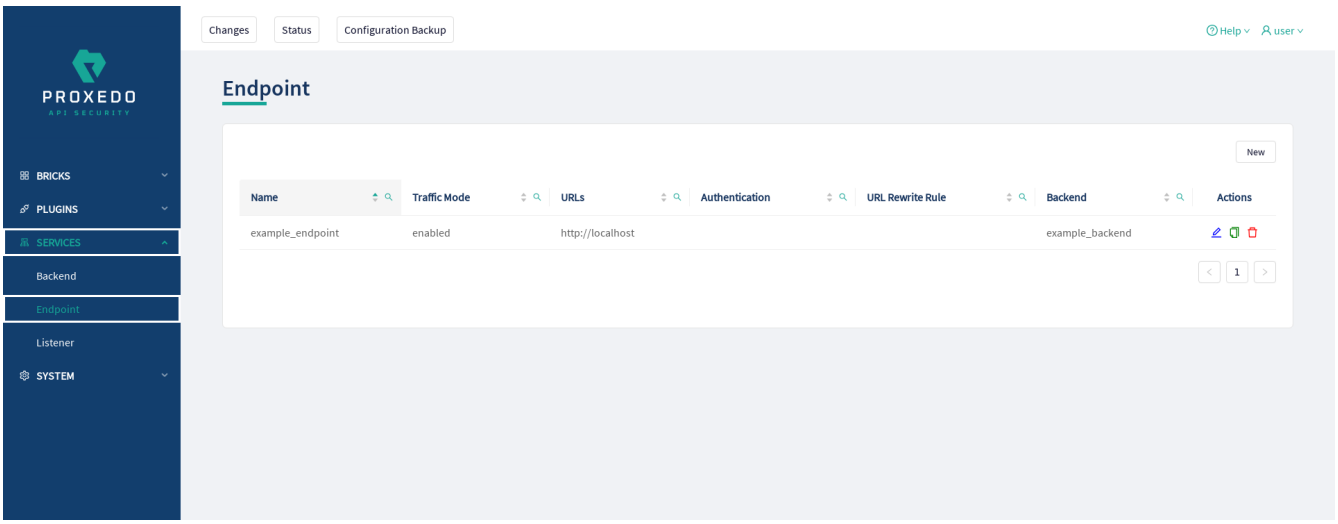


Figure 69. The main page for Endpoint

3. Click on the *New* navigation button to create an Endpoint.
4. Name the *Endpoint* Service.
5. Select the *Backend* parameter from the drop-down list. Backend servers are configured under the *SERVICES* main navigation item.
6. Complete a Security Flow from the configured (and the default) plugins. For more details, see Security Flow.
    - Choose the *Request* plugin from the drop-down list. The Plugin options available from the drop-down list

have been configured under the *PLUGINS* main navigation item.

- Choose the *Response* plugin from the drop-down list. The Plugin options available from the drop-down list have been configured under the *PLUGINS* main navigation item.

7. Provide the URL to address the API endpoint.
8. Click the *Validate* button to check if the defined parameters are of the correct type and all required parameters have been filled out for configuring the component. If the configuration is erroneous or incomplete, the Web UI provides a warning that the 'Component validation failed'. Also a warning with information on the missing or faulty elements appears at the problematic field. If the configuration of the component is valid, after clicking the *Validate* button, a 'Component validation successful' notification is shown.
9. Save the component configuration by clicking the *Save* button.

While ports must be unique, as only one listener can bind to a specific port, it is perfectly valid to route incoming traffic from multiple listeners to the same endpoint.

A typical security flow is configured with the *plugins* in the following order:

- a *Decompressor Plugin* that decompresses the content of the request
- a *Deserializer Plugin* that parses the content of the request
- an *Enforcer Plugins* that ensure the call is valid
- *Insight Plugins* that extract important data from certain calls
- a *Serializer Plugin* that rebuilds the contents of the request
- a *Compressor Plugin* that compresses the content of the request

The *Plugin* configurations are reusable.

The following values can be configured for the *Endpoint Service*:

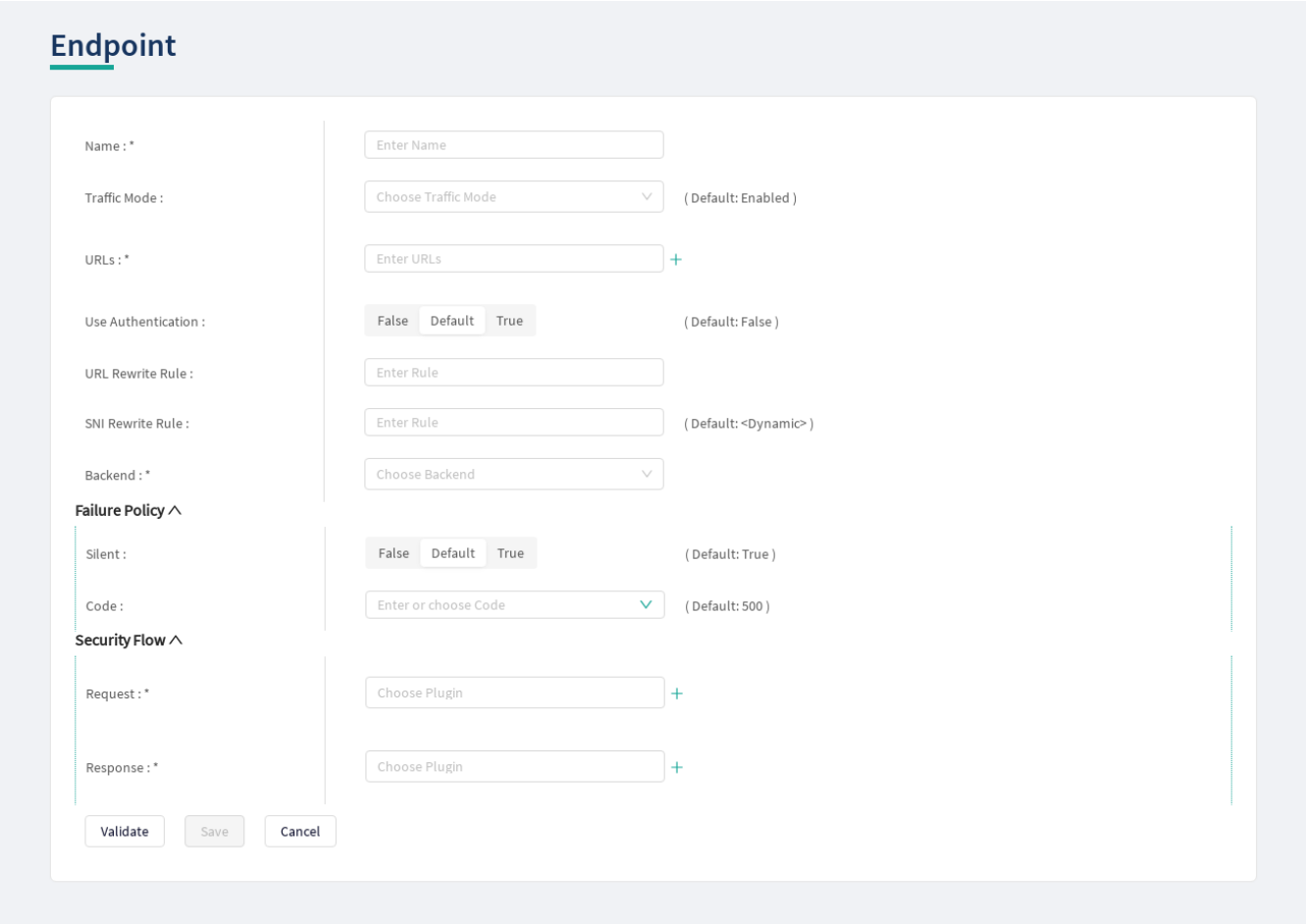


Figure 70. Configuring endpoint in the Web User Interface

Table 63. Endpoint configuration

Key	Values	Default value	Description
Name*	Free text. Alphanumeric, may contain underscores, may not start with a number.		The name identifying the endpoint. This name of the endpoint can be referenced from other parts of the configuration.
Traffic Mode	There are three options: • Enabled: the plugins will process the calls. This is the normal operation. • Disabled: calls will be automatically rejected with a silent HTTP 400 error code. This mode completely prevents traffic from reaching the backends. • Pass-through: allows calls to pass through the endpoint without interception or modification.	Enabled	This option allows the user to disable security flows on the endpoint for troubleshooting purposes, without altering the existing configuration. The pass-through option enables traversing traffic, while prevents any modification or interference.
URLs*			The URLs which the clients use to address the API endpoint.

Key	Values	Default value	Description
URL Rewrite Rule			The URL by which the backend servers understand incoming requests. When set, two transformations take place: • The original URL will be replaced by the matching URL configured for the *Endpoint*. • The *Host* header will be replaced by the host indicated in the URL rewrite rule.
SNI Rewrite Rule		<Dynamic>	It can be used to rewrite the Server Name Indication (SNI) field in a TLS handshake towards the backends. The *<Dynamic>* default value means that the SNI value used towards the backend will be the same as the value of the Host header, either coming from the client or defined in the URL Rewrite Rule.
Backend*	Reference to a *Backend Service*.		Backends are a set of servers for a given API endpoint. For more details, see Backend.
Use Authentication	True or False.	False	Enables using authentication for incoming calls. NOTE: for authenticated *Endpoints*, it is recommended to use a *Listener* with TLS.
Authentication*	Reference to an *Authentication Brick*.		The configuration of the selected authentication method. Mandatory if *Use Authentication* is set to *True*.
Failure Policy	Two values have to be configured: • Silent • Code	Silent: True; Code: 500	With the help of the **Failure Policy**, it can be configured whether the client shall receive notification or not, and whether the notification shall contain the code on the type of the failure. The values in details are as follows: • Silent: If the **silent** value is active, the Failure policy is not reported. If the **silent** value is inactive, the failure policy is reported towards the user. • Code: Code is an HTTP response code here, that can be set manually or from the provided drop-down list.

Key	Values	Default value	Description
Security Flow*	The security flow process requires the configuration of the following values, each containing a list of *Plugins*. • **Request*** • **Response***		The values in details are as follows: • Request: The Transport Director sets up client connection and routes the request to the Flow Director. The **Request** has numerous values to be configured. For more details, see Security Flow. • Response: The Transport Director routes the request to a backend server, and comes back with the response after handling TLS and HTTP. For more details, see Security Flow. Note, that both for the Request and Response parameters, it is possible to multiselect more than one element in the list by clicking on them. The multiple selected elements can then be added to the configuration by clicking on the plus sign.

6.6.3. Listeners

Listeners are network endpoints where services are exposed to the network. They consist of:

- a listening port
- an optional client-side TLS configuration if HTTPS is used
- a list of endpoints that handle the traffic.

Since these are the entry points for client traffic it must be routed here on the network.

6.6.3.1. Configuring Listeners

1. Click on the *SERVICES* main navigation item in the Left navigation area. Alternatively you can also click on the  sign to open up the sub-navigation items of *SERVICES*.
2. Select *Listener*.

In the configuration window that appears, you can either see the empty parameter values that can be configured for the actual component or you can see already configured component(s) and their parameters. The already configured components with defined parameters can be default components available in the system by default, or can be components configured by the administrator:

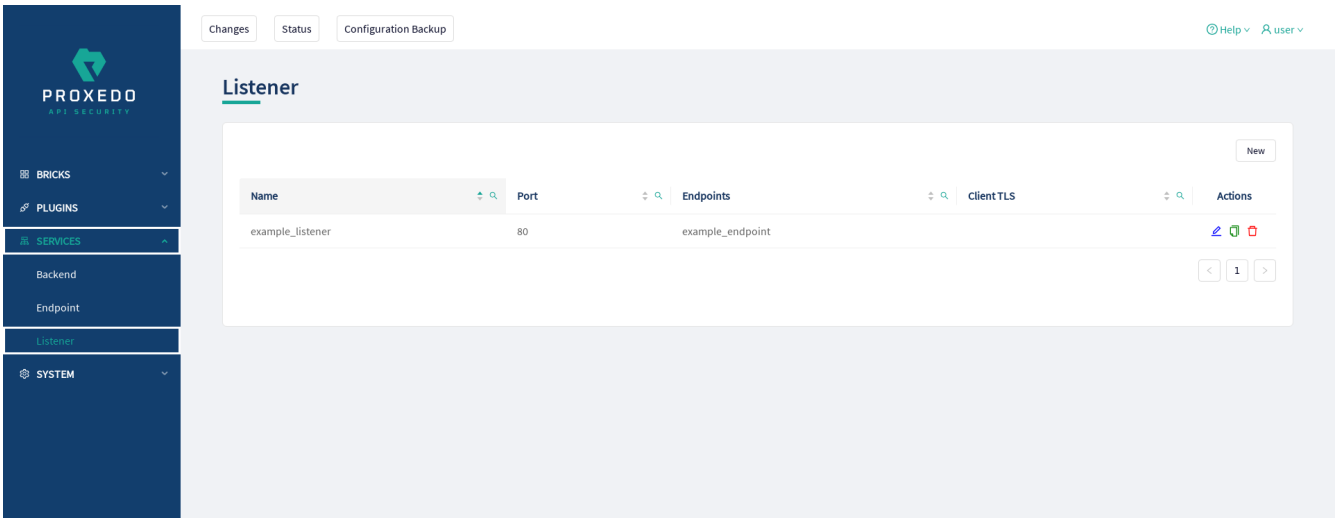


Figure 71. Listener's main page in the Web User Interface

3. Click on the *New* navigation button to create a Listener.

At least one listener must always be configured in the Proxedo API Security configuration.

4. Name the *Listener Service*.
5. Select the *Client TLS* parameter from the drop-down list. The client side TLS parameter values have to be defined previously under *BRICKS*.
6. Select the *Endpoint* from the drop-down list. The endpoint values have to be defined previously under SERVICES/Endpoint.

All endpoints in the list must have the same backend and backend URL configured.

7. Fill in the *Port* information. If it is not configured, the default value will be applied.

Ports must be unique, only one listener can bind to a specific port. It is however perfectly valid to route incoming traffic from multiple listeners to the same endpoint.

8. Click the *Validate* button to check if the defined parameters are of the correct type and all required parameters have been filled out for configuring the component. If the configuration is erroneous or incomplete, the Web UI provides a warning that the 'Component validation failed'. Also a warning with information on the missing or faulty elements appears at the problematic field. If the configuration of the component is valid, after clicking the *Validate* button, a 'Component validation successful' notification is shown.
9. Save the component configuration by clicking the *Save* button.

The following values can be configured for the *Listener Service*:

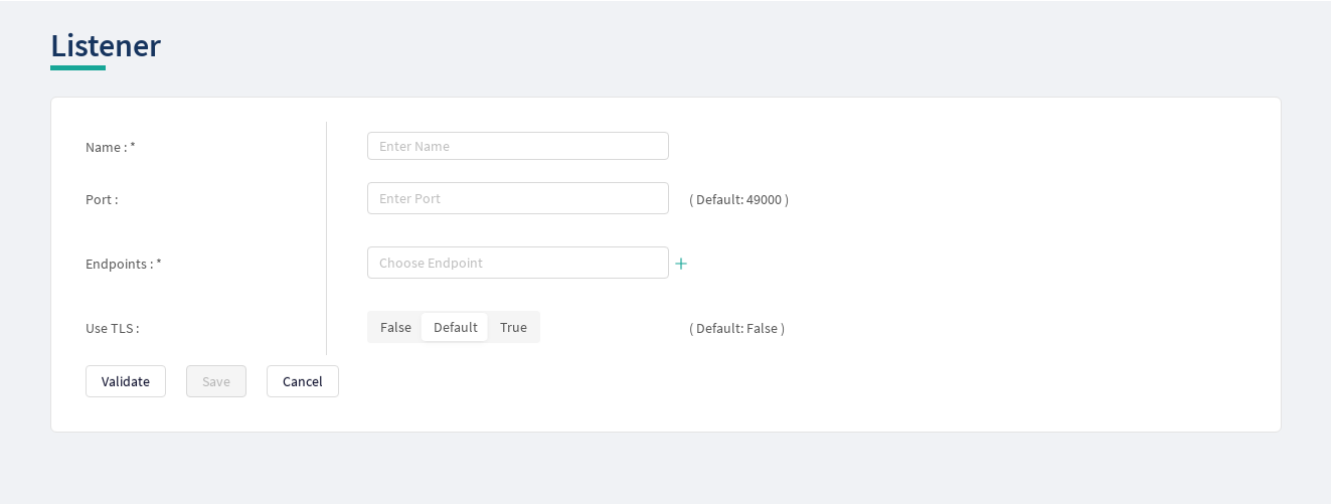


Figure 72. Configuring a listener in the Web User Interface

Table 64. Listeners' configuration options

Key	Values	Default value	Description
Name*	Free text. Alphanumeric, may contain underscores, may not start with a number.		The name identifying the listener. This name of the listener can be referenced from other parts of the configuration.
Port	Any port value can be defined. Note that the port value has to be within the range configured in the docker.	49000	The number of the port the listener binds to.
Endpoints*	A list of references to *Endpoint Services*.		The list of endpoint(s), as defined under Endpoint that serve traffic coming in on the listener.
Use TLS	True or False.	False	Enables using TLS in the connection towards the clients.
Client TLS*	Reference to a *TLS Brick* of type *Client TLS*.		The TLS configuration towards the clients. See TLS for details. Mandatory if *Use TLS* is set to *True*.

6.7. SYSTEM - Configuration units

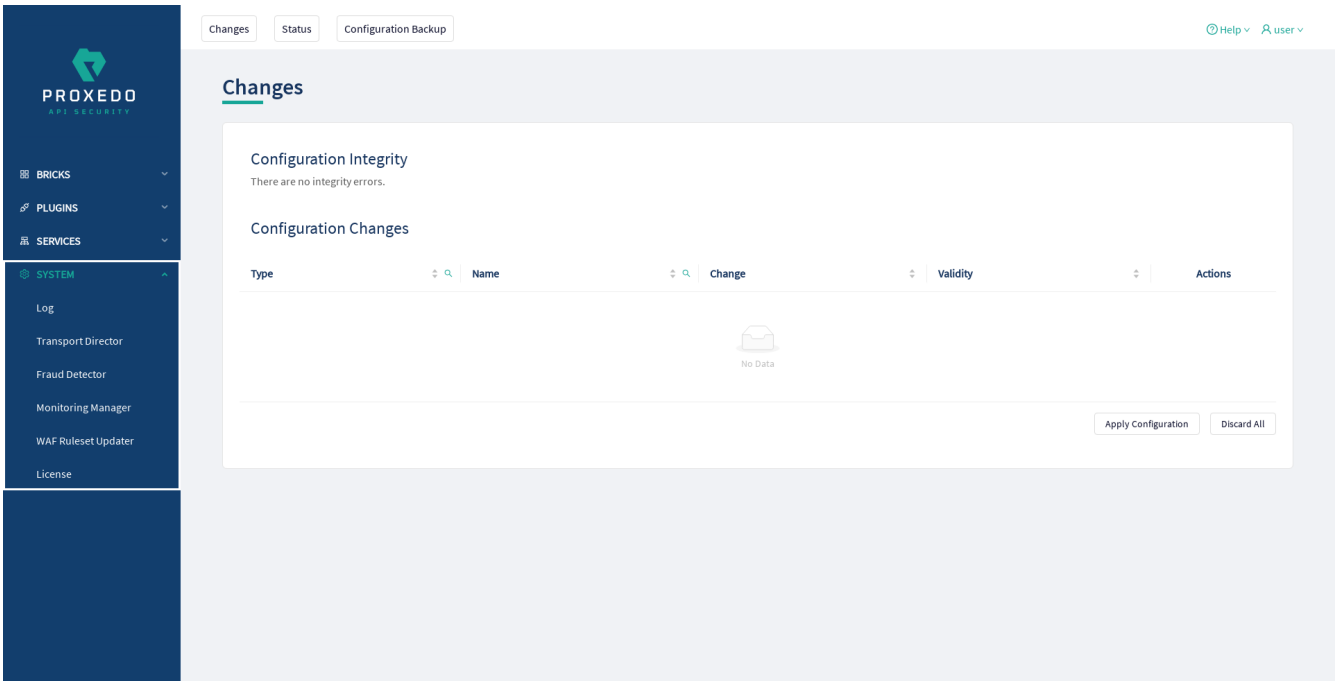


Figure 73. The SYSTEM main page in the Web User Interface

6.7.1. Log

If at any point an error occurs during the Security Flow, the error policy is applied and logging takes place if configured so.

6.7.1.1. Configuring Log

1. {step_open_systems}
2. Select *Log*.
3. Click the *Validate* button to check if the defined parameters are of the correct type and all required parameters have been filled out for configuring the component. If the configuration is erroneous or incomplete, the Web UI provides a warning that the 'Component validation failed'. Also a warning with information on the missing or faulty elements appears at the problematic field. If the configuration of the component is valid, after clicking the *Validate* button, a 'Component validation successful' notification is shown.
4. Save the component configuration by clicking the *Save* button.

Increasing the verbosity hugely increases the amount of logs generated, and will reduce performance.

The logs at the highest level of verbosity (9) might include sensitive information, such as passwords.

The following values can be configured for the *Log System*:

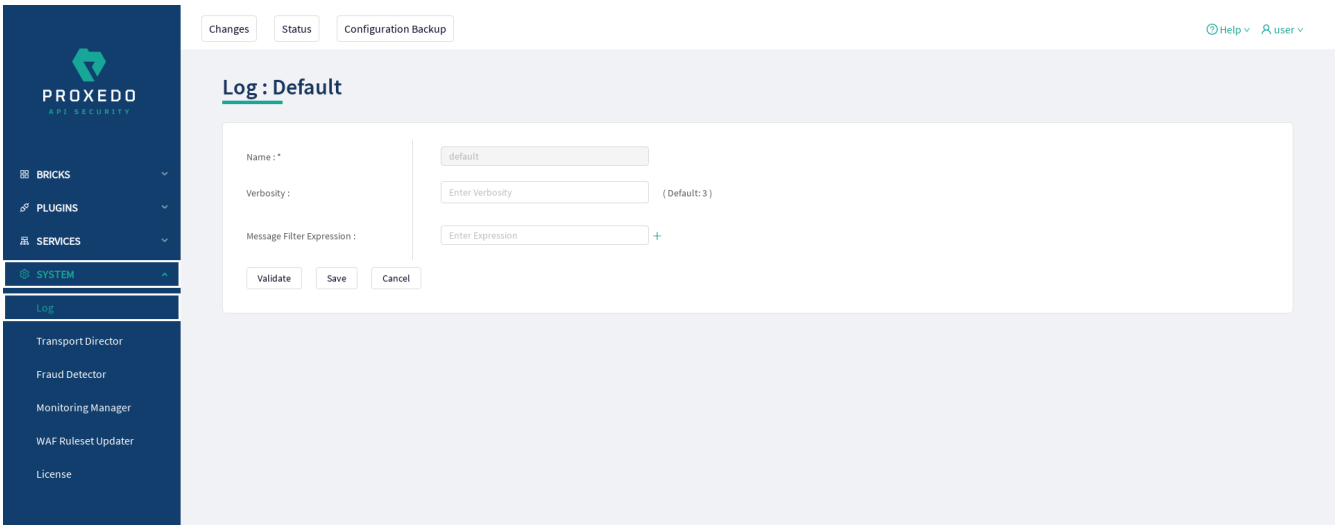


Figure 74. The main page for Logs

Table 65. Log configuration

Key	Values	Default value	Description
Name*	*Log* has a default name 'default', that cannot be changed.		The name identifying the log configuration.
Verbosity	The value can take number format.	3	The verbosity of logging. It must be between 1-9.
Message Filter Expression	A list of message filter expressions. A single message filter expression consists of a log category, a colon, and a number specifying the verbosity level of that given category. Categories match from left to right and wildcards can be used. For example: `http.*:5,core.info:3`. The last matching entry will be used as the verbosity of the given category. If no match is found the default verbosity specified with *verbosity* is used.	*.accounting :4, core.summary:4	Set verbosity mask on a per category basis. Each log message has an assigned multi-level category, where levels are separated by a dot.

6.7.2. Transport Director

The **Transport Director** manages the transport layer of API connections:

- handles network connections from the client
- handles network connections towards the backends
- handles TLS on these connections
- load-balances between multiple backend servers
- load-balances between multiple *Flow Directors*
- enforces HTTP protocol validity in calls

6.7.2.1. Configuring Transport Director

1. {step_open_systems}
2. Select *Transport Director*.
3. Click the *Validate* button to check if the defined parameters are of the correct type and all required

parameters have been filled out for configuring the component. If the configuration is erroneous or incomplete, the Web UI provides a warning that the 'Component validation failed'. Also a warning with information on the missing or faulty elements appears at the problematic field. If the configuration of the component is valid, after clicking the *Validate* button, a 'Component validation successful' notification is shown.

4. Save the component configuration by clicking the *Save* button.

The following values can be configured for the *Transport Director System*:

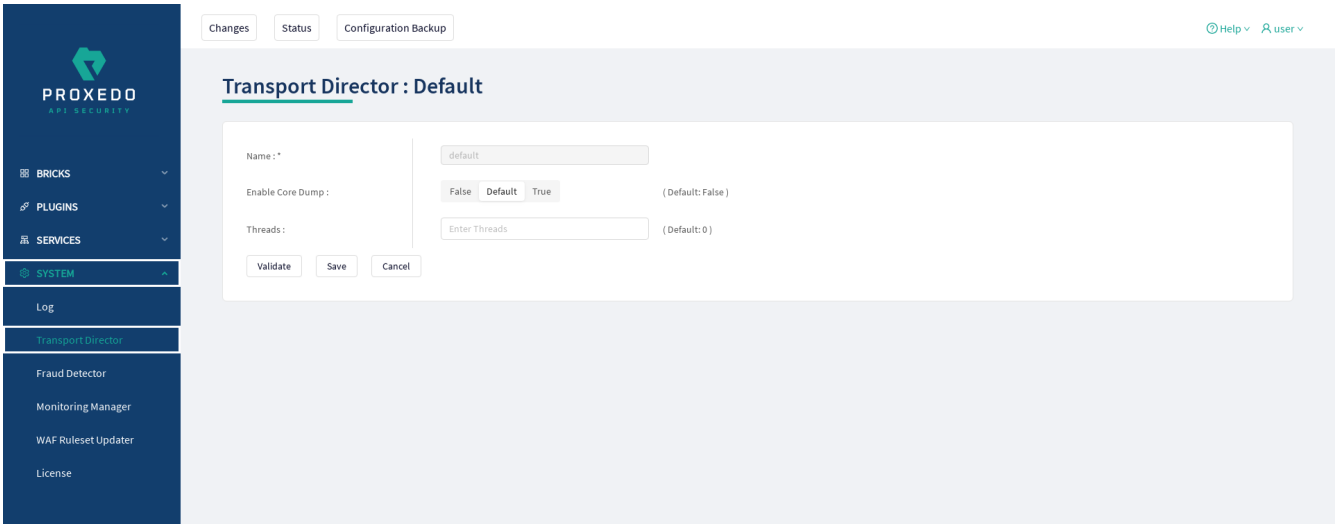


Figure 75. The main page for Transport Director

Table 66. Transport Director configuration

Key	Values	Default value	Description
Name*	*Transport Director* has a default name 'default', that cannot be changed.		The name identifying the Transport Director configuration. This name of the *Transport Director* can be referenced from other parts of the configuration.
Enable Core Dump	True or False.	False	Enables core dumps on failures.
Threads		0	Set the maximum number of threads that can be used in parallel. Note, that setting the value to zero means that the number of threads that can be used in parallel is unlimited.

6.7.3. Fraud Detector

The Fraud Detector, leveraging the data collected by the Fraud Detector plugin, establishes the actual connection with the Fraud API for an evaluation on the data of the calls.

Although the average response time of the Fraud API is half second, depending on the size and the complexity of the traffic to be investigated the response time might increase up to three seconds. Consequently, it is recommended to carefully identify the content selected for detection.

It is also recommended to consider that the API evaluates the maximum of 10 requests per second, therefore it is important to carefully define the matcher for the fraud detection, so that the load of requests is not unnecessarily high and the requests exceeding the value of 10 requests per second do not get dropped.

There are three recommended data types to be configured as selectors when configuring the Fraud Detector plugin, namely the IP address, the phone number and the e-mail address. For more details on how to configure Fraud Detector plugin, see Fraud Detector Plugin's configuration options.

6.7.3.1. Configuring Fraud Detector

1. {step_open_systems}
2. Select *Fraud Detector*.

Continue with the steps if the Fraud Detector is required in active state:

3. Set the Fraud Detector system to active state. The Fraud Detector is set to 'inactive' state by default, as for the 'active' state license is required.
4. Define the API Endpoint destination.
5. Fill in the API key. The API Key is provided together with the license purchased for the Fraud Detector.
6. Add the value for the Connection Timeout parameter. The value has to be provided in seconds.
7. Provide the value for the Response Timeout parameter. The value has to be provided in seconds.
8. Click the *Validate* button to check if the defined parameters are of the correct type and all required parameters have been filled out for configuring the component. If the configuration is erroneous or incomplete, the Web UI provides a warning that the 'Component validation failed'. Also a warning with information on the missing or faulty elements appears at the problematic field. If the configuration of the component is valid, after clicking the *Validate* button, a 'Component validation successful' notification is shown.
9. Save the component configuration by clicking the *Save* button.

The following values can be configured for the *Fraud Detector System*:

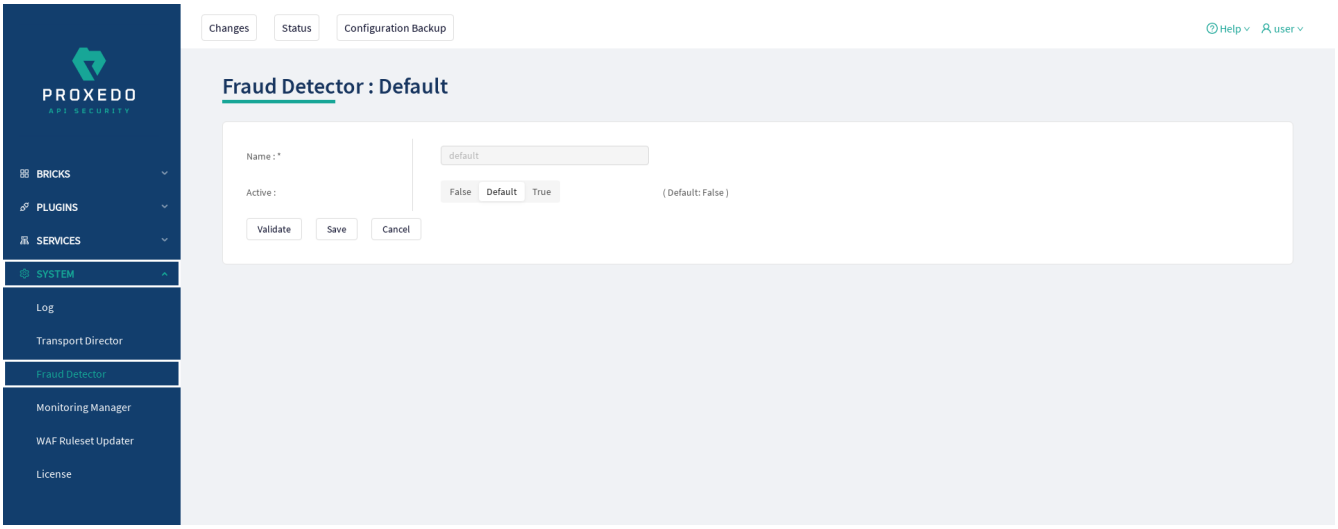


Figure 76. The Fraud Detector's main page in the Web User Interface

Table 67. Fraud Detector's configuration options

Key	Values	Default value	Description
Name*	*Fraud Detector* has a default name 'default', that cannot be changed.	default	The name identifying the Fraud Detector. This name of the Fraud Detector can be referenced from other parts of the configuration.
Active	True or False.	False	If the license for the Fraud Detector is purchased, the system can be activated.

If the Fraud Detector system is set to active, the following further parameters are available:

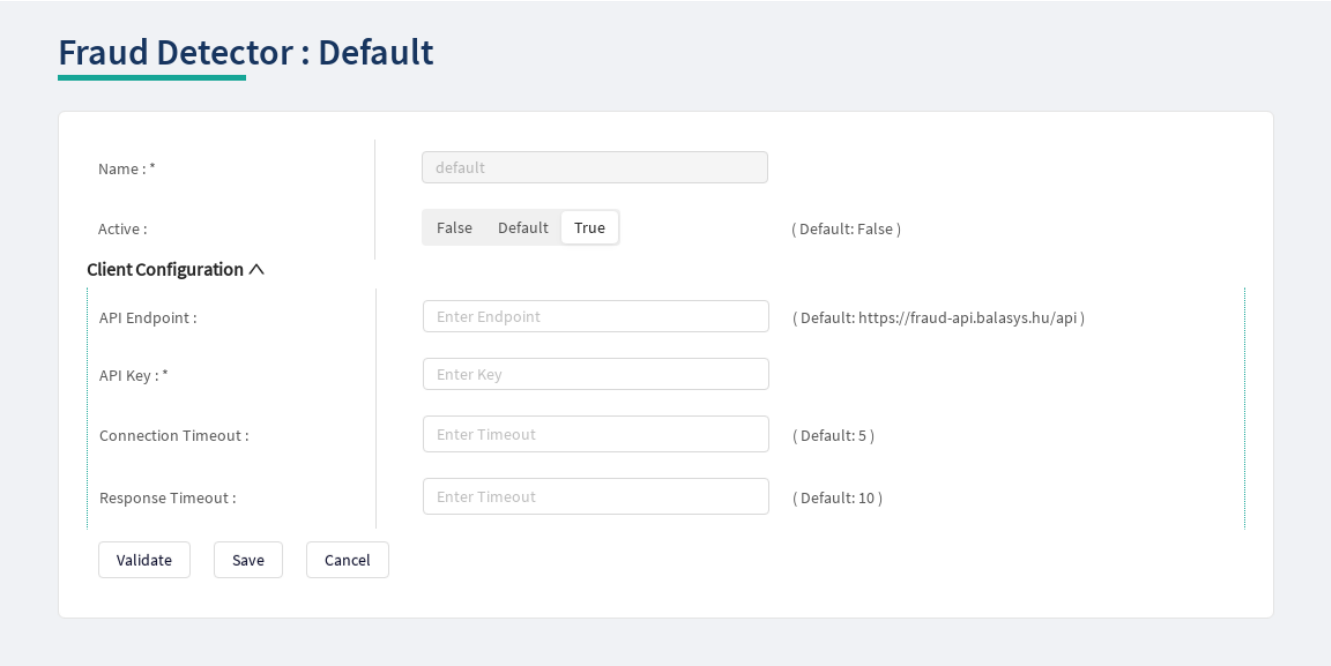


Figure 77. Configuring an active Fraud Detector in the Web User Interface

Table 68. The active Fraud Detector's configuration options

Key	Values	Default value	Description
Client Configuration			Configure the parameters of Fraud Detector.
API Endpoint		The default value is as follows: https://fraud-api.balasys.hu/api.	The API endpoint.
API Key*	The value for the API Key is provided by the purchase of the Fraud Detector license.		The API key is provided when the license for the Fraud Detector is purchased.
Connection Timeout	The value can be provided in seconds.	5	The time limit for establishing connection with the provided URL.
Response Timeout	The value can be provided in seconds.	10	The time limit for how long the PAS awaits the answer from the Fraud API after an established connection.

6.7.4. Monitoring Manager

The Monitoring manager systematically collects data on the components with the help of SNMP protocol. For the analysis of that data, the BALASYS-SNMP-MIB and the PAS-SNMP-MIB Management Information Base (MIB) documents can be downloaded from Balasys customer documentation. Further recommended MIB files for the analysis of this data are SNMPv2-MIB, IF-MIB and UCD-SNMP-MIB.

To see how monitoring data can be accessed, see chapter Monitoring in PAS.

6.7.4.1. Configuring Monitoring Manager

1. {step_open_systems}
2. Select *Monitoring Manager*.

The following values can be configured for the *Monitoring Manager System*:

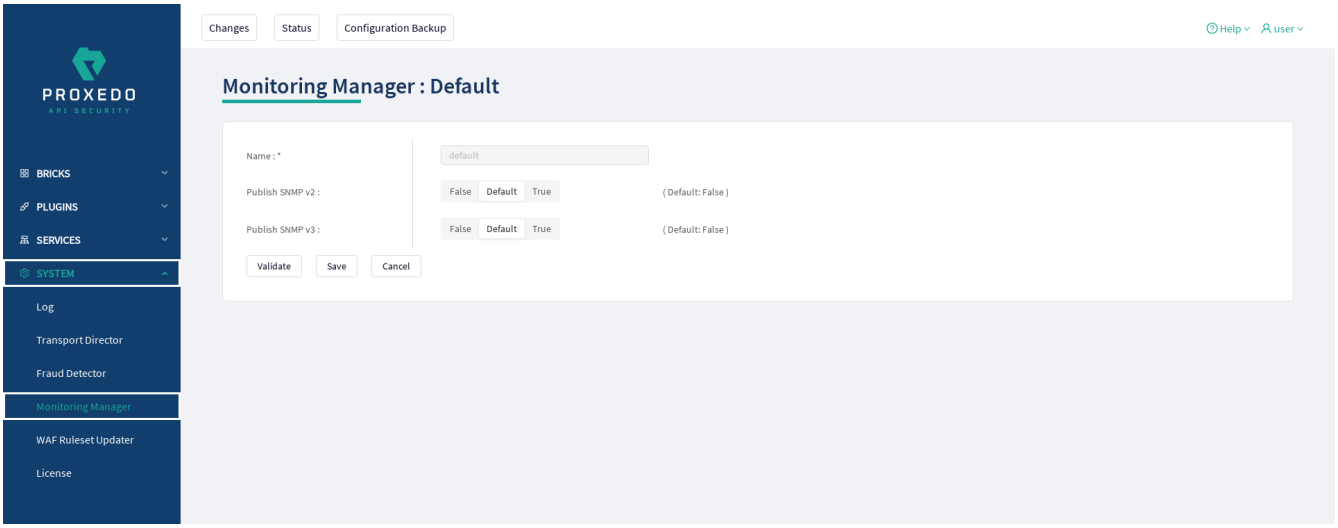


Figure 78. The main page for Monitoring Manager

Table 69. Monitoring Manager configuration

Key	Values	Default value	Description
Name*	*Monitoring Manager* has a default name 'default', that cannot be changed.		The name identifying the Monitoring Manager configuration.
Publish SNMP v2	True or False.	False	Enables publishing monitoring data using the SNMPv2 protocol. SNMPv2 is an improved SNMP protocol with community-based authentication.
SNMP v2 Authentication			Authentication settings to access PAS and host-related data.
SNMP v2 PAS Community Strings*	A list of accepted community strings.		The list of community strings to access data related to PAS.
SNMP v2 Host Community Strings*	A list of accepted community strings.		The list of community strings to access data related to the host.
Publish SNMP v3	True or False.	False	Enables publishing monitoring data using the SNMPv3 protocol. SNMPv3 is an SNMP protocol with user-based authentication and data encryption. Note, that in case SNMPv3 is used, all three parameters have to be filled in, that is, Username, Authentication password and Privacy password.
SNMP v3 Authentication			Authentication settings to access PAS and host-related data. SNMPv3 authentication requires to define all three values, such as Username, Authentication password and Privacy password.

Key	Values	Default value	Description
SNMP v3 Users*	The Username, the Authentication password and the Privacy password have to be provided as well for a complete SNMPv3 authentication.		Provide all the three values to achieve a secure SNMPv3 authentication.

3. Click the *Validate* button to check if the defined parameters are of the correct type and all required parameters have been filled out for configuring the component. If the configuration is erroneous or incomplete, the Web UI provides a warning that the 'Component validation failed'. Also a warning with information on the missing or faulty elements appears at the problematic field. If the configuration of the component is valid, after clicking the *Validate* button, a 'Component validation successful' notification is shown.
4. Save the component configuration by clicking the *Save* button.

6.7.5. WAF Ruleset Updater

The Web Application Firewall (WAF) Ruleset Updater System is designed to automatically update the ruleset used for WAF enforcers, it is thereby critical in ensuring real-time protection against zero-day attacks by maintaining an up-to-date defense mechanism.

To activate this system, extra credentials will be necessary which can be obtained from the Balasys sales team.

6.7.5.1. Configuring WAF Ruleset Updater

1. {step_open_systems}
2. Select *WAF Ruleset Updater*.

Continue with the steps if the WAF Ruleset Updater is required in active state:

3. Set the WAF Ruleset Updater system to active state. To activate the WAF Ruleset Updater a license is required. To acquire a license, contact our sales team at the e-mail address <sales@balasys.hu>.
4. Fill in the API Username. The API Username is provided together with the license purchased for the WAF API.
5. Fill in the API Password. The API Password is provided together with the license purchased for the WAF API.
6. Add the value for the Poll Interval Seconds parameter. The value has to be provided in seconds.
7. Add the value for the Connection Timeout Seconds parameter. The value has to be provided in seconds.
8. Provide the value for the Response Timeout Seconds parameter. The value has to be provided in seconds.
9. Click the *Validate* button to check if the defined parameters are of the correct type and all required parameters have been filled out for configuring the component. If the configuration is erroneous or incomplete, the Web UI provides a warning that the 'Component validation failed'. Also a warning with information on the missing or faulty elements appears at the problematic field. If the configuration of the component is valid, after clicking the *Validate* button, a 'Component validation successful' notification is shown.
10. Save the component configuration by clicking the *Save* button.

The following values can be configured for the *WAF Ruleset Updater System*:

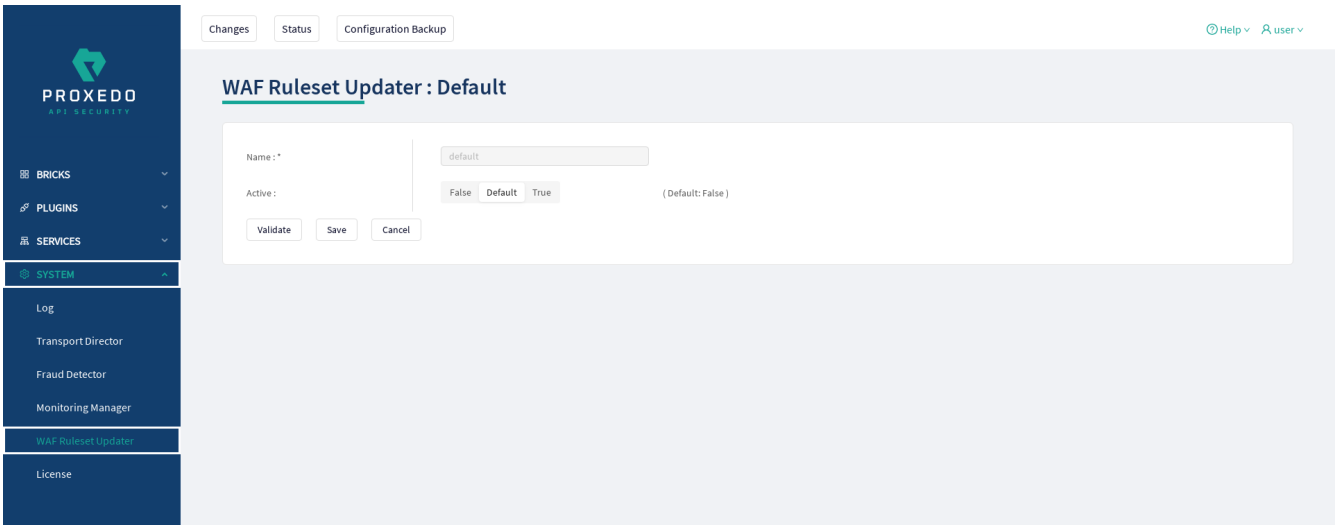


Figure 79. The WAF Ruleset Updater's main page in the Web User Interface

Table 70. WAF Ruleset Updater's configuration options

Key	Values	Default value	Description
Name*	*WAF Ruleset Updater* has a default name 'default', that cannot be changed.	default	The name identifying the WAF Ruleset Updater. This name of the WAF Ruleset Updater can be referenced from other parts of the configuration.
Active	True or False.	False	The system needs to be activated only if a *WAF Enforcer* is to be used.

If the WAF Ruleset Updater system is set to active, the following further parameters are available:

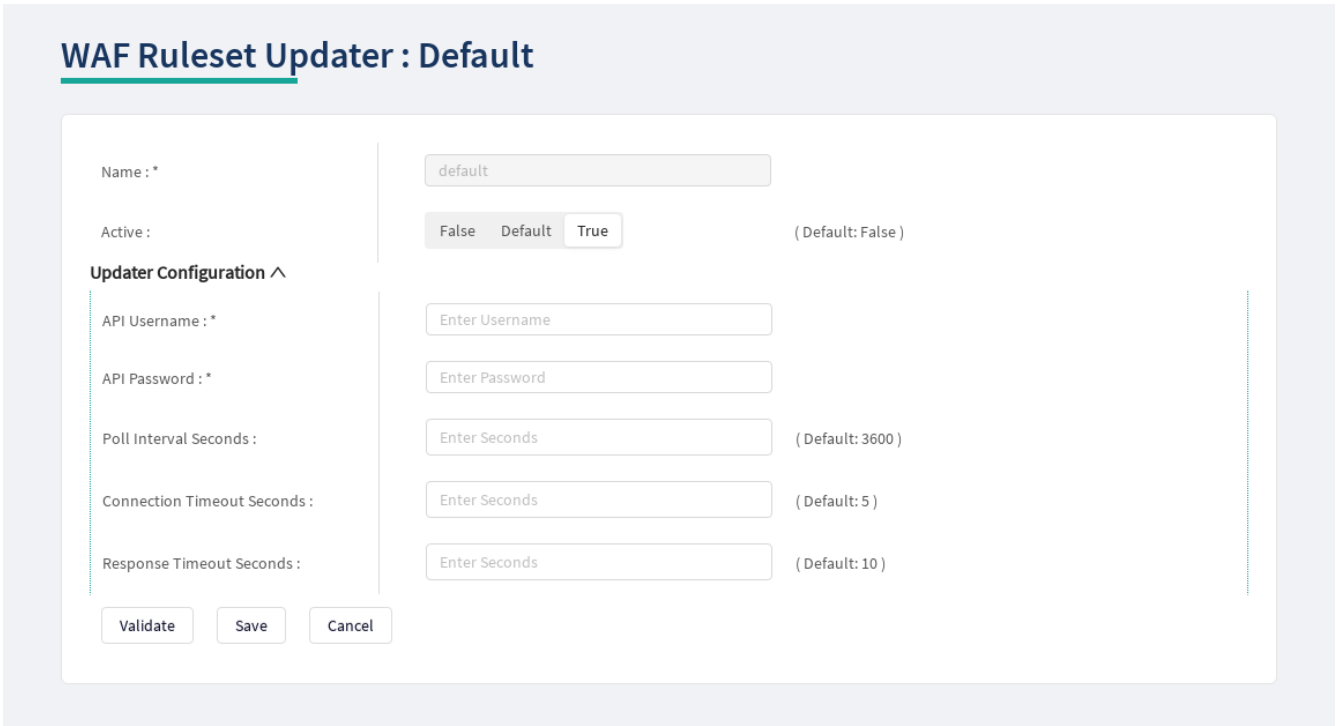


Figure 80. Configuring an active WAF Ruleset Updater in the Web User Interface

Table 71. The active WAF Ruleset Updater's configuration options

Key	Values	Default value	Description
Updater Configuration			Configure the parameters of WAF Ruleset Updater.
API Username*	The username required to download and update the WAF enforcer's ruleset. The value for the API Username is provided with the purchase of the WAF license.		The API Username is provided when the license for the WAF API is purchased.
API Password*	The password required to download and update the WAF enforcer's ruleset. The value for the API Password is provided with the purchase of the WAF license.		The API Password is provided when the license for the WAF API is purchased.
Poll Interval Seconds	The value must be provided in seconds.	3600	The time between two ruleset updates.
Connection Timeout Seconds	The value must be provided in seconds.	5	The time limit for how long the PAS awaits the answer from the WAF API to establish the connection.
Response Timeout Seconds	The value must be provided in seconds.	10	The time limit for how long the PAS awaits the answer from the WAF API after an established connection.

6.7.6. License

The License System holds the License File brick currently in use.

6.7.6.1. Configuring License

1. {step_open_systems}
2. Select *License*.
3. Choose an uploaded License File brick from the drop-down list.
4. Click the *Validate* button to check if the defined parameters are of the correct type and all required parameters have been filled out for configuring the component. If the configuration is erroneous or incomplete, the Web UI provides a warning that the 'Component validation failed'. Also a warning with information on the missing or faulty elements appears at the problematic field. If the configuration of the component is valid, after clicking the *Validate* button, a 'Component validation successful' notification is shown.
5. Save the component configuration by clicking the *Save* button.

The following values can be configured for the *License System*:

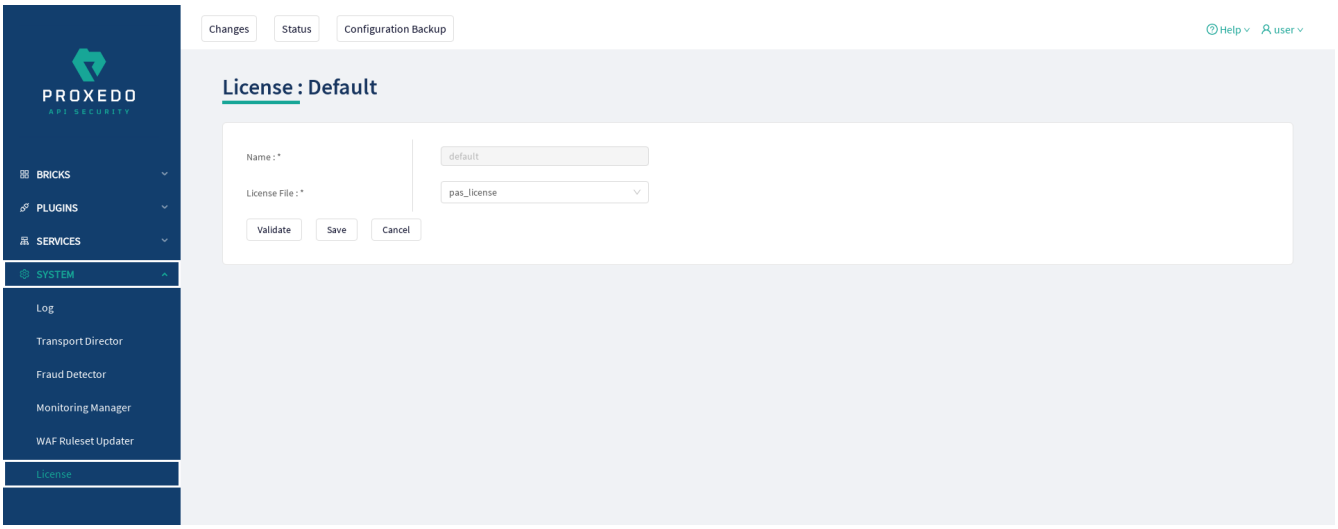


Figure 81. The License's main page in the Web User Interface

Table 72. License's configuration options

Key	Values	Default value	Description
Name*	*License* has a default name 'default', that cannot be changed.	default	The name identifying the License. This name of the License can be referenced from other parts of the configuration.
License File*	A reference to a *File Brick* of the License type.		The License File that is to be used.

6.8. System-wide status information

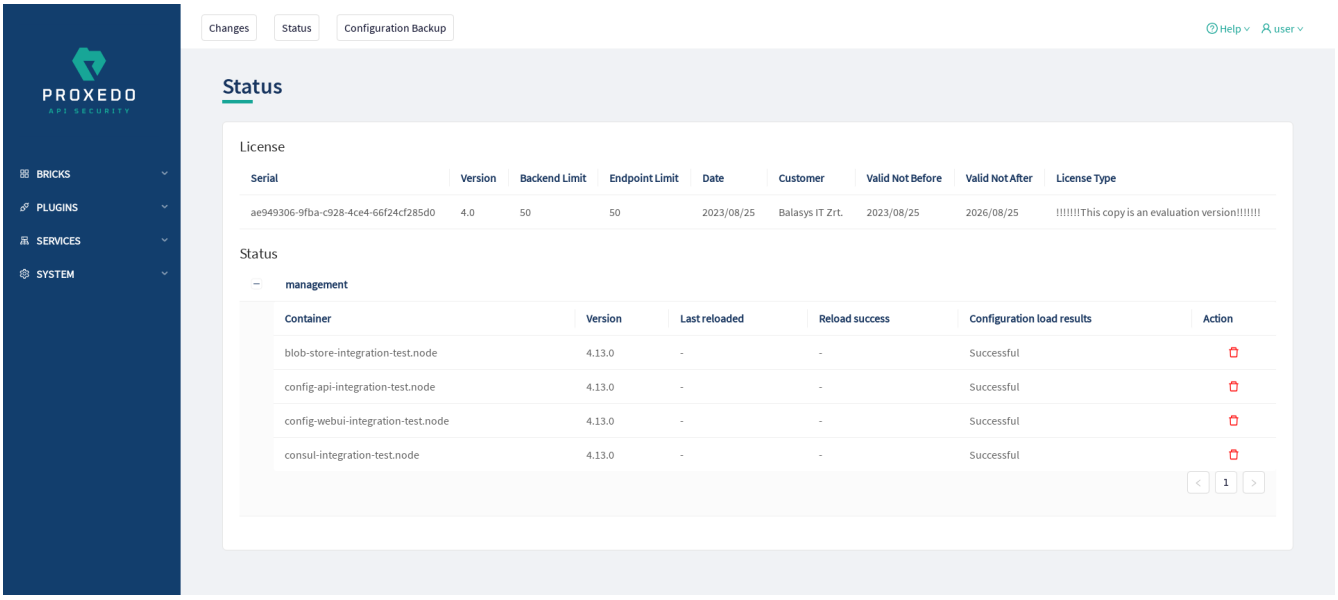


Figure 82. System-wide status information

6.8.1. License details

Details on the active license are presented in a table.

- **Serial**: The unique ID of the license.
- **Backend Limit**: The number of configured Backends that this license supports.

- **Endpoint Limit**: The number of configured Endpoints that this license supports.
- **Date**: The date this license was issued.
- **Customer**: The customer this license was issued to.
- **Valid Not Before**: The license is not valid before this date.
- **Valid Not After**: The license is not valid after this date.
- **License Type**: The type of this license.

6.8.2. Status information on the configuration of Proxedo API Security services

The administrator can check the status of the services and their containers. Whether a certain service runs properly or not might be a helpful piece of information in identifying some of the configuration problems.

The status information is presented for each node separately. Open up the details for each node by clicking on the plus sign next to the name of the node:

- **Container**: The component the PAS service runs in.
- **Version**: The version of the service, in practice this is the tag of the Docker container.
- **Last reloaded**: The exact timestamp of the last reload in ISO format.
- **Reload success**: Whether the reload was successful or not.
- **Configuration load result**: Whether loading the configuration was successful or not. The possible values are:
  - **Successful**
  - **Failed**
- **Action**: Click the  icon to delete the data for any configuration element.

6.9. Checking and finalizing changes in Proxedo API Security configuration

It is possible to list and check any changes made to the PAS configuration until the changes have not been applied with the *Apply Configuration* button.

Click on the *Changes* button in the Top-left navigation area to list the changes made to the configuration.

The following pieces of information are displayed:

- configuration integrity problems
- changes made to any of the configuration components

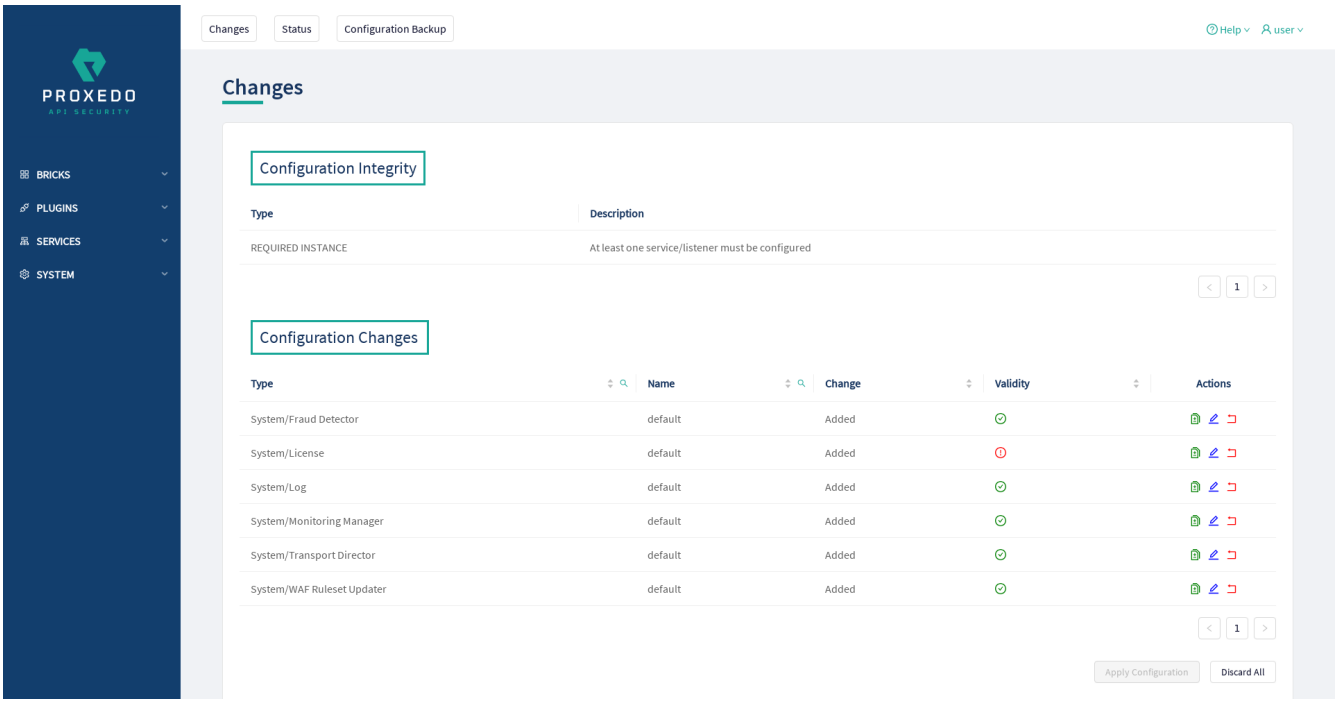


Figure 83. Checking changes made to the configuration

6.9.1. Configuration Integrity

For errors in configuration integrity, the following pieces of information are displayed in table format:

- **Type**: The type of the integrity problem, for example cycle detection.
- **Description**: Provides details on the nature of the integrity error.

Until the configuration integrity errors listed here are not corrected, the configuration cannot be applied.

For details on configuration integrity errors, see the examples in section Integrity errors.

6.9.2. Configuration Changes

For changes on the configuration components, the following pieces of information are displayed in table format:

- **Type**: Type denotes the category (Brick, Plugin, Service, System) and the class (for example, Matcher, Filter, Log) of the configuration component, for example Brick/Matcher.
- **Name**: The name of the configuration component is displayed here, to which the actual change has been made.
- **Change**: The nature of the change made to the configuration component is provided here, that is, *Added*, *Edited*, *Deleted* or *Broken Reference* (no direct change, but it refers to another component that is now missing or invalid).
- **Validity**: This field informs the user on whether the configured component is valid or not, as follows:
  - - Any instance marked with this sign is invalid.
  - - Any instance marked with this sign is valid.

	Click on the ⓘ sign to see more information on why the instance was found invalid.

Invalid configuration components can be corrected and revalidated by using the *Validate* button, available at each component's configuration page. For more information, see section *Component-level validation* in chapter Applying and validating Proxedo API Security configuration.

- **Actions**: This field provides several actions that can be taken.
  - Difference - Show the difference between the edited and the running versions of the component.
  - Edit - Edit the configuration data for a component. If the edit button is disabled, it means that the instance has been deleted.
  - Discard - Undo any configuration changes to a component. If the discard button is disabled, no direct changes have been made to the actual component.

By selecting the *Discard All* button, it is possible to discard all changes made to the configuration. However, the changes that have been applied to the configuration already and the factory-supplied configuration elements cannot be discarded.

6.10. Applying and validating Proxedo API Security configuration

PAS configuration can be checked and validated on two levels:

- component-level validation
- validating the whole configuration

6.10.1. Component-level validation

Component-level validation takes place while configuring the actual elements of the configuration and by using the *Validate* button on the Web UI page of the specific component.

If the configuration of the component is erroneous or not adequate, the Web UI provides a warning that the *Component validation failed*. Also a warning with information on the missing details appears at the problematic field for the user.

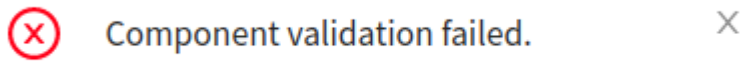


Figure 84. Component validation failed

If the configuration of the component is satisfactory, after clicking the *Validate* button, the user receives the *Component Validation successful* notification. Click *OK*. For related errors see, section Validation errors.

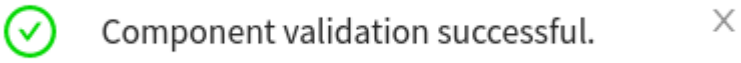


Figure 85. Component validation successful

6.10.2. Validating the whole configuration

Configuration integrity errors can be viewed on the *Changes page*, along with a summary of valid and invalid component changes. To make it available click the *Changes* button so that all the changes made to any component by the user will be visible. For related errors, see section Validation errors.

6.10.3. Applying the whole configuration

The *Apply Configuration* button is available from the *Changes* page. To make it available click the *Changes* button so that all the changes made to any component by the user will be visible. In order to take the changes into effect, click the *Apply Configuration* button. The configuration can only be applied if all changes are valid. When applying the configuration by using the *Apply Configuration* button, the Web UI provides either of the following messages:

- The configuration is applied successfully. Click *OK*.

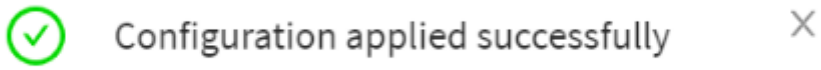


Figure 86. Apply Configuration result - successful

- The configuration failed.
  If applying the configuration failed, the Web UI also provides an additional pop-up window with the description of the problem. The problems can be as follows:
  - At least one of the services failed to start, the previous configuration settings have been restored.
  - Restoring the original configuration was not successful.

During the process of applying the configuration, no changes can be completed to the configuration. The process however shall not take more than 10 seconds.

6.10.4. Validation errors

In case the configuration could not be applied, the following result messages help the user to correct the configuration and achieve a valid configuration.

6.10.4.1. Component-related errors

These errors are the results of the validation of the actual components. By correcting these the user can achieve a functioning configuration.

6.10.4.1.1. Missing data for required fields

Each component has compulsory configuration fields that must be filled in. In case any of those fields are left empty, the Web UI provides a *Missing data for required field* notification when the component is validated, that is, the *Validate* button is used. Each compulsory field is highlighted with a * sign.

Example
The *Insight Target* component requires the *Host* field to be filled in, otherwise the component's configuration is not valid.

Error message: **Missing data for required field.**

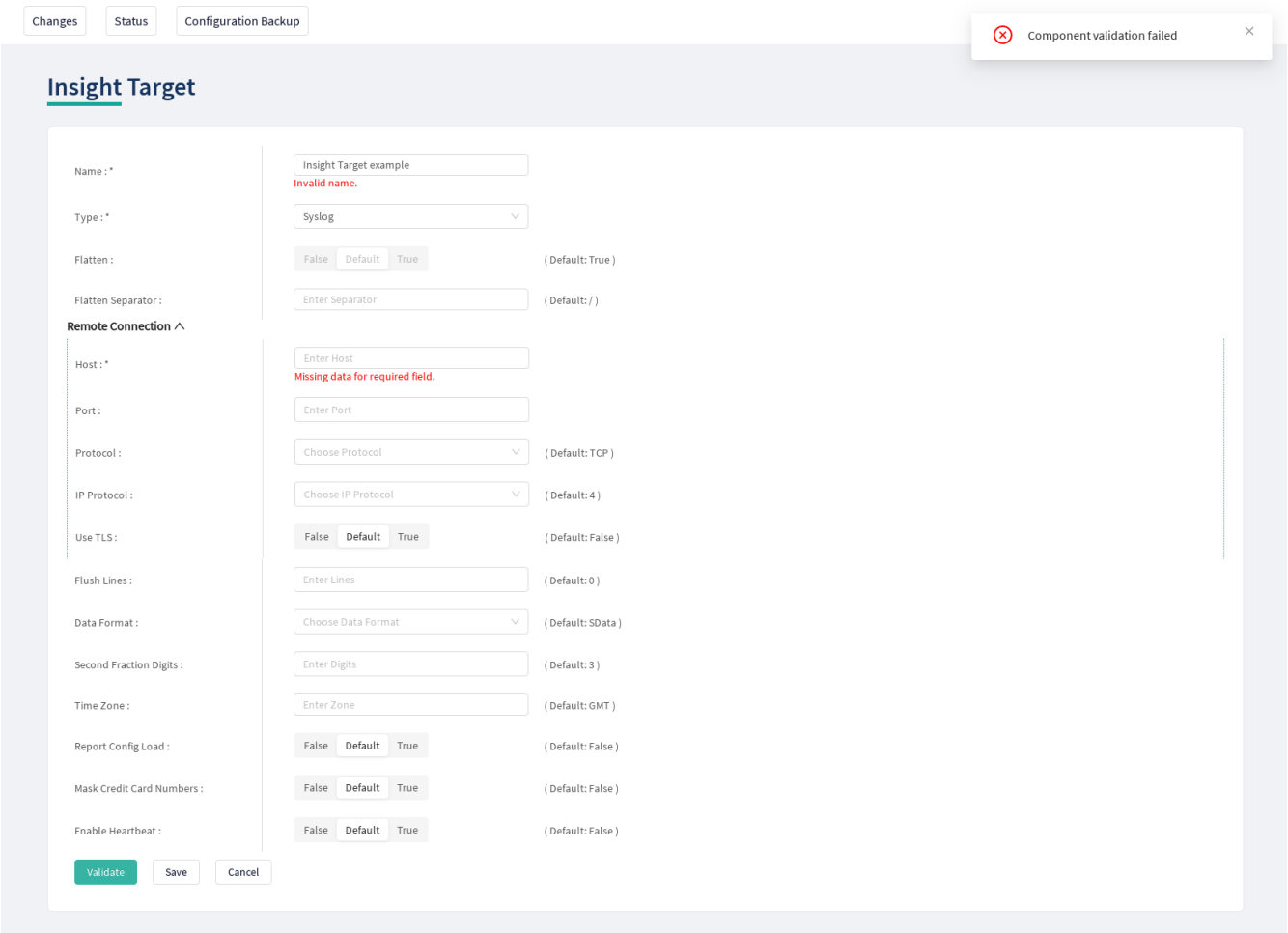


Figure 87. Missing required field - Insight Target

6.10.4.1.2. Missing reference

This error indicates that the component references a non-existing component.

Example
The user creates an Error Policy, *error_policy_1* which is referenced in a Filter. Following that, this specific Error Policy, *error_policy_1* is deleted from the configuration. This results in a missing reference in the Filter.

Error message: **Reference to a non-existing component: error_policy_1.**

To correct the missing reference, navigate to the Filter component. In order to clear the invalid reference to the missing component, the  icon has to be selected on the right side of the Error Policy drop-down list. By clicking this icon, the configuration data is cleared from this selection.

6.10.4.1.3. Port conflict

This error indicates that two or more Listeners are configured to use the same port. This leads to a failed configuration.

Example
Two Listeners are configured to use the same port.

Error message: **listener_1 uses the same port as listener_2.**

6.10.4.2. Integrity errors

6.10.4.2.1. Cycle detection

Error message: **Reference cycle detected in configuration: brick/matcher/matcher_1→brick/matcher/matcher_2→brick/matcher/matcher_1.**

This error indicates that there is a cycle of references between component instances.

Example
If the compound matcher *matcher_1* is configured to reference the compound matcher *matcher_2* and the compound matcher *matcher_2* is also referencing the compound matcher *matcher_1*, there will be a cycle of references between these two matchers.

6.10.4.2.2. Required instance is missing

Error message: **At least one service/listener must be configured.**

This error indicates that a required instance is not configured.

6.10.4.2.3. Fraud Detector plugin configured with the Fraud Detector system in inactive state

Error message: **Fraud Detector Plug-in must not be part of Security Flows while the Fraud Detector System is disabled.**

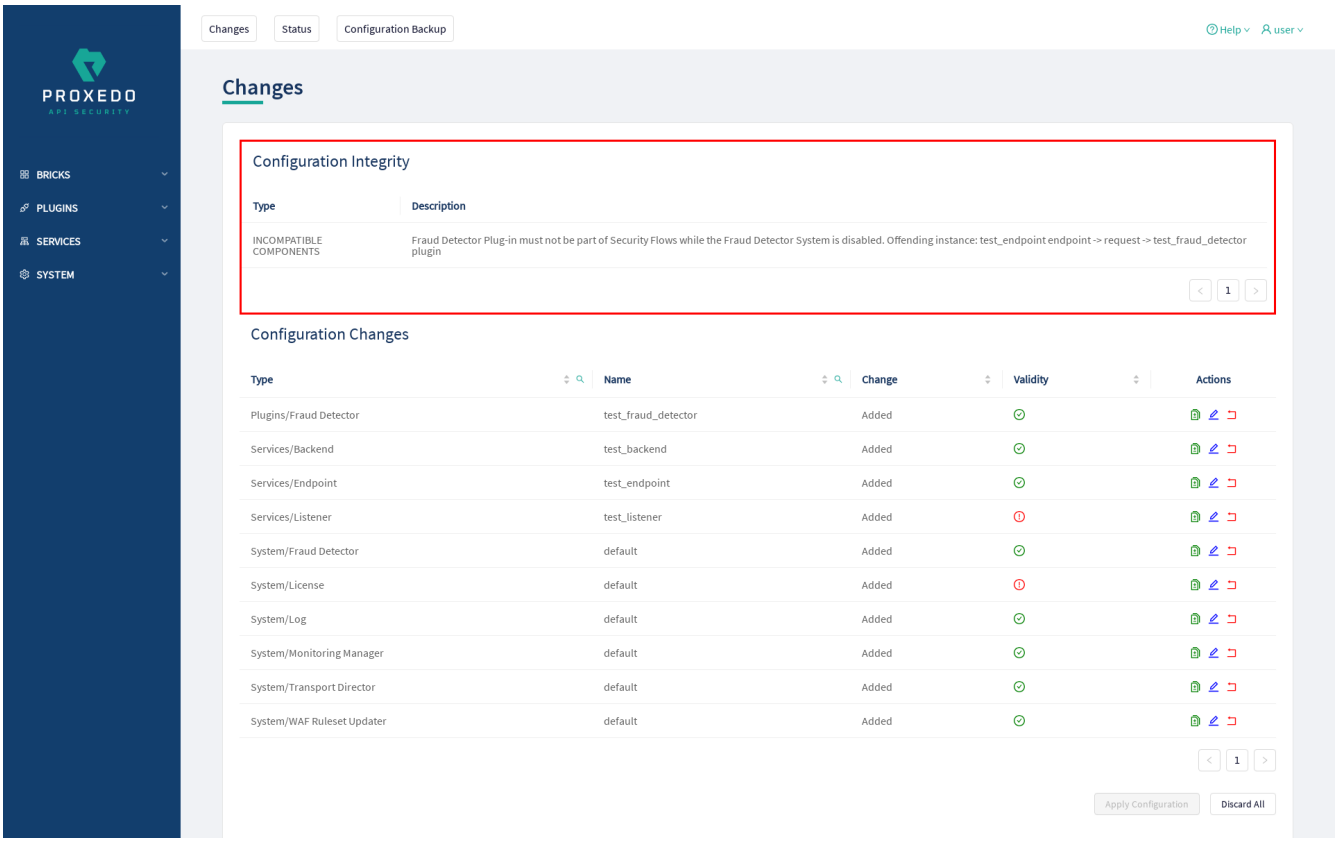


Figure 88. Fraud Detector plugin integrity error

This error indicates that there is a Fraud Detector plugin configured, however, the Fraud Detector system is not activated. In order to solve this integrity error, either the Fraud Detector plugin has to be removed from the configuration, or, in case the license for the Fraud Detector is purchased, the Fraud Detector system has to be activated and configured.

6.10.4.2.4. WAF Enforcer plugin configured with the WAF Ruleset Updater system in inactive state

Error message: **WAF Enforcer Plug-in cannot be cannot be configured if the WAF Ruleset Updater System is disabled.**

This error indicates that there is a WAF Enforcer plugin configured, however, the WAF Ruleset Updater system is not activated. In order to solve this integrity error, either the WAF Enforcer plugin has to be removed from the configuration, or, in case the license for the WAF Ruleset is purchased, the WAF Ruleset Updater system has to be activated and configured.

6.10.4.2.5. Insight Message field collision

Error message: **The message keys of some JSON formatted Syslog Insight Targets are in conflict with the Save As Key fields of some Selectors.**

This error indicates that there is an Insight that contains Selectors and Targets that have conflicting configurations. Insight plugins have a Message field that can identify the originating plugin in a log. Certain Insight Target bricks can be configured using the Include Message field to include this Message field in the log line even if the data format does not support a dedicated field for this (JSON for example). In this case, the field where the message should be keyed to can be configured using the Message Key field on the Insight Target. This key can conflict with the Save As Key field of Selectors in the same Insight, leading to lost data.

6.10.4.2.6. License limit exceeded

Error message: **The number of configured Backends (11) exceeds the limit allowed by the active license (10).**

This error indicates that the Backend or Endpoint limit of the selected license is exceeded. Either a different license should be purchased and configured in the License system, or fewer Backend or Endpoint services configured to stay below the limit.

Error message: **The active license's version (2.0) is not valid for this product version.**

This error indicates that the active license is not supported by the installed product version. This typically occurs when the license was issued for an older or newer product version than the currently running.

6.11. Backup and restore running or user configuration for Proxedo API Security

It is possible to backup and restore the Proxedo API Security configuration in the Web UI.

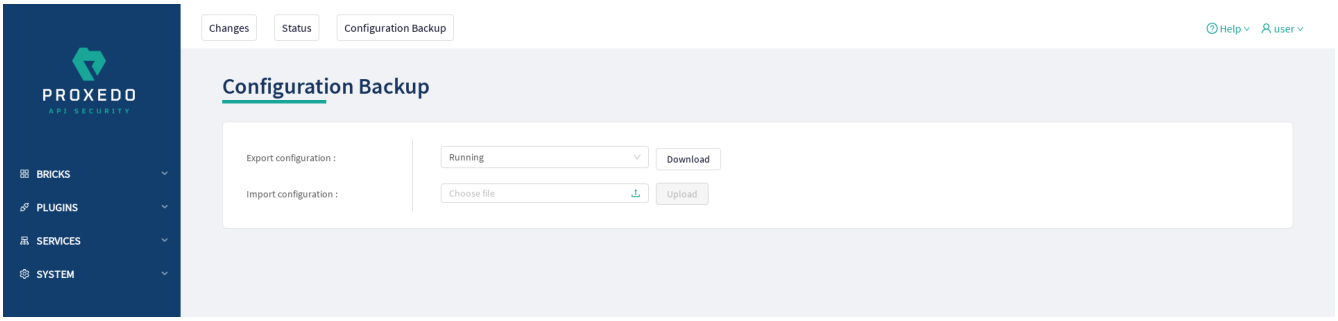


Figure 89. Backup and restore running or user configuration for Proxedo API Security

In order to export any configuration information from the system, complete the following steps:

1. Select the *Configuration Backup* button.
2. To export a configuration, select the type of the configuration to be exported at the *Export configuration* button. The following options can be selected from the drop-down menu:
    - Running: This export option downloads the configuration settings of the currently running configuration.
    - User: This export option downloads the default configuration settings of the system.

The configuration will be downloaded in .zip file format.

3. To import an existing configuration file, select the empty field beside *Import configuration*. Only .zip file formats can be uploaded.
4. Select the *Download* or the *Upload* buttons to finish the activity. The system will ask you to define the *Insight Target* or source destination for the activity. Note that only files in *.zip* format can be downloaded or uploaded.

In case of importing a configuration file, the system will notify the user that by importing a configuration file, the existing configuration will be overwritten: 'This operation overwrites user configuration. Are You sure?'

7. Operation of Proxedo API Security based on VMs

7.1. Operation of dockerd

Dockerd is managed through `systemd`, so common administration tasks are carried out through its interfaces.

Checking the status of docker

```
systemctl status docker
```

Example output

```
docker.service - Docker Application Container Engine
   Loaded: loaded (/usr/lib/systemd/system/docker.service; enabled; vendor preset:
disabled)
   Active: active (running) since Mon 2017-07-10 08:25:38 CEST; 4h 1min ago
     Docs: https://docs.docker.com
 Main PID: 2148 (dockerd)
    Tasks: 177 (limit: 4915)
   Memory: 119.1M
      CPU: 1min 36.272s
   CGroup: /system.slice/docker.service
           ├─2148 /usr/bin/dockerd
           ├─2185 docker-containerd -l unix:///var/run/docker/libcontainerd/docker-
containerd.sock --metrics-interval=0 --start-timeout 2m --state-dir
/var/run/docker/libcon
           ├─2542 docker-containerd-shim
fef20e5205c47b5cc18e612903a33e749ebd89a4bf30fd5bb8fb4a801450c84f
/var/run/docker/libcontainerd/fef20e5205c47b5cc18e612903a33e749ebd8
           ├─2582 docker-containerd-shim
410f0bc67c731635a7d60e9f259d2f62ef8a845e09595254217decd3b3885473
/var/run/docker/libcontainerd/410f0bc67c731635a7d60e9f259d2f62ef8a8
           ├─2704 /usr/bin/docker-proxy -proto tcp -host-ip 0.0.0.0 -host-port 5000
-container-ip 172.18.0.2 -container-port 5000
           ├─2732 docker-containerd-shim
3853efde62d1767e70372584812df07968a647f40039691d82ccd5cbc66ee32d
/var/run/docker/libcontainerd/3853efde62d1767e70372584812df07968a64
           ├─2770 /usr/bin/docker-proxy -proto tcp -host-ip 0.0.0.0 -host-port 8484
-container-ip 172.18.0.2 -container-port 443
           ├─2806 /usr/bin/docker-proxy -proto tcp -host-ip 0.0.0.0 -host-port 8181
-container-ip 172.18.0.2 -container-port 80
           ├─2832 /usr/bin/docker-proxy -proto tcp -host-ip 0.0.0.0 -host-port 2222
-container-ip 172.18.0.2 -container-port 22
           ├─2837 docker-containerd-shim
e24a8f2f189467601edb6bee0e63451e7230726feab50d43556e6c66a8f9fc56
/var/run/docker/libcontainerd/e24a8f2f189467601edb6bee0e63451e72307
```

```
           ├─2921 docker-containerd-shim
8ac62e1eee0d162e632eab95b08ea36aff69abd5d1eeac475bfee3f393cba179
/var/run/docker/libcontainerd/8ac62e1eee0d162e632eab95b08ea36aff69a
           ├─2974 docker-containerd-shim
6df61a17c29a132cb5886a494fc34e38ff38f2cf470919289c783fada579a70c
/var/run/docker/libcontainerd/6df61a17c29a132cb5886a494fc34e38ff38f
           ├─3043 docker-containerd-shim
d00a1de3994e2b11ecd93d938dc94702f4f6d0364d2f3c1c423ab2a1ec5c843a
/var/run/docker/libcontainerd/d00a1de3994e2b11ecd93d938dc94702f4f6d
           ├─3123 docker-containerd-shim
b9e93059835c2d343c912c7f7154b14625dcd2e8d242fd67328e9532e5829d64
/var/run/docker/libcontainerd/b9e93059835c2d343c912c7f7154b14625dcd
           ├─3187 docker-containerd-shim
2d058ab3987f2461c5f0029505eca264f94d34ed23c8464bfd83362ad9bcd142
/var/run/docker/libcontainerd/2d058ab3987f2461c5f0029505eca264f94d3
           └─3258 docker-containerd-shim
882c51a1a693230ea2d84f2f1a422655f9051d3a21a5f916a03e62614b17ed4a
/var/run/docker/libcontainerd/882c51a1a693230ea2d84f2f1a422655f9051
```

Starting docker

```
systemctl start docker
```

Stopping docker

```
systemctl stop docker
```

Configuring docker to start automatically

```
systemctl enable docker
```

Configuring docker not to start automatically

```
systemctl disable docker
```

7.2. Operation of services

The services of PAS are consolidated under the user *pas* who has privileges for common administration tasks.

Unless otherwise noted administrative commands should therefore be run as *pas* and not as *root*. This is especially true for docker compose commands.

7.2.1. Helper scripts for PAS in general

To help administrators with frequent PAS-related operations, we provide helper scripts in `/opt/balasys/bin`. To get the full list of helper scripts, just list this directory.

All script names are prefixed with the name of the component they correspond to. Therefore they follow the scheme `pas-<component-name>-script-name` except for core where the `component-name` part is omitted.

7.2.2. Checking configuration with `pas-*-checkconfig`

Some components of PAS have a textual configuration file, and it is possible to check them without actually starting the component. For different components, the following helper scripts are available.

Component	Helper script
HA	pas-ha-checkconfig
Storage	pas-storage-checkconfig

Component	Helper script
Management	pas-mgmt-checkconfig

`pas-*-checkconfig` can check the validity of PAS bootstrap configuration. Namely:

- it makes sure that the mandatory containers are defined in docker-compose.yml.
- it checks config.yml against the defined constraints one by one.

When running `pas-storage-checkconfig` on the management node, the output displays a parameter value for bootstrap configuration. Note, that currently bootstrap parameter values cannot be changed or configured and this warning is expected on the management node. See the following output example:

```
BootstrapExpect is set to 1; this is the same as Bootstrap mode.
bootstrap = true: do not enable unless necessary
```

docker-compose.yml is only checked for making sure that services have a proper image tag.

Currently, there is no configuration checker script for the core component.

7.2.3. Checking services

7.2.3.1. Storage service

Checking the status of PAS storage

```
systemctl status proxedo-api-security-storage
```

Listing the status of the services

```
docker compose -f /opt/balasys/etc/infrastructure/storage/docker-compose.yml ps
```

Example output

```
      Name                     Command               State                         Ports
-------------------------------------------------------------------------------------------
------------------
pas_blob-store_1   /usr/bin/dumb-init /usr/lo ...   Up      0.0.0.0:9000-
>9000/tcp,:::9000->9000/tcp
pas_consul_1       /usr/bin/dumb-init /usr/lo ...   Up      0.0.0.0:8300-
>8300/tcp,:::8300->8300/tcp, [...]
```

Checking which images are used by the services

```
docker compose -f /opt/balasys/etc/infrastructure/storage/docker-compose.yml images
```

Example output

```
   Container                       Repository                    Tag       Image Id
Size
-------------------------------------------------------------------------------------------
-------
pas_blob-store_1   docker.balasys.hu/api-security/blob-store     4.7.0   40bdc2d7665e
```

```
434.4 MB
pas_consul_1       docker.balasys.hu/api-security/consul         4.7.0   afd247e1e8c4
565.4 MB
```

7.2.3.2. Management service

Checking the status of PAS management

```
systemctl status proxedo-api-security-mgmt
```

Listing the status of the services

```
docker compose -f /opt/balasys/etc/infrastructure/mgmt/docker-compose.yml ps
```

Example output

```
      Name                     Command               State                         Ports
-------------------------------------------------------------------------------------------
--------------
pas_config-api_1   /usr/bin/dumb-init /usr/lo ...   Up      8080/tcp
pas_frontend_1     /usr/bin/dumb-init /usr/lo ...   Up      0.0.0.0:80->8080/tcp,
0.0.0.0:443->8443/tcp
```

Checking which images are used by the services

```
docker compose -f /opt/balasys/etc/infrastructure/mgmt/docker-compose.yml images
```

Example output

```
   Container                        Repository                    Tag       Image Id
Size
-------------------------------------------------------------------------------------------
-------
pas_config-api_1   docker.balasys.hu/api-security/config-api     4.7.0   025bf7529113
346.7 MB
pas_frontend_1     docker.balasys.hu/api-security/config-webui   4.7.0   1f2536bf1cf2
344.3 MB
```

7.2.3.3. Core service

Checking the status of PAS

```
systemctl status proxedo-api-security
```

Listing the status of the services

```
docker compose -f /opt/balasys/etc/infrastructure/pas/docker-compose.yml ps
```

Example output

```
              Name                            Command               State
Ports
-------------------------------------------------------------------------------------------
-------------------------------------
pas_content-filtering-director_1   /usr/bin/dumb-init /usr/lo ...   Up      1318/tcp
pas_flow-director_1                /usr/bin/dumb-init /usr/lo ...   Up
pas_flow-director_2                /usr/bin/dumb-init /usr/lo ...   Up
pas_insight-director_1             /usr/bin/dumb-init /usr/lo ...   Up
pas_monitoring-manager_1           /usr/bin/dumb-init /usr/lo ...   Up      0.0.0.0:161-
>161/udp,:::161->161/udp
pas_transport-director_1           /usr/bin/dumb-init /usr/lo ...   Up
0.0.0.0:49000->49000/tcp, 0.0.0.0:49001->49001/tcp
```

Checking which images are used by the services

```
docker compose -f /opt/balasys/etc/infrastructure/pas/docker-compose.yml images
```

Example output

```
           Container                                                Repository
Tag       Image Id       Size
-------------------------------------------------------------------------------------------
-----------------------------------
pas_content-filtering-director_1   docker.balasys.hu/api-security/content-filtering-
director   4.7.0   f6edae8b2d1b   420.7 MB
pas_flow-director_1                docker.balasys.hu/api-security/flow-director
4.7.0   a2b7ccc88823   441 MB
pas_flow-director_2                docker.balasys.hu/api-security/flow-director
4.7.0   a2b7ccc88823   441 MB
pas_insight-director_1             docker.balasys.hu/api-security/insight-director
4.7.0   db005e0fa5b6   331.9 MB
pas_monitoring-manager_1           docker.balasys.hu/api-security/monitoring-manager
4.7.0   772becf42dbe   467.6 MB
pas_transport-director_1           docker.balasys.hu/api-security/transport-director
4.7.0   c53bfaed2db0   377.1 MB
```

7.2.4. Starting and stopping services

PAS will ensure that containers are always clean on startup, which means that manual changes to the containers will not persist after a restart.

7.2.4.1. Storage service

Starting PAS storage

```
systemctl start proxedo-api-security-storage
```

Stopping PAS storage

```
systemctl stop proxedo-api-security-storage
```

Restarting PAS storage

```
systemctl restart proxedo-api-security-storage
```

pas-storage-checkconfig is invoked prior to (re)starting and reloading the service. The requested operation is interrupted if `pas-storage-checkconfig` fails.

Configuring PAS storage to start automatically

```
systemctl enable proxedo-api-security-storage
```

Configuring PAS storage not to start automatically

```
systemctl disable proxedo-api-security-storage
```

7.2.4.2. Management service

Starting PAS management

```
systemctl start proxedo-api-security-mgmt
```

Stopping PAS management

```
systemctl stop proxedo-api-security-mgmt
```

Restarting PAS management

```
systemctl restart proxedo-api-security-mgmt
```

pas-mgmt-checkconfig is invoked prior to (re)starting and reloading the service. The requested operation is interrupted if `pas-mgmt-checkconfig` fails.

Configuring PAS management to start automatically

```
systemctl enable proxedo-api-security-mgmt
```

Configuring PAS management not to start automatically

```
systemctl disable proxedo-api-security-mgmt
```

7.2.4.3. Core service

Starting PAS

```
systemctl start proxedo-api-security
```

Stopping PAS

```
systemctl stop proxedo-api-security
```

Restarting PAS

```
systemctl restart proxedo-api-security
```

Configuring PAS to start automatically

```
systemctl enable proxedo-api-security
```

Configuring PAS not to start automatically

```
systemctl disable proxedo-api-security
```

The same operations are available for the `proxedo-api-security-ha` service.

7.2.5. Operational dependencies between the core and the HA services

As the `proxedo-api-security-ha` service makes PAS highly available, the two services have a specific dependency relation. The `proxedo-api-security-ha` service can be started alone without PAS running to enable debugging without having to deal with PAS as well.

Although, if the `proxedo-api-security` service is also started, the changes of its state affect the HA service too. Stop and restart operations are propagated to the HA service and if the `proxedo-api-security` service enters failed state, it will also stop the HA service. This is to ensure renouncing *MASTER* state unless PAS is up and running.

7.2.6. Upgrading services

Prior to upgrading services, make sure that the image tags point to the right version. See section docker-compose.conf for details.

The upgrade process will cause a service disruption.

To upgrade PAS docker images, you need to run the update script of the corresponding components.

- `pas-update`

- `pas-mgmt-update`
- `pas-storage-update`

> The update scripts can be called with a `-y` or `--yes` option to automatically confirm the operation.

Major and minor version upgrades also include installing the new *.deb* package.

7.3. Checking Logs

All the container logs are collected in the system journal. Container logs are identified with the name of the container such as *pas-[transport|insight|flow|content-filtering|ha]-director*, *pas-monitoring-manager*. Management container identifiers are *pas-[frontend|config-api]*. Storage container identifiers are *pas-[consul|blob-store]*.

You can check the system journal with the `journalctl` command. It accepts various possibilities for filtering, consult its manual page for details.

> When using the `--unit` option of *journalctl*, note that the services are docker containers and their logs show up under the *docker* service, and not under *proxedo-api-security*.

> One option for checking a specific container’s logs is to use the `--identifier` option for `journalctl` and specify the identifier of the component.

7.3.1. Understanding logs

As multiple pieces of software run in each container, there are two layers of logs in each containers' output. The first field is always an ISO formatted date. Then the name of the process inside the container follows. The remaining fields are the output of the process itself. In the below example, we see logs from the `flow-director` container. It prints output for processes called `pre`, `event-handler`, `flow-director` and `service-adaptor`.

Container log output

```
2021-04-20T09:15:30 pre Container starts
2021-04-20T09:15:33 pre INFO:confgen: Generating configuration files;
service_name='twisted'
2021-04-20T09:15:34 event-handler INFO:SupervisordEventDispatcher:Dispatching event;
processname='pre', eventname='PROCESS_STATE_EXITED'
2021-04-20T09:15:34 event-handler INFO:SupervisordEventDispatcher:Process exited;
processname=pre, success=True
2021-04-20T09:15:34 event-handler INFO:SupervisordEventDispatcher:Starting main
processes.
2021-04-20T09:15:34 event-handler INFO:SupervisordEventDispatcher:Starting process;
process='flow-director'
[...]
2021-04-20T09:15:37 flow-director flow_builder.info(3) (nosession): Loaded plugin; [...]
2021-04-20T09:15:37 flow-director flow_builder.info(3) (nosession): Loaded plugin; [...]
2021-04-20T09:15:37 flow-director flow_builder.info(3) (nosession): Loaded plugin; [...]
2021-04-20T09:15:37 flow-director flow_builder.info(3) (nosession): Loaded plugin; [...]
2021-04-20T09:15:37 flow-director flow_builder.info(3) (nosession): Loaded plugin; [...]
2021-04-20T09:15:37 flow-director flow_builder.info(3) (nosession): Loaded plugin; [...]
2021-04-20T09:15:37 flow-director flow_builder.info(3) (nosession): Loaded plugin; [...]
2021-04-20T09:15:37 flow-director flow_set.info(3) (nosession): Start building flows
[...]
```

```
2021-04-20T09:15:39 event-handler INFO:SupervisordEventDispatcher:Starting process;
process='service-adaptor'
[...]
2021-05-07T14:23:55 service-adaptor INFO:HealthCheck:All services are healthy; [...]
2021-05-07T14:23:55 service-adaptor Request received; [...]
```

7.3.1.1. Flow Director and Transport Director logs

As from the API security perspective, the most important components are *Flow Director* and *Transport Director*, we discuss their logs more in detail. There are two important concepts related to these logs: categories and Session IDs.

- ***Categories*** help filtering logs based on their relevance. They are composed of a *component*, a *tag*, and a *severity*, for example: *http.info(3)*.
  - The **component** helps to identify the part of the solution. For the *Transport Director* this is usually *core* or *http*, for the *Flow Director* it is either *core*, or the *Plugin's* type, such as *serializer* or *enforcer*.
  - The **tag** helps to define the type of the message. Usually one of *info*, *error*, *debug*, *policy* or *accounting*.
  - The **severity** defines how important the message is. It is a number between 1-9 where 1 is the highest.
- ***Session ID*** helps identifying log lines that belong to the same session. This is especially important as the calls travel between the *Transport Director* and the *Flow Director*.

It is usually in the form of *svc/default/<listener>:<transport-director-session>/default/http#<http-request-count>/flow:<flow-director-id>/ch:<flow-director-channel>/<endpoint_name>/<plugin_type>/<plugin_name>*, for example: *svc/default/httpbin:14/default/http#0/flow:1/ch:28/endpoint_test/enforcer/manualtest*.

Information that is not available at the time, will be missing from the Session ID. Generally, the part until */flow:* belongs to the *Transport Director*. Consequently, the *Transport Director* will never see that part. The *Flow Director* however will fetch and include that information. Nevertheless, in early phases it might not be available, and the Session ID will start with *flow*.

Despite some parts not being always available, the ID is constructed in such a manner that grepping on any part will find other messages with extra information as well.

7.4. Disabling firewall logs from storage containers

If firewall logs from the storage containers are deemed unnecessary, they can be disabled. To do that, create a file at `/etc/sysctl.d/20-pas-disable-container-iptables-logs.conf` with the following content. Content of /etc/sysctl.d/20-pas-disable-container-iptables-logs.conf

```
net.netfilter.nf_log_all_netns=1
```

7.5. Monitoring in PAS

Monitoring data in PAS can be accessed by using either SNMPv2 or SNMPv3 protocol versions. The metrics collected with SNMPv2 and SNMPv3 in PAS form two distinctive groups:

- the PAS-related data, like container statuses and component versions
- the host-related data, like the version of the host Operating System and network statistics

The forthcoming sections introduce both the common and the distinctive configuration settings related to SNMPv2 and SNMPv3.

7.5.1. Common client configuration options with SNMPv2 and SNMPv3

The configuration of the listed parameters are common for both SNMPv2 and SNMPv3 protocol versions:

- Port: If publishing monitoring data via SNMP is enabled, any SNMP client can query metrics on the UDP port 161.
- MIB: The BALASYS-SNMP-MIB and the PAS-SNMP-MIB Management Information Base (MIB) documents can be downloaded from Balasys customer documentation. Further recommended MIB files for the analysis of this data are SNMPv2-MIB, IF-MIB and UCD-SNMP-MIB.

7.5.2. SNMPv2 client configuration options

To access the collected metrics, consider the following notes on community strings related to SNMPv2:

- PAS-related data can be accessed using any PAS community string configured under SNMPv2 authentication.
- Host-related data can be accessed using any host community string configured under SNMPv2 authentication.

7.5.3. SNMPv3 client configuration options

Note that some of the parameters for SNMPv3 are predefined and mandatory. Consider the following configuration details on the SNMPv3 parameters:

- Context name:
  - PAS-related data can be accessed using the `pas-context` string.
  - Host-related data can be accessed using the `host-context` string.
- Username: It is the username value of a user configured under SNMPv3 authentication.
- Security level: Use the predefined parameter `priv` here.
- Authentication protocol: Use the predefined parameter `SHA`.
- Authentication password: It is the authentication password value of a user, configured under SNMPv3 authentication.
- Privacy protocol: Use the predefined parameter `AES`.
- Privacy password: It is the privacy password value of a user configured under SNMPv3 authentication.

7.5.4. Example command line client usage

The following example presents querying all the available PAS-related metrics using SNMPv2:

Example output

```
snmpwalk -M <MIB file location> -m +PAS-SNMP-MIB -v2c -c <a PAS community string> <host
address>:161 1.3
```

The following example presents querying the major version of the PAS core component using SNMPv3:

Example output

```
snmpget -M <MIB file location> -m +PAS-SNMP-MIB -v3 -u <a username> -l priv -a SHA -A <an
authentication password> -x AES -X <a privacy password> -n pas-context <host address>:161
pasCoreMajorVersion
```

7.6. Backup and restore

Configuration

The following files and folders need to be backed up or restored:

- `/opt/balasys/etc`

Data

The following files or folders need to be backed up or restored:

- `/opt/balasys/var/persistent`

Process to backup files or folders

- Pack files or folders mentioned earlier (optional).
- Copy (packed) configuration and data to the backup server.

Process to restore files or folders

- Stop all PAS services.
- Copy (packed) configuration and data from the remote server.
- Unpack files or folders mentioned earlier (optional).
- Start all PAS services.

7.7. Recreating services

Recreating services will cause a service disruption.

Factory reset for PAS services

Remove all persistent data from the host.

This operation must be run as *root*.

- `/opt/balasys/bin/pas-storage-factory-reset`
- `/opt/balasys/bin/pas-factory-reset`
- `/opt/balasys/bin/pas-mgmt-factory-reset`

Resetting an individual service without removing persistent data

- Use `docker ps` to find the container name of the service, the container of which you want to reset.
- Stop the services by `systemctl stop proxedo-api-security` or `systemctl stop proxedo-api-security-mgmt` or `systemctl stop proxedo-api-security-storage`.
- Remove the containers by `docker rm <name-of-container>`.
- Start the services by `systemctl start proxedo-api-security` or `systemctl start proxedo-api-security-mgmt` or `systemctl start proxedo-api-security-storage`.

7.8. Troubleshooting docker services

The troubleshooting procedures are applicable to the following docker services:

PAS component	Docker service
Storage component	• consul • blob-store

PAS component	Docker service
Management component	• config-api • frontend
Core component	• content-filtering-director • flow-director • insight-director • transport-director • monitoring-manager

7.8.1. Inspect running processes inside docker services

- `docker compose -f /opt/balasys/etc/infrastructure/pas/docker-compose.yml top <Docker service>`
- `docker compose -f /opt/balasys/etc/infrastructure/mgmt/docker-compose.yml top <Docker service>`
- `docker compose -f /opt/balasys/etc/infrastructure/storage/docker-compose.yml top <Docker service>`
- `docker compose -f /opt/balasys/etc/infrastructure/ha/docker-compose.yml top <Docker service>`

You can list available docker services by running `docker compose -f <docker-compose-file> ps <Docker service>`.

Example output

```
pas_flow-director_1
UID     PID    PPID    C   STIME   TTY     TIME
CMD
-------------------------------------------------------------------------------------------
-----------------------------------------
root   26109   26052   0   13:46   ?     00:00:00   /usr/bin/dumb-init
/usr/local/bin/supervisord -c /opt/balasys/etc/supervisord.conf
[...]
root   26529   26252   0   13:46   ?     00:00:01   /usr/bin/python3
/usr/local/bin/twistd -ny /opt/balasys/etc/twisted.tac
[...]

pas_flow-director_2
UID     PID    PPID    C   STIME   TTY     TIME
CMD
-------------------------------------------------------------------------------------------
-----------------------------------------
root   26350   26314   0   13:46   ?     00:00:00   /usr/bin/dumb-init
/usr/local/bin/supervisord -c /opt/balasys/etc/supervisord.conf
[...]
root   26545   26434   0   13:46   ?     00:00:01   /usr/bin/python3
/usr/local/bin/twistd -ny /opt/balasys/etc/twisted.tac
[...]
```

7.8.2. Inspect files inside docker services

To find out what files are available with what content in docker services, use the appropriate `pas-login`

command. This command provides an interactive shell in the selected container in which file inspection and editing tools are available.

The list of login commands are the following:

- `pas-login <Docker service>`
- `pas-mgmt-login <Docker service>`
- `pas-storage-login <Docker service>`

Run these commands without parameters to get the list of available docker service names.

Example usage of the `pas-login` command

```
$ pas-login flow-director
root@b2c55b0b91aa:/# ls -l /opt/balasys/etc/version
-rw-r--r-- 1 balasys balasys 31 Aug 13 10:32 /opt/balasys/etc/version
```

7.8.3. Inspect process state and network traffic inside docker containers

To inspect process states and network traffic, use the appropriate `pas-network-and-process-debug-login` command. This will start a debug container which has access to the processes and network traffic of the target container.

The list of login commands are the following:

- `pas-network-and-process-debug-login <Docker container>`
- `pas-mgmt-network-and-process-debug-login <Docker container>`
- `pas-storage-network-and-process-debug-login <Docker container>`

Run these commands without parameters to get the list of available docker container names.

Compared to the `pas-login` commands, these ones work with docker containers, not services. This is to enable inspecting different instances of *Flow Director*.

Example usage of the `pas-network-and-process-debug-login`

```
$ pas-network-and-process-debug-login pas_flow-director_1
root@f64d5a4c421c:/# ps x
    PID TTY      STAT   TIME COMMAND
      1 ?        Ss     0:00 /usr/bin/dumb-init [...]
      6 ?        Ss     0:01 /usr/bin/python3 [...]
      7 ?        S      0:00 /usr/bin/python3 [...]
      8 ?        Sl     0:00 /usr/sbin/syslog-ng [...]
     28 ?        Sl     0:01 /usr/bin/python3 [...]
     29 ?        S      0:00 uwsgi [...]
     30 ?        Sl     0:00 uwsgi [...]
    539 pts/0    Ss     0:00 /bin/bash
    597 pts/0    R+     0:00 ps x
```

Appendix A: config.yml examples

A.1. Minimal storage configuration

The configuration example is set as follows:

- Standalone server is used, not joining to a cluster
- Default TLS settings are used for storage-storage configuration
- Certificates and encryption key are generated by pas-storage-consul-* commands
- INFO log level is defined

Example /opt/balasys/etc/storage/config.yml

```
common:
  short_product_name: pas
  standalone_mode: true

consul:
  bind_cluster_addr: 192.168.1.220
  gossip_encryption_key: lzT4l6mS4O7lj9Y9KeJYcABpn9q5GczbsoMG7fRuAfE=
  node_name: mgmt

blob_store:
  access_key: your_access_key
  secret_key: your_secret_key
  rpc_secret: your_rpc_secret
  admin_token: your_admin_token
```

A.2. Minimal management configuration

The configuration example is set as follows:

- Only HTTP access is set for the web interface, no TLS is used
- htpasswd authentication is used, no LDAP is set

Example /opt/balasys/etc/mgmt/config.yml

```
frontend: {}

configapi: {}
```

A.3. Management configuration with HTTPS (TLS) and LDAP authentication

The configuration example is set as follows:

- A certificate for the web service must be generated and copied to the management node beforehand.
- LDAP authentication is configured without TLS.
- The authentication configuration was tested using Microsoft Active Directory.

Example /opt/balasys/etc/mgmt/config.yml with NTLM on

```
frontend:
  tls:
    certificate_path: '/opt/balasys/etc/mgmt/pas.example.com.crt'
    key_path: '/opt/balasys/etc/mgmt/pas.example.com.key'

configapi:
  ldap:
    ldap_url: ldap://ad.example.com
    use_ntlm: on
    bind_user: AD_domain\administrator  # The name of the user follows the domain.
    bind_password: your_administrator_password
    user_base_dn: CN=Users,DC=example,DC=com
    group_base_dn: CN=Users,CN=Builtin,DC=example,DC=com
    allowed_groups:
      - Users
```

Example /opt/balasys/etc/mgmt/config.yml with NTLM off

```
frontend:
  tls:
    certificate_path: '/opt/balasys/etc/mgmt/pas.example.com.crt'
    key_path: '/opt/balasys/etc/mgmt/pas.example.com.key'

configapi:
  ldap:
    ldap_url: ldap://ad.example.com
    use_ntlm: off
    bind_user: CN=administrator,CN=Users,DC=example,DC=com  # This must be the DN of the
user
    bind_password: your_administrator_password
    user_base_dn: CN=Users,DC=example,DC=com
    group_base_dn: CN=Users,CN=Builtin,DC=example,DC=com
    allowed_groups:
      - Users
```

A.4. Minimal HA configuration

The configuration example is set as follows:

- HA node with the highest priority (other node must have priority less than 200)

Example /opt/balasys/etc/ha/config.yml

```
ha:
  interface: eth0
  priority: 250
  auth_pass: your_ha_password
  virtual_ip: 192.168.1.254
```

Appendix B: LDAP certificate examples

Single CA file example

```
-----BEGIN CERTIFICATE-----
```

```
... (the certificate for the CA)...
-----END CERTIFICATE-----
```

Example on certificate chain with multiple CAs

```
-----BEGIN CERTIFICATE-----
... (the certificate for the CA)...
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
... (the root certificate for the CA's issuer)...
-----END CERTIFICATE-----
```

Appendix C: Selector configuration for the Fraud Detector Plugin

The following fields can be defined in the *Save As Key* field when creating a new *Selector*. The saved *Selector* can be used by the *Fraud Detector* plugin.

The data type selected in the API for the actual selector option shall be the one listed in this table as *Type* for the actual selector. Currently, no data type conversion is possible for selectors.

Table 73. Selector configuration for the Fraud Detector Plugin

Values for *Save As Key* field	Data type	Description	Example
action_type	string	The type of the user action being scored. Any string can be valid.	`update_content`, `verification` or `account_login_fail`
client_address	string	The user's IP address at the time of the transaction. It shall include the full IPv4 or IPv6 address.	
transaction_id	string	A unique identifier for the transaction, as found in the system. If it is not specified, it is automatically generated.	`98db9a56b2e3`
affiliate_id	string	The user's unique affiliate identifier in the system.	
affiliate_name	string	The name of the affiliate for the registered user. Can be ASCII-encoded via a secure hash algorithm, such as MD5 or SHA-2.	`jdoe345`
order_memo	string	The description of the transaction found in the system.	
email	string	The full email address of the registered user.	
email_domain	string	The email address domain of the registered user.	
password_hash	string	The hash of the user's password in ASCII encoding (we recommend using HMAC-SHA256 or RSA-SHA256).	

Values for *Save As Key* field	Data type	Description	Example
user_fullname	string	The user's registered full name. Can be hashed in ASCII encoding as well (e.g. MD5, SHA-2 family).	`John Doe`
user_name	string	The user's registered username. Can be hashed in ASCII encoding as well (e.g. MD5, SHA-2 family).	`jdoe325`
user_id	string	The user's unique identifier. If the request was sent without a user_id value, a unique ID is automatically generated based on the user_name and/or the email fields, based on which is available. If none of these identifiers were included in the request, the user ID is generated randomly.	`00ab11-as2233`
user_created	integer	The date when the user first registered to the protected service, using the UNIX time format and UTC time zone, without milliseconds.	`1446370717` (Sun, 01 Nov 2015 09:38:37 +0000)
user_category	string	The user's category.	`VIP`
user_account_status	string	The user's current account status.	`login_blocked`
user_bank_account	string	The user's bank account number for monetary transfer.	IBAN number
user_bank_name	string	The name of the user's bank account.	
user_balance	float	The user's current balance.	`1010.25`
user_verification_level	string	The user's verification level.	`ID_verified`
user_dob	date	The user's date of birth in the format of YYYY-MM-DD.	`1983-01-01`
user_country	string	The country code for the user's registered address. Uses the two-character ISO 3166-1 format.	`US`, `DE`
user_city	string	The complete name of the city associated with the user's registered address.	`London`, `New York`
user_region	string	The state or region code for the user's registered address. Uses the two-character ISO 3166-2 format.	`NY`, `DE`
user_zip	string	The zip/postal code of the user's registered address.	`10005`, `PH1 1EU`
user_street	string	The first line of the user's registered street address. Can be hashed in ASCII encoding as well.	MD5, SHA-2 family: `157 W 26th St`
user_street2	string	The second line of the user's registered street address. Can be hashed in ASCII encoding as well.	MD5, SHA-2 family: `Apt.432`

Values for *Save As Key* field	Data type	Description	Example
session_id	string	The session ID is a custom, unique ID that links the user's device data with the transactions. It shall be based on the user's current browsing session, by tracking cookies for example. If JavaScript Agent v4 is used, the encrypted payload returned by the SDK (supported by JS Agent v4, iOS SDK 3.0.0, Android SDK 3.0.0) shall be sent in the session field, instead of the session_id.	
session	string	The base64 encoded session data returned by the SDKs.	
device_id	string	This field shall only be used if a device fingerprinting solution is used already. This is the ID that shall be linked to the transactions or in case rules are required to be built on those IDs.	
payment_mode	string	The method of payment used.	`card`, `paypal`, `wire transfer`, `bitcoin`
payment_provider	string	The name of the payment service provider related to the transaction.	`skrill`
card_fullname	string	The user's full name found on the card. Can be hashed in ASCII encoding as well.	MD5, SHA-2 family
card_bin	string	The first 4, 6 or 8 digits of the card number.	
card_hash	string	The hash of the credit card used by the user in ASCII encoding. We recommend using HMAC-SHA256 or RSA-SHA256 formats and strictly advise not to use MD5 hash format.	
card_expire	string	The card's expiration date.	`2022-01`
card_last	string	The last 4 digits of the card number. These help to identify the card.	
avs_result	string	The standard Address verification Service (AVS) codes sent by the credit card processor.	`N`, `A`
cvv_result	boolean	The Cad Verification Value (CVV) result.	`true`, `false`
status_3d	string	The Cad Verification Value (CVV) result.	`true`, `false`
sca_method	string	The result of the Strong Customer Authentication method.	`2FA`
phone_number	string	The user's registered phone number, including the country code. Cannot include spaces or hyphens, the + sign is optional. The maximum length is 19 characters.	`36704316088`
transaction_type	string	The transaction type of the actual business.	`purchase`, `return`
transaction_amount	float	The full transaction amount. As a decimal point use '.' (full stop).	`539.99`
transaction_currency	string	The currency used by the user, in ISO 4217 format. Crypto currencies are also supported.	`EUR`, `BTC`, `USDT`

Values for *Save As Key* field	Data type	Description	Example
shipping_country	string	A two-character ISO 3166-1 country code for the country associated with the user's shipping address.	`US`, `DE`
shipping_city	string	The full name of the city associated with the user's shipping address.	`London`, `New York`
shipping_region	string	The state or region code for the user's shipping address. Uses the two-character ISO 3166-2 format	`NY`, `DE`
shipping_zip	string	The zip/postal code of the user's shipping address.	`10005`, `PH1 1EU`
shipping_street	string	The first line of the user's shipping street address. Can be hashed in ASCII encoding as well (e.g. MD5, SHA-2 family).	`157 W 26th St`
shipping_street2	string	The second line of the user's shipping street address. Can be hashed in ASCII encoding as well (e.g. MD5, SHA-2 family).	`Apt.432`
shipping_phone	string	The phone number associated with the user's shipping address, including the country code. Cannot include spaces or hyphens, the + sign is optional. The maximum length is 19 characters.	`36704316088`
shipping_fullname	string	The user's registered full name. Can be hashed in ASCII encoding as well (e.g. MD5, SHA-2 family).	`John Doe`
shipping_method	string	The type of the shipping method used by the customer.	`standard`, `UPS`, `FedEx`
billing_country	string	The country code for the user's billing address. Uses the two-character ISO 3166-1 format.	`US`, `DE`
billing_city	string	The full name of the city associated with the user's billing address.	`London`, `New York`
billing_region	string	The state or region code for the user's billing address. Uses the two-character ISO 3166-2 format	`NY`, `DE`
billing_zip	string	The zip/postal code of the user's billing address.	`10005`, `PH1 1EU`
billing_street	string	The user's billing street address line 1. Can be hashed in ASCII encoding as well (e.g. MD5, SHA-2 family).	`157 W 26th St`
billing_street2	string	The user's billing street address line 2. Can be hashed in ASCII encoding as well (e.g. MD5, SHA-2 family).	`Apt.432`
billing_phone	string	The phone number associated with the user's billing address, including the country code. Cannot include spaces or hyphens, the + sign is optional. The maximum length is 19 characters.	`36704316088`
discount_code	string	The discount code that the user applied during the checkout.	

Values for *Save As Key* field	Data type	Description	Example
gift	boolean	The user can mark the order with true or false value, dependent on if it is a gift or not.	
gift_message	boolean	The user can mark the order with true or false value, dependent on if the order has a gift message or not.	
merchant_category	string	The category of the merchant.	`digital_item_seller`
merchant_id	string	The unique merchant identifier in case the orders are from different merchants.	`ab01-cd23-4567`
merchant_created_at	integer	The date the merchant was created, using the UNIX time format and UTC time zone.	`1446370717` (Sun, 01 Nov 2015 09:38:37 +0000)
merchant_country	string	The country code for the merchant's address. Uses the two-character ISO 3166-1 format.	`US`, `DE`
receiver_fullname	string	The receiver's full name for monetary transfer.	IBAN number
details_url	string	The URL of the transaction in the management platform.	
regulation	string	The license or market name for gambling operator.	`MGA`
bonus_campaign_id	string	The bonus campaign's unique identifier.	`bonus100a`
brand_id	string	The brand's unique identifier.	`brand123`

The maximum length of all request parameters is 100 characters, except for the following: **500 characters for card_hash** 64 characters for the session_id (sent directly or within the session field) **19 characters for the phone_number** 15 characters for card_bin **4 characters for transaction_currency** 50 characters for discount_code and shipping_method ** 255 characters for transaction_id

Appendix D: Time zones

Country Code	Time zone Name
AD	Europe/Andorra
AE	Asia/Dubai
AF	Asia/Kabul
AG	America/Antigua
AI	America/Anguilla
AL	Europe/Tirane
AM	Asia/Yerevan
AO	Africa/Luanda

Country Code	Time zone Name
AQ	Antarctica/McMurdo
AQ	Antarctica/Casey
AQ	Antarctica/Davis
AQ	Antarctica/DumontDUrville
AQ	Antarctica/Mawson
AQ	Antarctica/Palmer
AQ	Antarctica/Rothera
AQ	Antarctica/Syowa
AQ	Antarctica/Troll
AQ	Antarctica/Vostok
AR	America/Argentina/Buenos_Aires
AR	America/Argentina/Cordoba
AR	America/Argentina/Salta
AR	America/Argentina/Jujuy
AR	America/Argentina/Tucuman
AR	America/Argentina/Catamarca
AR	America/Argentina/La_Rioja
AR	America/Argentina/San_Juan
AR	America/Argentina/Mendoza
AR	America/Argentina/San_Luis
AR	America/Argentina/Rio_Gallegos
AR	America/Argentina/Ushuaia
AS	Pacific/Pago_Pago
AT	Europe/Vienna
AU	Australia/Lord_Howe
AU	Antarctica/Macquarie
AU	Australia/Hobart
AU	Australia/Currie
AU	Australia/Melbourne
AU	Australia/Sydney
AU	Australia/Broken_Hill
AU	Australia/Brisbane
AU	Australia/Lindeman
AU	Australia/Adelaide

Country Code	Time zone Name
AU	Australia/Darwin
AU	Australia/Perth
AU	Australia/Eucla
AW	America/Aruba
AX	Europe/Mariehamn
AZ	Asia/Baku
BA	Europe/Sarajevo
BB	America/Barbados
BD	Asia/Dhaka
BE	Europe/Brussels
BF	Africa/Ouagadougou
BG	Europe/Sofia
BH	Asia/Bahrain
BI	Africa/Bujumbura
BJ	Africa/Porto-Novo
BL	America/St_Barthelemy
BM	Atlantic/Bermuda
BN	Asia/Brunei
BO	America/La_Paz
BQ	America/Kralendijk
BR	America/Noronha
BR	America/Belem
BR	America/Fortaleza
BR	America/Recife
BR	America/Araguaina
BR	America/Maceio
BR	America/Bahia
BR	America/Sao_Paulo
BR	America/Campo_Grande
BR	America/Cuiaba
BR	America/Santarem
BR	America/Porto_Velho
BR	America/Boa_Vista
BR	America/Manaus

Country Code	Time zone Name
BR	America/Eirunepe
BR	America/Rio_Branco
BS	America/Nassau
BT	Asia/Thimphu
BW	Africa/Gaborone
BY	Europe/Minsk
BZ	America/Belize
CA	America/St_Johns
CA	America/Halifax
CA	America/Glace_Bay
CA	America/Moncton
CA	America/Goose_Bay
CA	America/Blanc-Sablon
CA	America/Toronto
CA	America/Nipigon
CA	America/Thunder_Bay
CA	America/Iqaluit
CA	America/Pangnirtung
CA	America/Atikokan
CA	America/Winnipeg
CA	America/Rainy_River
CA	America/Resolute
CA	America/Rankin_Inlet
CA	America/Regina
CA	America/Swift_Current
CA	America/Edmonton
CA	America/Cambridge_Bay
CA	America/Yellowknife
CA	America/Inuvik
CA	America/Creston
CA	America/Dawson_Creek
CA	America/Fort_Nelson
CA	America/Vancouver
CA	America/Whitehorse

Country Code	Time zone Name
CA	America/Dawson
CC	Indian/Cocos
CD	Africa/Kinshasa
CD	Africa/Lubumbashi
CF	Africa/Bangui
CG	Africa/Brazzaville
CH	Europe/Zurich
CI	Africa/Abidjan
CK	Pacific/Rarotonga
CL	America/Santiago
CL	America/Punta_Arenas
CL	Pacific/Easter
CM	Africa/Douala
CN	Asia/Shanghai
CN	Asia/Urumqi
CO	America/Bogota
CR	America/Costa_Rica
CU	America/Havana
CV	Atlantic/Cape_Verde
CW	America/Curacao
CX	Indian/Christmas
CY	Asia/Nicosia
CY	Asia/Famagusta
CZ	Europe/Prague
DE	Europe/Berlin
DE	Europe/Busingen
DJ	Africa/Djibouti
DK	Europe/Copenhagen
DM	America/Dominica
DO	America/Santo_Domingo
DZ	Africa/Algiers
EC	America/Guayaquil
EC	Pacific/Galapagos
EE	Europe/Tallinn

Country Code	Time zone Name
EG	Africa/Cairo
EH	Africa/El_Aaiun
ER	Africa/Asmara
ES	Europe/Madrid
ES	Africa/Ceuta
ES	Atlantic/Canary
ET	Africa/Addis_Ababa
FI	Europe/Helsinki
FJ	Pacific/Fiji
FK	Atlantic/Stanley
FM	Pacific/Chuuk
FM	Pacific/Pohnpei
FM	Pacific/Kosrae
FO	Atlantic/Faroe
FR	Europe/Paris
GA	Africa/Libreville
GB	Europe/London
GD	America/Grenada
GE	Asia/Tbilisi
GF	America/Cayenne
GG	Europe/Guernsey
GH	Africa/Accra
GI	Europe/Gibraltar
GL	America/Godthab
GL	America/Danmarkshavn
GL	America/Scoresbysund
GL	America/Thule
GM	Africa/Banjul
GN	Africa/Conakry
GP	America/Guadeloupe
GQ	Africa/Malabo
GR	Europe/Athens
GS	Atlantic/South_Georgia
GT	America/Guatemala

Country Code	Time zone Name
GU	Pacific/Guam
GW	Africa/Bissau
GY	America/Guyana
HK	Asia/Hong_Kong
HN	America/Tegucigalpa
HR	Europe/Zagreb
HT	America/Port-au-Prince
HU	Europe/Budapest
ID	Asia/Jakarta
ID	Asia/Pontianak
ID	Asia/Makassar
ID	Asia/Jayapura
IE	Europe/Dublin
IL	Asia/Jerusalem
IM	Europe/Isle_of_Man
IN	Asia/Kolkata
IO	Indian/Chagos
IQ	Asia/Baghdad
IR	Asia/Tehran
IS	Atlantic/Reykjavik
IT	Europe/Rome
JE	Europe/Jersey
JM	America/Jamaica
JO	Asia/Amman
JP	Asia/Tokyo
KE	Africa/Nairobi
KG	Asia/Bishkek
KH	Asia/Phnom_Penh
KI	Pacific/Tarawa
KI	Pacific/Enderbury
KI	Pacific/Kiritimati
KM	Indian/Comoro
KN	America/St_Kitts
KP	Asia/Pyongyang

Country Code	Time zone Name
KR	Asia/Seoul
KW	Asia/Kuwait
KY	America/Cayman
KZ	Asia/Almaty
KZ	Asia/Qyzylorda
KZ	Asia/Qostanay
KZ	Asia/Aqtobe
KZ	Asia/Aqtau
KZ	Asia/Atyrau
KZ	Asia/Oral
LA	Asia/Vientiane
LB	Asia/Beirut
LC	America/St_Lucia
LI	Europe/Vaduz
LK	Asia/Colombo
LR	Africa/Monrovia
LS	Africa/Maseru
LT	Europe/Vilnius
LU	Europe/Luxembourg
LV	Europe/Riga
LY	Africa/Tripoli
MA	Africa/Casablanca
MC	Europe/Monaco
MD	Europe/Chisinau
ME	Europe/Podgorica
MF	America/Marigot
MG	Indian/Antananarivo
MH	Pacific/Majuro
MH	Pacific/Kwajalein
MK	Europe/Skopje
ML	Africa/Bamako
MM	Asia/Yangon
MN	Asia/Ulaanbaatar
MN	Asia/Hovd

Country Code	Time zone Name
MN	Asia/Choibalsan
MO	Asia/Macau
MP	Pacific/Saipan
MQ	America/Martinique
MR	Africa/Nouakchott
MS	America/Montserrat
MT	Europe/Malta
MU	Indian/Mauritius
MV	Indian/Maldives
MW	Africa/Blantyre
MX	America/Mexico_City
MX	America/Cancun
MX	America/Merida
MX	America/Monterrey
MX	America/Matamoros
MX	America/Mazatlan
MX	America/Chihuahua
MX	America/Ojinaga
MX	America/Hermosillo
MX	America/Tijuana
MX	America/Bahia_Banderas
MY	Asia/Kuala_Lumpur
MY	Asia/Kuching
MZ	Africa/Maputo
NA	Africa/Windhoek
NC	Pacific/Noumea
NE	Africa/Niamey
NF	Pacific/Norfolk
NG	Africa/Lagos
NI	America/Managua
NL	Europe/Amsterdam
NO	Europe/Oslo
NP	Asia/Kathmandu
NR	Pacific/Nauru

Country Code	Time zone Name
NU	Pacific/Niue
NZ	Pacific/Auckland
NZ	Pacific/Chatham
OM	Asia/Muscat
PA	America/Panama
PE	America/Lima
PF	Pacific/Tahiti
PF	Pacific/Marquesas
PF	Pacific/Gambier
PG	Pacific/Port_Moresby
PG	Pacific/Bougainville
PH	Asia/Manila
PK	Asia/Karachi
PL	Europe/Warsaw
PM	America/Miquelon
PN	Pacific/Pitcairn
PR	America/Puerto_Rico
PS	Asia/Gaza
PS	Asia/Hebron
PT	Europe/Lisbon
PT	Atlantic/Madeira
PT	Atlantic/Azores
PW	Pacific/Palau
PY	America/Asuncion
QA	Asia/Qatar
RE	Indian/Reunion
RO	Europe/Bucharest
RS	Europe/Belgrade
RU	Europe/Kaliningrad
RU	Europe/Moscow
UA	Europe/Simferopol
RU	Europe/Kirov
RU	Europe/Astrakhan
RU	Europe/Volgograd

Country Code	Time zone Name
RU	Europe/Saratov
RU	Europe/Ulyanovsk
RU	Europe/Samara
RU	Asia/Yekaterinburg
RU	Asia/Omsk
RU	Asia/Novosibirsk
RU	Asia/Barnaul
RU	Asia/Tomsk
RU	Asia/Novokuznetsk
RU	Asia/Krasnoyarsk
RU	Asia/Irkutsk
RU	Asia/Chita
RU	Asia/Yakutsk
RU	Asia/Khandyga
RU	Asia/Vladivostok
RU	Asia/Ust-Nera
RU	Asia/Magadan
RU	Asia/Sakhalin
RU	Asia/Srednekolymsk
RU	Asia/Kamchatka
RU	Asia/Anadyr
RW	Africa/Kigali
SA	Asia/Riyadh
SB	Pacific/Guadalcanal
SC	Indian/Mahe
SD	Africa/Khartoum
SE	Europe/Stockholm
SG	Asia/Singapore
SH	Atlantic/St_Helena
SI	Europe/Ljubljana
SJ	Arctic/Longyearbyen
SK	Europe/Bratislava
SL	Africa/Freetown
SM	Europe/San_Marino

Country Code	Time zone Name
SN	Africa/Dakar
SO	Africa/Mogadishu
SR	America/Paramaribo
SS	Africa/Juba
ST	Africa/Sao_Tome
SV	America/El_Salvador
SX	America/Lower_Princes
SY	Asia/Damascus
SZ	Africa/Mbabane
TC	America/Grand_Turk
TD	Africa/Ndjamena
TF	Indian/Kerguelen
TG	Africa/Lome
TH	Asia/Bangkok
TJ	Asia/Dushanbe
TK	Pacific/Fakaofo
TL	Asia/Dili
TM	Asia/Ashgabat
TN	Africa/Tunis
TO	Pacific/Tongatapu
TR	Europe/Istanbul
TT	America/Port_of_Spain
TV	Pacific/Funafuti
TW	Asia/Taipei
TZ	Africa/Dar_es_Salaam
UA	Europe/Kiev
UA	Europe/Uzhgorod
UA	Europe/Zaporozhye
UG	Africa/Kampala
UM	Pacific/Midway
UM	Pacific/Wake
US	America/New_York
US	America/Detroit
US	America/Kentucky/Louisville

Country Code	Time zone Name
US	America/Kentucky/Monticello
US	America/Indiana/Indianapolis
US	America/Indiana/Vincennes
US	America/Indiana/Winamac
US	America/Indiana/Marengo
US	America/Indiana/Petersburg
US	America/Indiana/Vevay
US	America/Chicago
US	America/Indiana/Tell_City
US	America/Indiana/Knox
US	America/Menominee
US	America/North_Dakota/Center
US	America/North_Dakota/New_Salem
US	America/North_Dakota/Beulah
US	America/Denver
US	America/Boise
US	America/Phoenix
US	America/Los_Angeles
US	America/Anchorage
US	America/Juneau
US	America/Sitka
US	America/Metlakatla
US	America/Yakutat
US	America/Nome
US	America/Adak
US	Pacific/Honolulu
UY	America/Montevideo
UZ	Asia/Samarkand
UZ	Asia/Tashkent
VA	Europe/Vatican
VC	America/St_Vincent
VE	America/Caracas
VG	America/Tortola
VI	America/St_Thomas

Country Code	Time zone Name
VN	Asia/Ho_Chi_Minh
VU	Pacific/Efate
WF	Pacific/Wallis
WS	Pacific/Apia
YE	Asia/Aden
YT	Indian/Mayotte
ZA	Africa/Johannesburg
ZM	Africa/Lusaka
ZW	Africa/Harare

Glossary

API	Application Programming Interface
CA	Certification Authority
CRL	Certificate Revocation List
HTTP	HyperText Transport Protocol
HTTPS	HyperText Transport Protocol Secure
JSON	JavaScript Object Notation
LDAP	Lightweight Directory Access Protocol
MIB	Management Information Base
NTLM	NT LAN Manager
PEM	Privacy Enhanced Mail
SNI	Server Name Indication
SNMP	Simple Network Management Protocol
SOAP	Simple Object Access Protocol
SSL	Secure Socket Layer
SIEM	Security Information and Event Management
TLS	Transport Layer Security
URI	Universal Resource Indicator
URL	Universal Resource Locator
WSDL	Web Service Definition Language
XML	Extensible Markup Language

XSD	XML Schema Definition