PROXEDO

API SECURITY

Proxedo API Security in Kubernetes

Administration Guide

DISCLAIMER

Balasys is not responsible for any third-party websites mentioned in this document. Balasys does not endorse and is not responsible or liable for any content, advertising, products, or other material on or available from such sites or resources. Balasys will not be responsible or liable for any damage or loss caused or alleged to be caused by or in connection with use of or reliance on any such content, goods, or services that are available on or through any such sites or resources.

2022-08-29

Table of Contents

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.

ℹ 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.

In the parameter listing tables the required parameters are also emphasized with bold text:

Key	Description
param1	This is a required parameter.
param2	This is an optional parameter.

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.

Key	Description
+	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 the Web User Interface for the Proxedo API Security in Kubernetes. The purpose of this document is to present the designed approach and the usage for the configuration of Proxedo API Security via Web User Interface (UI). The Web UI allows easy configuration for Proxedo API Security. All the functionalities are grouped visually and logically into thematic units which follow the logical built up of Proxedo API Security's configuration. 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
- 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 in Kubernetes environment if you need to set up a new PAS.
- The Operation of Proxedo API Security in Kubernetes environment 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 enpoint'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. 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.

3.1.4. 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.

3.2. 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*.

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

It is 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

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

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.

Serializer

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

Filter

It is a *Plugin* that rejects calls when they match defined rules.

Enforcer

It is a *Plugin* that validates calls against externally defined schemas.

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).

Brick

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

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*.

3.3. Architecture for Proxedo API Security

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

The components of the architecture are each responsible for well-defined subset of handling traffic between the client and the backend. Proxedo API Security is built up of three components:

Transport Director

It 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

It is responsible for the execution of the *Plugins* in the *Endpoint's* flow and for applying *Error Policies* as necessary.

Insight Director

It manages the connections to *Insight Targets*. It is responsible for sending the data collected by *Insight plugins* to *Insight Target* systems.

The handling of a connection with the help of components is shown in this figure:

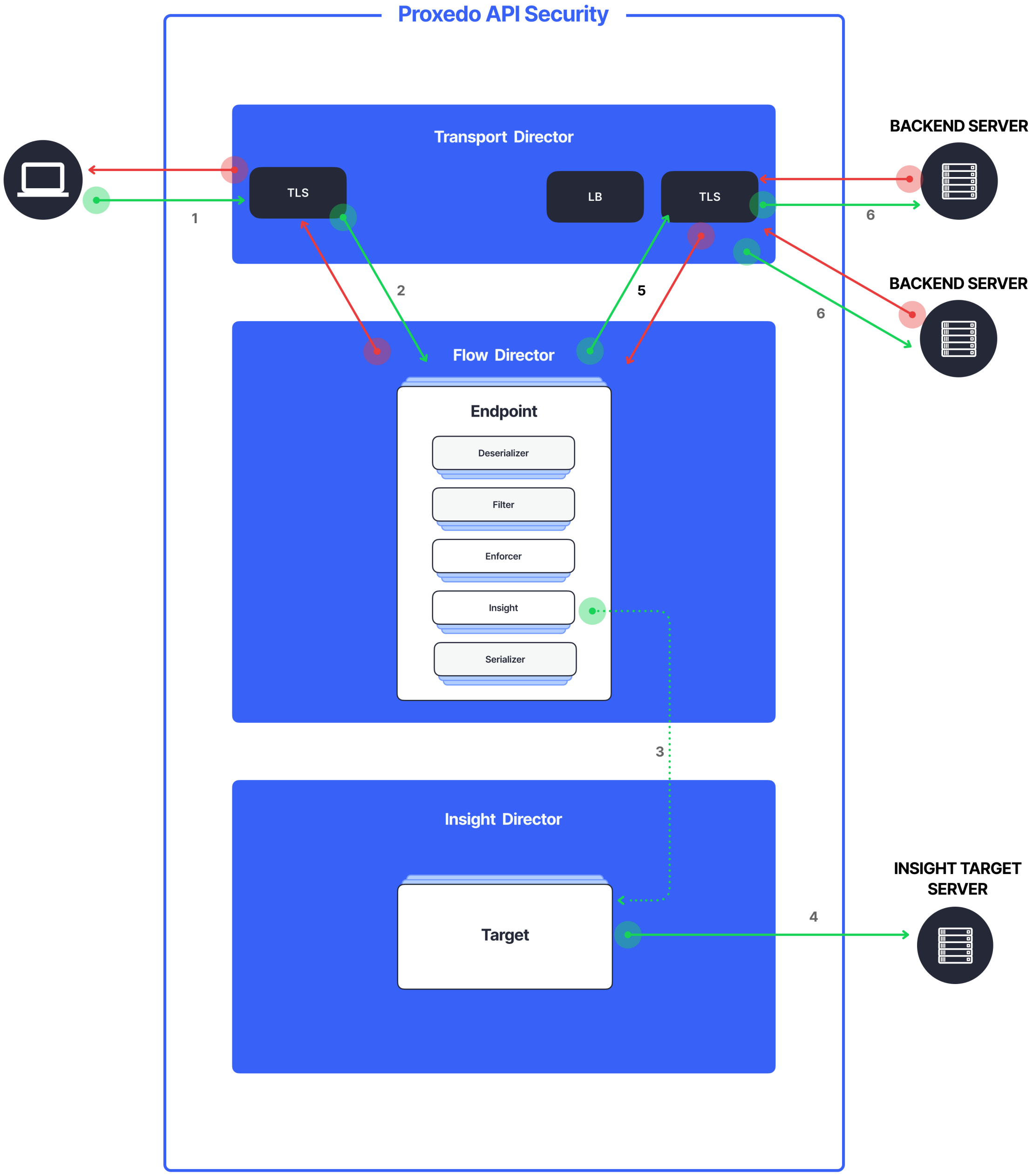


Figure 1. PAS Architecture

1. Incoming connections are accepted by the *Transport Director*.
    - It handles TLS with the client if necessary.
2. It hands over the connection to the *Flow Director*.
    - The *Flow Director* chooses the *Endpoint* based on the URL.
    - 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. The *Flow Director* hands the connection back to the *Transport Director*.
6. 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*.

3.3.1. 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:

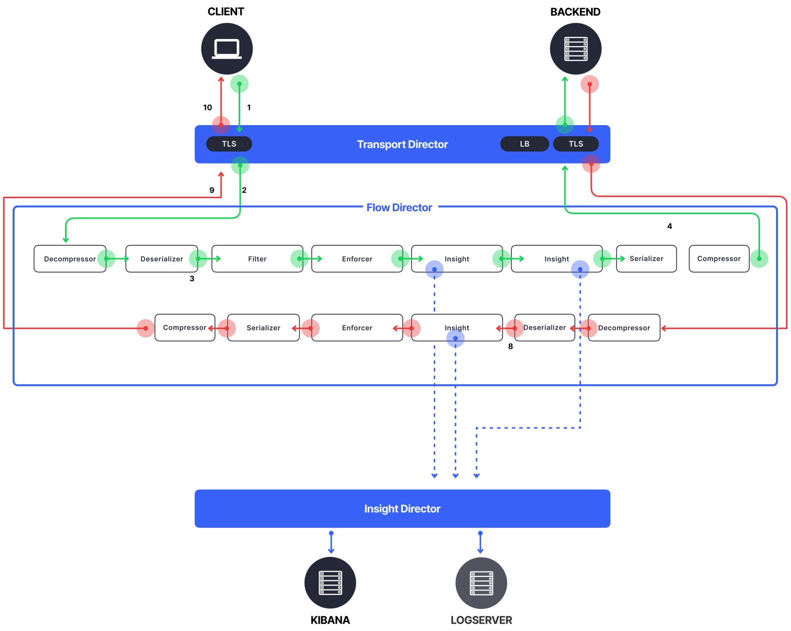


Figure 2. PAS processing flow

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* hands over the connection to the *Flow Director*, indicating which *Listener* the connection belongs to.
3. The *Flow Director* then chooses the *Endpoint* based on the URL in the request. First endpoint has matching URL is chosen.
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.

4. Installation of Proxedo API Security in Kubernetes environment

The forthcoming sections describe the installation of PAS in Kubernetes.

> To manage Kubenetes (K8s) applications, Helm, the package manager for Kubernetes is used. Packages are called *charts* in the *Helm* context.

4.1. Prerequisites for installing PAS

The followings are needed prior to the installation of PAS:

- the license file for PAS
- a technical user for accessing Balasys' download site
- the Helm chart

> Prior to the installation of the *Helm chart*, the *Helm chart* itself must be configured. For minimum configuration of the *Helm chart* see section Minimum configuration settings for the Helm chart.

4.1.1. Cluster components necessary for PAS

To make use of some of the features, PAS shall be deployed in a cluster, with the following components installed:

- metrics server for auto-scaling
- *Persistent volume* for storing configuration in the management component

> *Persistent Volume Claim* parameters can be set up to match a manually managed *Persistent volume*, so is *Storage Class* name.

- access for the target namespace to deploy PAS in

4.1.2. Tools necessary for the installation

To create the basic configuration for the installation, the following tools are necessary:

- openssl for storage certificate generation
- the htpasswd tool, which is part of the apache2-utils package on debian distributions, the httpd-tools package on Red Hat based distributions
- the helm command line tool to manage the package installation
- the kubectl command line tool to communicate with the Kubernetes cluster

4.1.3. Minimum configuration settings for the Helm chart

The *Helm* chart contains the following:

- configuration parameters to bootstrap PAS in K8s
- definitions of
  - pods
  - services
  - autoscaling configuration for the core component
  - a *Persistent Volume Claim* for the management

Ingress configuration for any component is not included.

HTTP and HTTPS management access is recommended to be configured using an Ingress (kubernetes object).

In order to be able to install the *Helm chart* the minimum configuration settings have to be completed. The following sections contain the details only for the necessary minimum configuration, however for checking further possible configuration options, see section Base system configuration for PAS in Kubernetes.

The files detailed in the next sections need to be created and filled in prior to PAS installation.

4.1.3.1. Using values.yml file

1. Use the values.yml (values file) with the default and necessary values. Run the following command to output the configuration options:

```
helm show values /path/to/chart/proxedo-api-security-4.0.1.tgz
```

2. Create a local values.yml file with the preferred values to overwrite the default values if required. The values file with minimum configuration is as follows (with example values):

```
config:
  storage:
    consul:
      gossip_encryption_key: MhstT80sqle63WC7knOak+c7GfK7k5OY2n/4Qk/fSXs=
    blob_store:
      access_key: "8i8YJB3JhFmkT5KK6EV5EGw9dK10B4ZllWjEYlvUwKM="
      secret_key: "L/aLsKkoDFDFnMNdp8MFl1/CIkAQC1hrXV+HlbgKyOM="
```

3. Generate these necessary secrets with the help of the following command. The values above are examples, they shall not be copied directly.

```
# config.consul.gossip_encryption_key
$ openssl rand -base64 32
gI97yg2Zcq4XL20ne8NBwH2e0PbzkmXjqMFdp8jQZac=

# consig.blob_store.access_key
$ openssl rand -base64 32
+WDpoDV7EcJrgkRgK65M3y8OcLdrZmYBASVTFE1I8pg=

# config.blob_store.secret_key
$ openssl rand -base64 32
ECuGiOwyJtjlB8Bl3yNgIgdk/nlb4HFmxE/4oiq5V+w=
```

4.1.3.2. Creating certificates for storage

For technical reasons, a TLS certificate is necessary for configuration storage purposes. Create the internal CAs and signed certificates either with a preferred method, or else the necessary files can be created with the following example commands as well.

1. Generate a CA key pair.

> ℹ The *-days* parameter in the example commands define the validity period of the generated certificates in days. Change it, if it is required.

> ❗ The certificate files generated here and used with the *Helm chart* are sensitive pieces of information, therefore handle those with attention.

```
openssl req -nodes -new -x509 -days +3650 -keyout storage-ca-key.pem -out storage-ca.pem
-subj "/CN=PAS Storage CA"
```

2. Generate a private server key and a Certificate Signing Request (CSR).

```
openssl req -nodes -new -keyout consul-0-key.pem -out consul-0.csr -days +3650 -subj
"/CN=storage.pas"
```

3. Sign the CSR using the CA.

```
openssl x509 -req -days +3650 -in consul-0.csr -CA storage-ca.pem -CAkey storage-ca-
key.pem -CAcreateserial  -out consul-0.pem
```

With the help of the above examples, further files need to be generated. These files will need to be provided for the *Helm chart*:

- consul-0.csr
- consul-0-key.pem
- consul-0.pem
- storage-ca-key.pem
- storage-ca.pem

4.1.3.3. Creating management users' file

For logging into the management component, the `users.htpass` file is required. Run the following command to generate one, and provide the password.

```
htpasswd -c users.htpass username
```

4.2. Installing PAS in Kubernetes

The following sections and the example commands use the `proxedo-api-security` kubernetes namespace as an example, but it can be replaced with any other namespace name.

To create a new namespace, run the following command:

```
kubectl create namespace proxedo-api-security
```

4.2.1. Setting up docker registry connection

1. Log in to the PAS docker registry to access the docker images of PAS.
2. Create the `proxedo-api-security-registry-credentials` secret using the following command to enable kubernetes to access the docker images:

```
kubectl create --namespace proxedo-api-security \
    secret docker-registry proxedo-api-security-registry-credentials \
    --docker-server=docker.balasys.hu \
    --docker-username=<<your username>> \
    --docker-password="$(read -sp "Docker registry password: " DOCKER_PASSWORD; echo
$DOCKER_PASSWORD)"
```

4.2.2. Providing the necessary files for *Helm* installation

Provide the created files for the *Helm* install command, an example of which can be seen below (substitute your values):

```
helm upgrade --install proxedo-api-security --namespace=<<namespace>> \
    --values /path/to/config/files/values.yml \
    --set-file license=/path/to/config/files/license.txt \
    --set-file mgmt_users=/path/to/config/files/users.htpass \
    --set-file storage_ca_key=/path/to/config/files/storage-ca-key.pem \
    --set-file storage_ca_cert=/path/to/config/files/storage-ca.pem \
    --set-file storage_server_key=/path/to/config/files/consul-0-key.pem \
    --set-file storage_server_cert=/path/to/config/files/consul-0.pem \
    /path/to/chart/proxedo-api-security-4.0.1.tgz
```

4.3. Verifying the installation of PAS in Kubernetes

If everything is correct, the Helm command will present the following output:

```
NAME: proxedo-api-security
LAST DEPLOYED: Mon May  2 13:51:46 2022
NAMESPACE: proxedo-api-security
STATUS: deployed
REVISION: 1
TEST SUITE: None
```

1. Run the `kubectl get pods --selector=app=proxedo-api-security` command to investigate the running pods. The output shall be similar to the following example:

```
NAME                                                       READY   STATUS
RESTARTS   AGE
proxedo-api-security-blob-store-86ccc6d864-frc5k           1/1     Running             0
40s
proxedo-api-security-config-api-76d587d6cd-wpw5d           1/1     Running             0
40s
proxedo-api-security-consul-68c5c87f75-mvlct               1/1     Running             0
40s
proxedo-api-security-flow-director-5cddf58677-qxczd        0/1     ContainerCreating   0
40s
proxedo-api-security-frontend-676bfd8956-qrtm4             1/1     Running             0
40s
proxedo-api-security-insight-director-585cc5f86-j8rrz      0/1     ContainerCreating   0
40s
proxedo-api-security-transport-director-5bbdf58d7d-whzsq   0/1     ContainerCreating   0
40s
```

The core pod is missing the core configuration, therefore it will not enter the "Running" state until the first configuration is applied in the management.

2. Run the following command to access the management component for verification.

```
kubectl port-forward service/proxedo-api-security-frontend 8080:80
```

3. Open the http://127.0.0.1:8080/ in the browser.

5. Base system configuration for PAS in Kubernetes

This chapter explains configuration details for setting up a working PAS. Configuration settings are detailed here, which are based on the installation of the *Helm chart*.

The *Helm chart* carries Kubernetes manifest files for each component, and requires a set of parameters to be configured by the user for the installation.

The values.yml file

The configuration of PAS components is condensed into a `values.yml` file. The default version of this file can be printed by using the following command:

`helm show values /path/to/chart/proxedo-api-security-4.0.1.tgz`

To configure the necessary parameters and to overwrite the not suitable default values, save the output to a file, and keep only those parts that has to be overwritten. The modified file can be provided as `--values my-values.yml` to the Helm installation command.

There are two main sections of this file:

1. Infrastructure - This section defines the options necessary for kubernetes to deploy the components.
2. Configuration - This section defines the options for PAS itself. The main configuration of the storage and management components is defined in this file.

The format of this file must adhere to the YAML 1.1 specification.

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

For information on how to provide the custom `values.yml` file, see section Providing the necessary files for *Helm* installation. See configuration examples in Appendix B.

5.1. Infrastructure configuration

In this infrastructure part of the configuration, many parameter fields are directly associated with the configuration attributes defined for the Kubernetes objects. For such parameters that have a Kubernetes equivalent, the Kubernetes parameter is referenced in the format that can directly be used with the `kubectl explain` command. This command provides the most specific documentation of each field. However, for using this command, access to a cluster is required.

In case it is not feasible to use the `kubectl explain` command, the referenced format can also be used to navigate to the correct object and field at the following site: Kubernetes API.

The following tables describe the infrastructure parameters and their Kubernetes equivalent if that exists.

Table 1. Docker-related parameters

Parameter field	Default value	Description
infrastructure.docker.registry	docker.balasys.hu	The registry to download docker images from.
infrastructure.docker.pull_policy	IfNotPresent	This parameter has a Kubernetes equivalent in all pods: *pod.spec.containers*.
infrastructure.docker.image_tag		The image tag to use instead of the one corresponding to the current PAS version.

Table 2. Storage-related infrastructure parameters

Parameter field	Default value	Description
infrastructure.storage.volume_claim		This parameter has a Kubernetes equivalent: *PersistentVolumeClaim*.
infrastructure.storage.storage_class_name		This parameter has a Kubernetes equivalent: *PersistentVolumeClaim.spec.storageClassName*.
infrastructure.storage.access_modes	ReadWrite Once	This parameter has a Kubernetes equivalent: *PersistentVolumeClaim.spec.accessModes*.
infrastructure.storage.requests		This parameter has a Kubernetes equivalent: *PersistentVolumeClaim.spec.resources.requests*.
infrastructure.storage.requests.storage	100Mi	This parameter has a Kubernetes equivalent: *PersistentVolumeClaim.spec.resources.requests.storage*.

Table 3. Transport Director infrastructure parameters

Parameter field	Default value	Description
Service		
infrastructure.core.transport_director.service		This parameter has a Kubernetes equivalent: *service*.
infrastructure.core.transport_director.service.type	ClusterIP	This parameter has a Kubernetes equivalent: *service.spec.type*.
infrastructure.core.transport_director.service.ports		This parameter has a Kubernetes equivalent: *service.spec.ports*. A port with a specific *target_port* value needs to be set up for each listener port in the PAS configuration on the management interface.
infrastructure.core.transport_director.service.ports.name	HTTP	This parameter has a Kubernetes equivalent: *service.spec.ports.name*.
infrastructure.core.transport_director.service.ports.port	80	This parameter has a Kubernetes equivalent: *service.spec.ports.port*.
infrastructure.core.transport_director.service.ports.protocol	TCP	This parameter has a Kubernetes equivalent: *service.spec.ports.protocol*.
infrastructure.core.transport_director.service.ports.target_port	49 000	This parameter has a Kubernetes equivalent: *service.spec.ports.targetPort*.
infrastructure.core.transport_director.service.ports.node_port		This parameter has a Kubernetes equivalent: *service.spec.ports.nodePort*.
Resources		
infrastructure.core.transport_director.resources		This parameter has a Kubernetes equivalent: *pod.spec.containers.resources*.
infrastructure.core.transport_director.resources.limits		This parameter has a Kubernetes equivalent: *pod.spec.containers.resources.limits*. If this is defined, both CPU and memory limits need to be defined.
infrastructure.core.transport_director.resources.limits.cpu		This parameter has a Kubernetes equivalent: *pod.spec.containers.resources.limits.cpu*.
infrastructure.core.transport_director.resources.limits.memory		This parameter has a Kubernetes equivalent: *pod.spec.containers.resources.limits.memory*.
infrastructure.core.transport_director.resources.requests		This parameter has a Kubernetes equivalent: *pod.spec.containers.resources.requests*.
infrastructure.core.transport_director.resources.requests.cpu	250 m	This parameter has a Kubernetes equivalent: *pod.spec.containers.resources.requests.cpu*.
infrastructure.core.transport_director.resources.requests.memory	450 Mi	This parameter has a Kubernetes equivalent: *pod.spec.containers.resources.requests.memory*.
Scaling		
infrastructure.core.transport_director.scaling		For scaling parameters, see the separate table on scaling, Parameters for Scaling - Transport Director, Flow Director, Insight Director.

Table 4. Flow Director infrastructure parameters

Parameter field	Default value	Description
Resources		
infrastructure.core.flow_director.resources		This parameter has a Kubernetes equivalent: *pod.spec.containers.resources*.
infrastructure.core.flow_director.resources.limits		This parameter has a Kubernetes equivalent: *pod.spec.containers.resources.limits*. If this is defined, both CPU and memory limits need to be defined.
infrastructure.core.flow_director.resources.limits.cpu		This parameter has a Kubernetes equivalent: *pod.spec.containers.resources.limits.cpu*.
infrastructure.core.flow_director.resources.limits.memory		This parameter has a Kubernetes equivalent: *pod.spec.containers.resources.limits.memory*.
infrastructure.core.flow_director.resources.requests		This parameter has a Kubernetes equivalent: *pod.spec.containers.resources.requests*.
infrastructure.core.flow_director.resources.requests.cpu	250 m	This parameter has a Kubernetes equivalent: *pod.spec.containers.resources.requests.cpu*.
infrastructure.core.flow_director.resources.requests.memory	550 Mi	This parameter has a Kubernetes equivalent: *pod.spec.containers.resources.requests.memory*.
Scaling		
infrastructure.core.flow_director.scaling		For scaling parameters, see the separate table on scaling, Parameters for Scaling - Transport Director, Flow Director, Insight Director.

Table 5. Insight Director infrastructure parameters

Parameter field	Default value	Description
Resources		
infrastructure.core.insight_director.resources		This parameter has a Kubernetes equivalent: *pod.spec.containers.resources*.
infrastructure.core.insight_director.resources.limits		This parameter has a Kubernetes equivalent: *pod.spec.containers.resources.limits*. If this is defined, both CPU and memory limits need to be defined.
infrastructure.core.insight_director.resources.limits.cpu		This parameter has a Kubernetes equivalent: *pod.spec.containers.resources.limits.cpu*.
infrastructure.core.insight_director.resources.limits.memory		This parameter has a Kubernetes equivalent:_pod.spec.containers.resources.limits.memory_.
infrastructure.core.insight_director.resources.requests		This parameter has a Kubernetes equivalent: *pod.spec.containers.resources.requests*.
infrastructure.core.insight_director.resources.requests.cpu	120 m	This parameter has a Kubernetes equivalent: *pod.spec.containers.resources.requests.cpu*.
infrastructure.core.insight_director.resources.requests.memory	350 Mi	This parameter has a Kubernetes equivalent: *pod.spec.containers.resources.requests.memory*.

Parameter field	Default value	Description
Scaling		
infrastructure.core.insight_director.scaling		For scaling parameters, see the separate table on scaling, Parameters for Scaling - Transport Director, Flow Director, Insight Director.

Table 6. Parameters for Scaling - Transport Director, Flow Director, Insight Director

Parameter field	Default value	Description
infrastructure.core.<transport/flow/insight>_director. scaling		This parameter has a Kubernetes equivalent: *HorizontalPodAutoscaler*.
infrastructure.core.<transport/flow/insight>_director. scaling.create_autoscaler	true	This parameter defines whether to create the HoizontalPodAutoscaler object with the forthcoming configuration options. If it is set to false, the HPA object to enable core autoscaling will need to be created manually.
infrastructure.core.<transport/flow/insight>_director. scaling.min_replicas	1	This parameter has a Kubernetes equivalent: *horizontalpodautoscaler.spec.minReplicas*.
infrastructure.core.<transport/flow/insight>_director. scaling.max_replicas	10	This parameter has a Kubernetes equivalent: *horizontalpodautoscaler.spec.maxReplicas*.
infrastructure.core.<transport/flow/insight>_director. scaling.metrics		This parameter has a Kubernetes equivalent: *horizontalpodautoscaler.spec.metrics*.
infrastructure.core.<transport/flow/insight>_director. scaling.metrics.cpu		This parameter defines the CPU metric configuration.
infrastructure.core.<transport/flow/insight>_director. scaling.metrics.cpu.average_utilization	80	This parameter has a Kubernetes equivalent: *horizontalpodautoscaler.spec.metrics.resource. target.averageUtilization*.
infrastructure.core.<transport/flow/insight>_director. scaling.metrics.memory		This parameter defines the memory metric configuration.
infrastructure.core.<transport/flow/insight>_director. scaling.metrics.memory.average_utilization	80	This parameter has a Kubernetes equivalent: *horizontalpodautoscaler.spec.metrics.resource. target.averageUtilization*.
infrastructure.core.<transport/flow/insight>_director. scaling.metrics.behavior		This parameter has a Kubernetes equivalent: *horizontalpodautoscaler.spec.behavior*. If it is defined, either *scale_down* or *scale_up* parameter must be defined.
infrastructure.core.<transport/flow/insight>_director. scaling.metrics.behavior.scale_down		This parameter has a Kubernetes equivalent: *horizontalpodautoscaler.spec.behavior.scaleDown*. If it is defined, all included parameters need to be defined.
infrastructure.core.<transport/flow/insight>_director. scaling.metrics.behavior.scale_down.stabilization_window_seconds		This parameter has a Kubernetes equivalent: *horizontalpodautoscaler.spec.behavior.scaleDown. stabilizationWindowSeconds*.
infrastructure.core.<transport/flow/insight>_director. scaling.metrics.behavior.scale_down.policies		This parameter has a Kubernetes equivalent: *horizontalpodautoscaler.spec.behavior.scaleDown. policies*.

Parameter field	Default value	Description
infrastructure.core.<transport/flow/insight>_director. scaling.metrics.behavior.scale_down.policies.type		This parameter has a Kubernetes equivalent: *horizontalpodautoscaler.spec.behavior.scaleDown. policies.type*.
infrastructure.core.<transport/flow/insight>_director. scaling.metrics.behavior.scale_down.policies.value		This parameter has a Kubernetes equivalent: *horizontalpodautoscaler.spec.behavior.scaleDown. policies.value*.
infrastructure.core.<transport/flow/insight>_director. scaling.metrics.behavior.scale_down.policies.period_seconds		This parameter has a Kubernetes equivalent: *horizontalpodautoscaler.spec.behavior.scaleDown. policies.periodSeconds*.
infrastructure.core.<transport/flow/insight>_director. scaling.metrics.behavior.scale_down.select_policy		This parameter has a Kubernetes equivalent: *horizontalpodautoscaler.spec.behavior.scaleDown. selectPolicy*.
infrastructure.core.<transport/flow/insight>_director. scaling.metrics.behavior.scale_up		This parameter has a Kubernetes equivalent: *horizontalpodautoscaler.spec.behavior.scaleUp*. If it is defined, all included parameters need to be defined.
infrastructure.core.<transport/flow/insight>_director. scaling.metrics.behavior.scale_up.stabilization_window_seconds		This parameter has a Kubernetes equivalent: *horizontalpodautoscaler.spec.behavior.scaleUp. stabilizationWindowSeconds*.
infrastructure.core.<transport/flow/insight>_director. scaling.metrics.behavior.scale_up.policies		This parameter has a Kubernetes equivalent: *horizontalpodautoscaler.spec.behavior.scaleUp. policies*.
infrastructure.core.<transport/flow/insight>_director. scaling.metrics.behavior.scale_up.policies.type		This parameter has a Kubernetes equivalent: *horizontalpodautoscaler.spec.behavior.scaleUp. policies.type*.
infrastructure.core.<transport/flow/insight>_director. scaling.metrics.behavior.scale_up.policies.value		This parameter has a Kubernetes equivalent: *horizontalpodautoscaler.spec.behavior.scaleUp. policies.value*.
infrastructure.core.<transport/flow/insight>_director. scaling.metrics.behavior.scale_up.policies.period_seconds		This parameter has a Kubernetes equivalent: *horizontalpodautoscaler.spec.behavior.scaleUp. policies.periodSeconds*.
infrastructure.core.<transport/flow/insight>_director. scaling.metrics.behavior.scale_up.select_policy		This parameter has a Kubernetes equivalent: *horizontalpodautoscaler.spec.behavior.scaleUp. selectPolicy*.

5.2. PAS configuration in Kubernetes

5.2.1. Configuration options for the storage component

The `config.storage` section controls keys to be used between the management and storage components.

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 7. Storage configuration **common** *options*

Key	Default	Description
config.storage.common.standalone_mode	true	This parameter must be set to 'true'. It denotes whether the storage is run in standalone or in cluster mode. If it is set to true, the cluster-related parameters are ignored. The required parameters still need to be provided.

Table 8. Storage configuration **consul** *options*

Key	Default	Description
config.storage.consul.bind_cluster_addr	127.0.0.1	It denotes the address to bind on as a cluster member. This will be used to communicate with other members. **This is a required paramater.**
config.storage.consul.gossip_encryption_key		This parameter denotes the encryption key to use for the gossip protocol. It is a 32-byte shared key encoded into base64 format. Use `openssl rand -base64 32` to generate it. For more information on the keys produced as part of the configuration, see Using values.yml file. **This is a required paramater.**
config.storage.consul.log_level	INFO	It denotes the log level of consul. The possible values are: TRACE, DEBUG, INFO, WARN, ERR

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

Table 9. Storage configuration **blob-store** *options*

Key	Default	Description
config.storage.blob_store.access_key		It denotes the access key used for connecting to MinIO. A preferably random generated string must be provided. Min length: 3 **This is a required parameter.**
config.storage.blob_store.secret_key		It denotes the secret key used for connecting to MinIO. A preferably random generated string must be provided. Min length: 8. **This is a required parameter.**

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 configuration.

5.2.2. Configuration options for the management component

The `config.mgmt` section controls:

- Web service parameters
- Authentication

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 10. Management configuration **frontend** *options*

Key	Default	Description
config.mgmt.frontend.server_name	`_`	It is 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.
config.mgmt.frontend.cors_api	N/A	This section configures cross origin request sharing options for API access.
config.mgmt.frontend.allow_origin		It denotes 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 11. Management configuration log level setting options - **configapi** *section*

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

Table 12. Management configuration user session options - **configapi** *section*

Key	Default	Description
config.mgmt.configapi.session	N/A	This section configures the options for session lifetimes.
config.mgmt.configapi.session.session_validity	7200	It denotes 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.**
config.mgmt.configapi.session.renew_validity	36000	It denotes 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 configuration and section Management configuration with 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

It is required to provide the *htpass* file already for the *Helm chart* installation. See section Providing the necessary files for *Helm* 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 created and provided for the *Helm* installation command. 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. If you want to add new users to the *htpasswd* file, run the `htpasswd users.htpass username` command and provide the password.

Example command and output

```
$ htpasswd 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:

- the *htpasswd* file is created and provided for the *Helm* installation command
- new-user is the name of the new user

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

`new-user:$apr1$GDRF00xV$DmqFFfl.O5GWFpDjQl6tJ`.

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.

ℹ 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 13. Management configuration LDAP authentication options - **configapi** *section*

Key	Default	Description
config.mgmt.configapi.ldap	N/A	This section configures the options for LDAP authentication. **LDAP authentication is disabled by default.**
config.mgmt.configapi.ldap.ldap_url		It is the URL of the LDAP server. It must start with `ldap[s]://`. **This is a required parameter in case of LDAP authentication.**
config.mgmt.configapi.ldap.bind_user		It denotes the service user to use, for searching the LDAP server. If `use_ntlm` parameter is OFF, this must be the whole DN. If it is ON, it must be the username as expected by the service. **This is a required parameter in case of LDAP authentication.**
config.mgmt.configapi.ldap.bind_password		It denotes the password of the service user. **This is a required parameter in case of LDAP authentication.**
config.mgmt.configapi.ldap.use_ntlm	OFF	Set this parameter to ON to use NTLM authentication.
config.mgmt.configapi.ldap.tls_version	TLSv1_2	It denotes 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.
config.mgmt.configapi.ldap.validate_cert	no	Set it to yes to validate certificates.

Key	Default	Description
config.mgmt.configapi.ldap.ca_certs_file	/opt/balasys/etc/ldap_ca_certs.pem	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 exmaple 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/3.6/library/ssl.html#certificates. Use the `--set-file mgmt_ldap_ca_certs_file=<path/to/file>` command during helm installation to specify this file. Also uncomment the `ca_certs_file` parameter without changing its value.
config.mgmt.configapi.ldap.user_base_dn		It is the base DN under which users reside. **This is a required parameter in case of LDAP authentication.**
config.mgmt.configapi.ldap.username_attribute	sAMAccountName	It is the attribute that contains the name of the user.
config.mgmt.configapi.ldap.user_object_class	user	It is the object class of the users.
config.mgmt.configapi.ldap.memberof_attribute	memberof	It is the attribute that contains membership information (groups) on user objects.
config.mgmt.configapi.ldap.group_base_dn		It is the base DN under which groups reside. **This is a required parameter in case of LDAP authentication.**
config.mgmt.configapi.ldap.groupname_attribute	name	It is the attribute that contains the name of the group.
config.mgmt.configapi.ldap.member_attribute	member	It is the attribute that contains membership information (users) on group objects.

Key	Default	Description
config.mgmt.configapi.ldap.group_object_class	group	It is the object class for groups.
config.mgmt.configapi.ldap.allowed_groups		It is a list of group names (as contained by 'groupname_attribute') allowed to log in. **This is a required parameter in case of LDAP authentication.**

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 found in the journal under the `pas-config-api` identifier.

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

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:

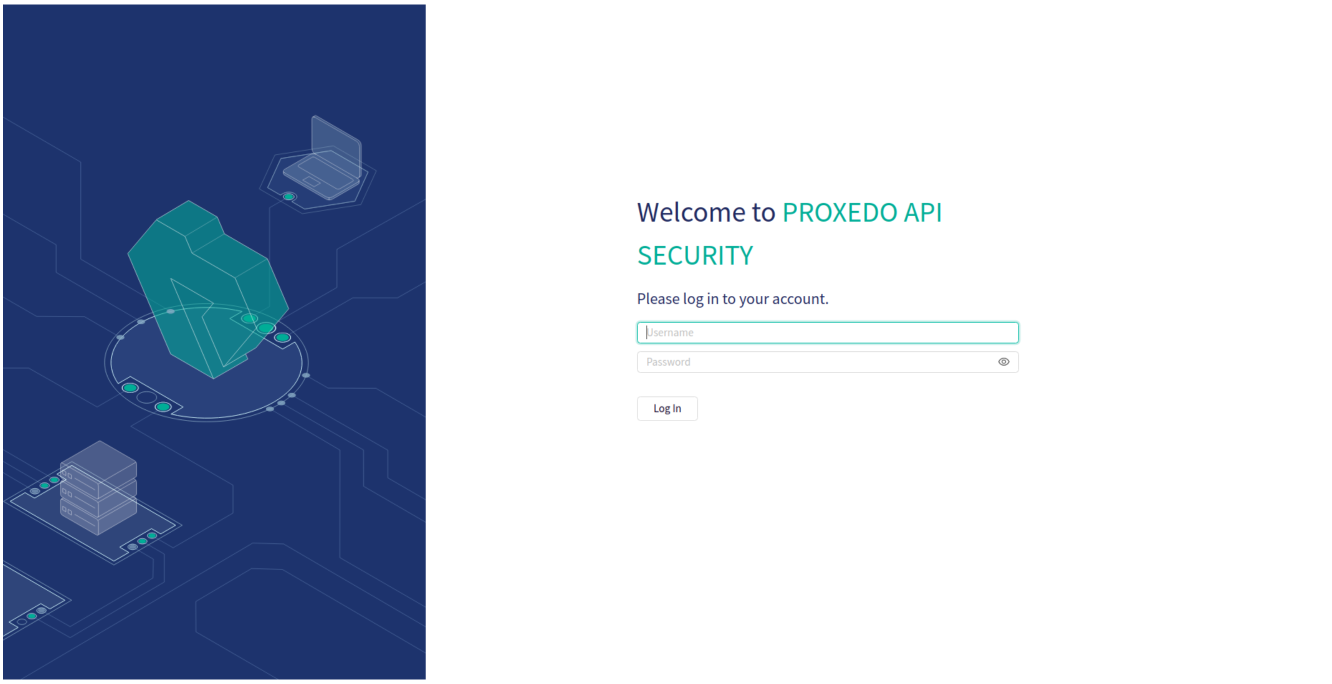
- Listeners
  - Port
  - Endpoint
    For more details on the *Listener*'s parameters, see Listeners' configuration options.
- Endpoint
  - Name
  - Url
    For more details on the *Endpoint*'s parameters, see Endpoint configuration.
- Security Flow
  - Request
  - Response
  - Backend

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.



Welcome to PROXEDO API SECURITY

Please log in to your account.

Figure 3. Login page for Proxedo API Security Web User Interface

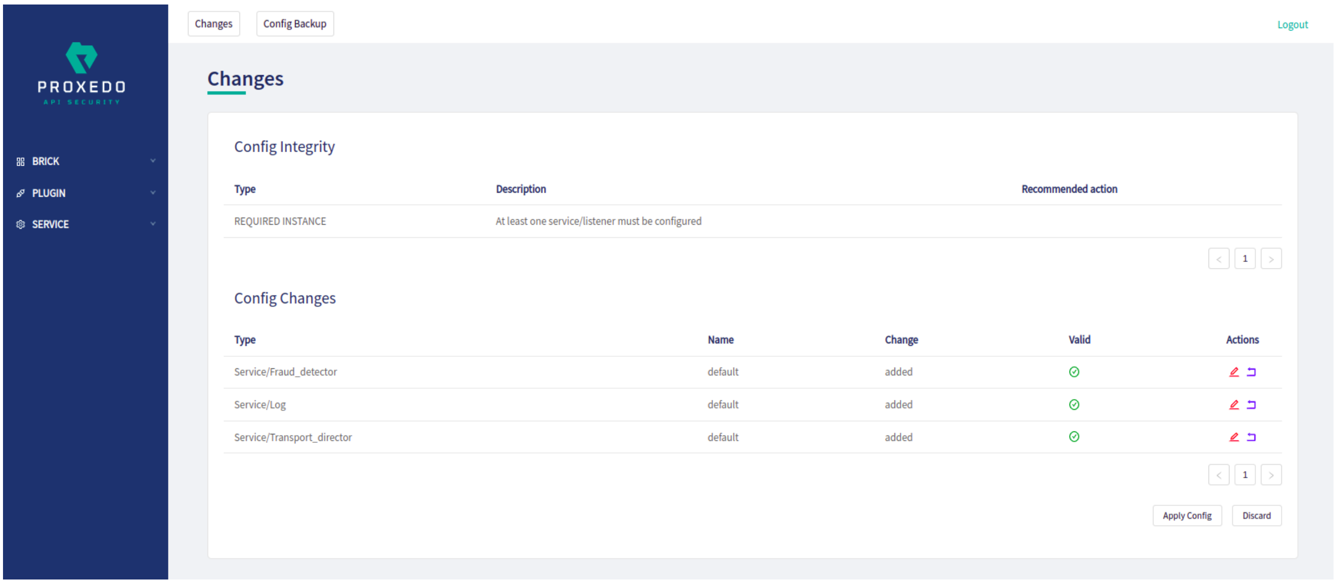
For accessing the Web User Interface:

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.



Changes

Config Integrity

Type	Description	Recommended action
REQUIRED INSTANCE	At least one service/listener must be configured	

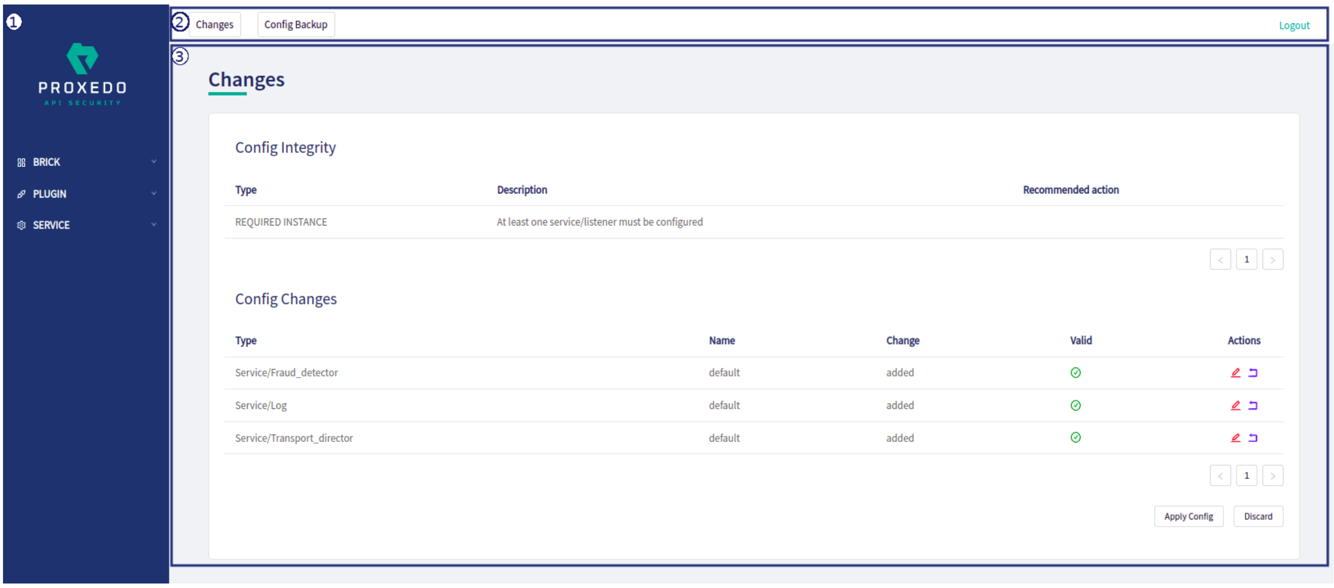
Config Changes

Type	Name	Change	Valid	Actions
Service/Fraud_detector	default	added	✓	
Service/Log	default	added	✓	
Service/Transport_director	default	added	✓	

Figure 4. Proxedo API Security Web User Interface main page

6.3.1. Navigation

The PAS Web UI has the following navigation areas:



Changes

Config Integrity

Type	Description	Recommended action
REQUIRED INSTANCE	At least one service/listener must be configured	

Config Changes

Type	Name	Change	Valid	Actions
Service/Fraud_detector	default	added	✓	
Service/Log	default	added	✓	
Service/Transport_director	default	added	✓	

Figure 5. 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, three main navigation units are available, that is, BRICK, PLUGIN, and SERVICE.
These three 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* and the *Config backup* buttons in the top left corner and the *Logout* button in the top right corner. For more information on these services, see Checking and finalizing changes in Proxedo API Security configuration and Backup and restore services for Proxedo API Security configuration.

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
New	By selecting the *New* navigation button, you can configure a new component, previously selected from the Left navigation panel (1) for configuration.

Navigation option	Description
✎	By selecting the *Pen* navigation button, the Web UI navigates back to the configuration page of the selected element. You can change the so far configured details or add new configuration details.
🗑	By selecting the *Bin* button, you can delete the configuration element active in the window. If you select an element 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.
< 1 >	By selecting the *Next page* button you can 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. BRICK - 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:

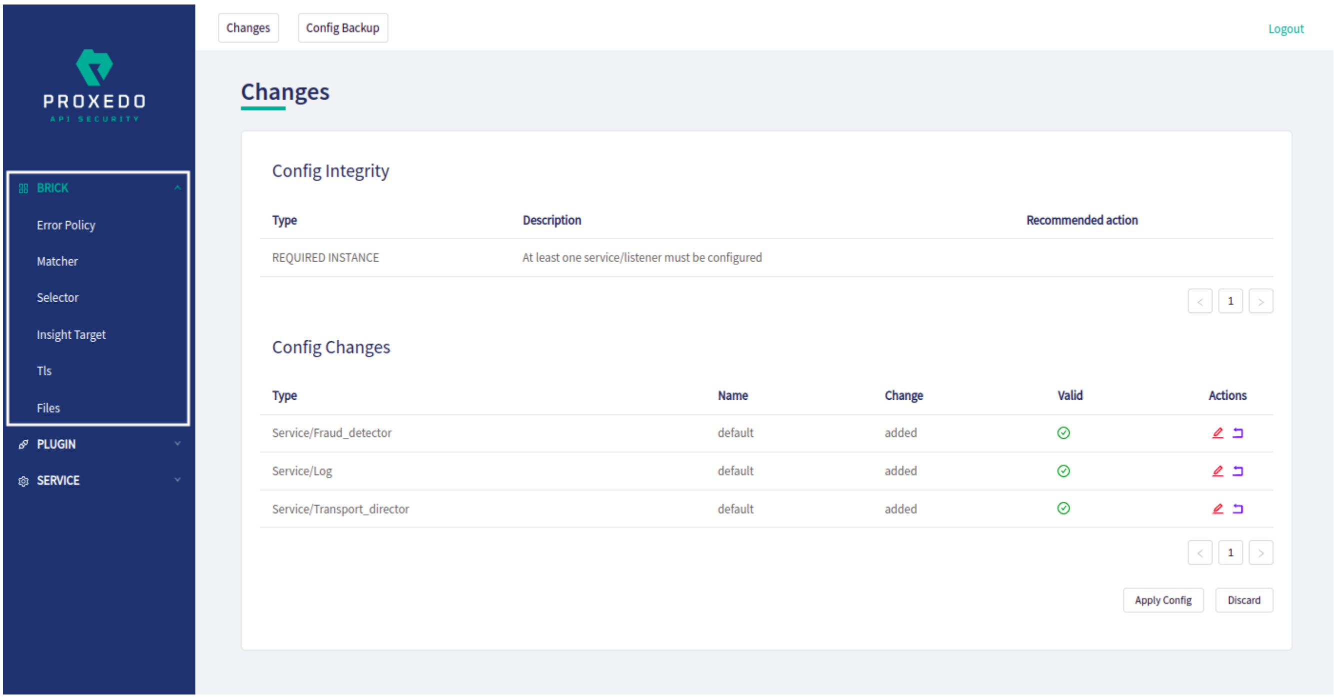
Table 15. Default objects - BRICK

Default object name	Class
Always	Matcher
Never	Matcher
content_type_json	Matcher
content_type_json_regexp	Matcher
json_content	Matcher
content_type_xml_base	Matcher
content_type_xml_dtd	Matcher
content_type_xml_ext_parsed	Matcher
content_type_xml_regexp	Matcher
content_type_xml_text	Matcher

Default object name	Class
content_type_xml_text_ext_parsed	Matcher
xml_content	Matcher
error_policy	Error policy
enforcer_default	Error policy
insight_default	Error policy

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

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



Config Integrity

Type	Description	Recommended action
REQUIRED INSTANCE	At least one service/listener must be configured	

Config Changes

Type	Name	Change	Valid	Actions
Service/Fraud_detector	default	added	✓	
Service/Log	default	added	✓	
Service/Transport_director	default	added	✓	

Figure 6. The Brick's main page in the Web User Interface

1. Click on the *BRICK* main configuration item in the Left navigation area. Alternatively you can also click on the ⌄ sign to open up the sub-navigation items of *BRICK*.
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.

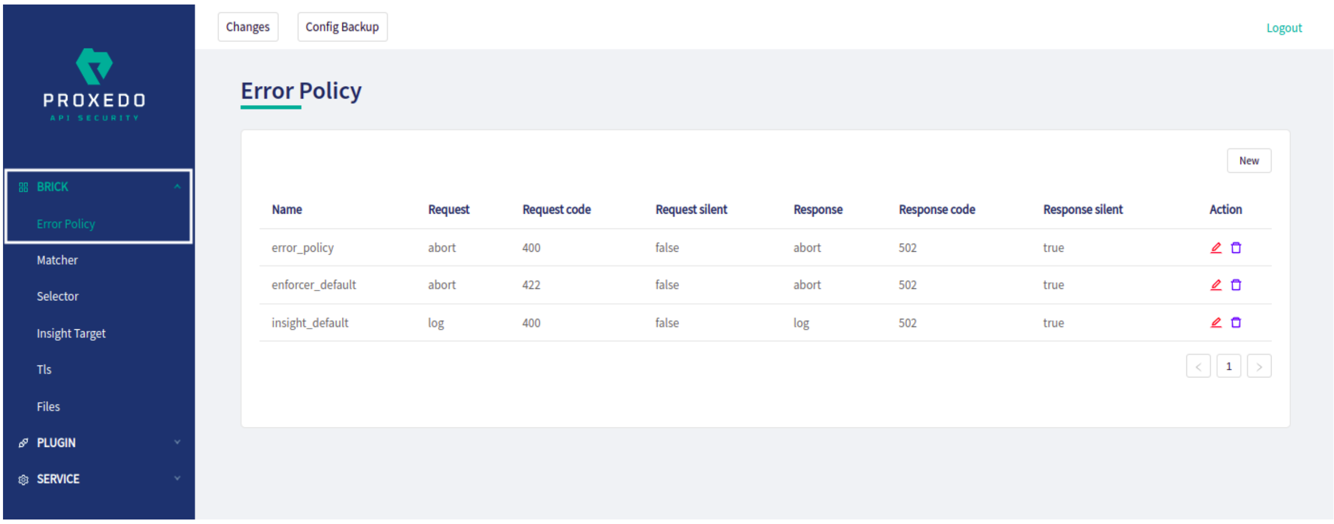
Every Plugin has a default error policy, namely, the 'error_policy', except for the Enforcer and the Insight Plugins, which have their own default error policies already configured for usage, the enforcer_default and the insight_default error policies.

6.4.1.1. Configuring Error policies

Error policies can be configured from the *BRICK* main menu item.

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

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



Error Policy

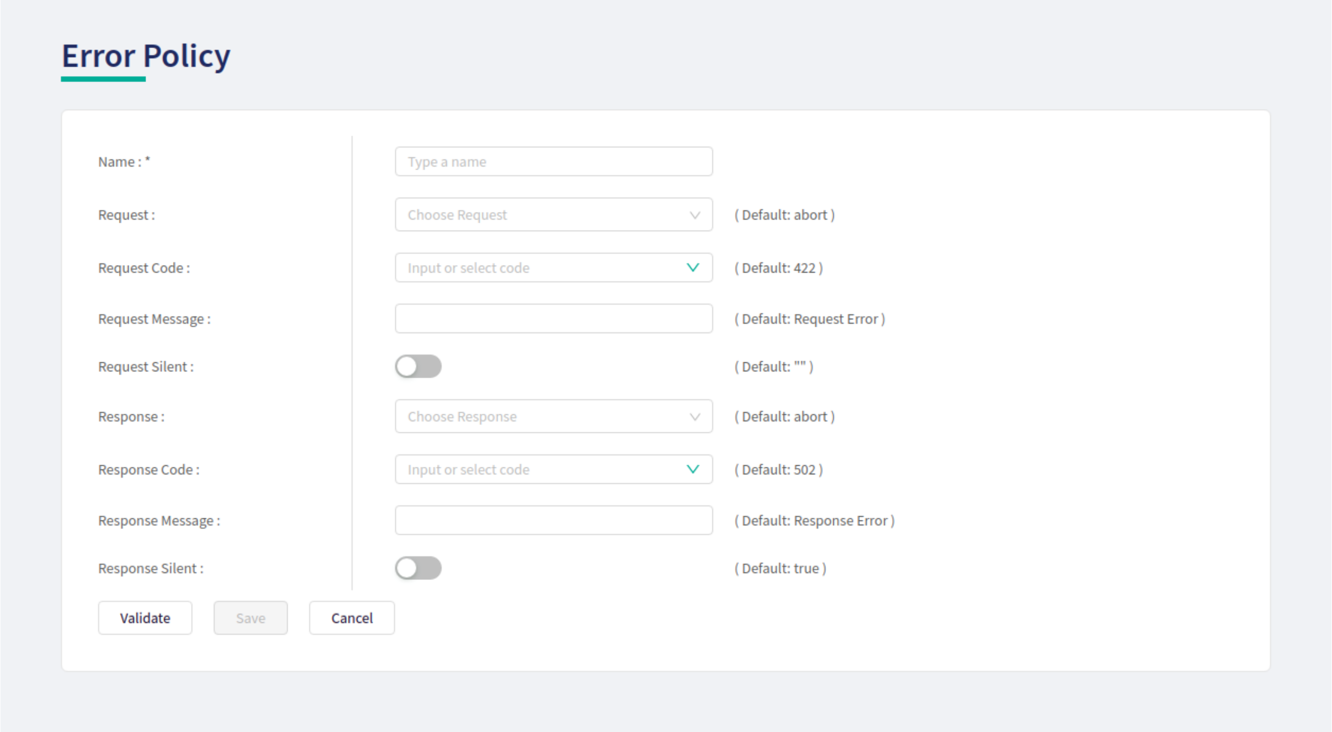
Name	Request	Request code	Request silent	Response	Response code	Response silent	Action
error_policy	abort	400	false	abort	502	true	✎ 🗑
enforcer_default	abort	422	false	abort	502	true	✎ 🗑
insight_default	log	400	false	log	502	true	✎ 🗑

Figure 7. 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.

An *Error Policy* contains the following settings:



Error Policy

Name : *	Type a name	
Request :	Choose Request	(Default: abort)
Request Code :	Input or select code	(Default: 422)
Request Message :		(Default: Request Error)
Request Silent :	○	(Default: "")
Response :	Choose Response	(Default: abort)
Response Code :	Input or select code	(Default: 502)
Response Message :		(Default: Response Error)
Response Silent :	○	(Default: true)

Figure 8. Configuring error policies in the Web User Interface

The following table provides details on what values can be figured for an *Error policy* and what these values define for an *Error policy*. Configure the following options:

Table 16. Error policy configuration options

Key	Values	Default value	Description
Name*	It is a mandatory value. It can be defined in free text.		It is 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 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	It provides the HTTP status code to be used when denying invalid requests.
Request message	The message can be provided in free text.	Request error	The reason is provided here in the HTTP response line when denying invalid requests.
Request silent	The parameter can be configured by switching it on or off. When it is switched on, the *Plugins* do not report on the denial of the invalid request. When it is turned off, the *Plugins* have the ability to report the error in detail in the body of the HTTP error request.	true	Do not report validation errors of the request to the client.
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.

Key	Values	Default value	Description
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	It provides the HTTP status code to be used when denying invalid requests.
Response message	The message can be provided in free text.	Response error	The reason is provided here that can be used in the HTTP response line when denying invalid requests.
Response silent	The parameter can be configured by switching it on or off. When it is switched on, the *Plugins* do not report on the denial of the invalid response. When it is turned off, the *Plugins* have the ability to report the error in detail in the body of the HTTP error response.	true	Do not report validation errors of the response to the client.

The default values in the above table represent the hard coded default values. They form a strict security policy: all errors are fatal, and only mistakes made by the client are reported in detail.

For configuring error policies, continue with completing the following steps:

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 suitable and adequate for configuring the 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. If the configuration of the component is satisfactory, after clicking the *Validate* button, the user receives the 'Component Validation successful' notification.
6. Click the *Save* button.

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

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.

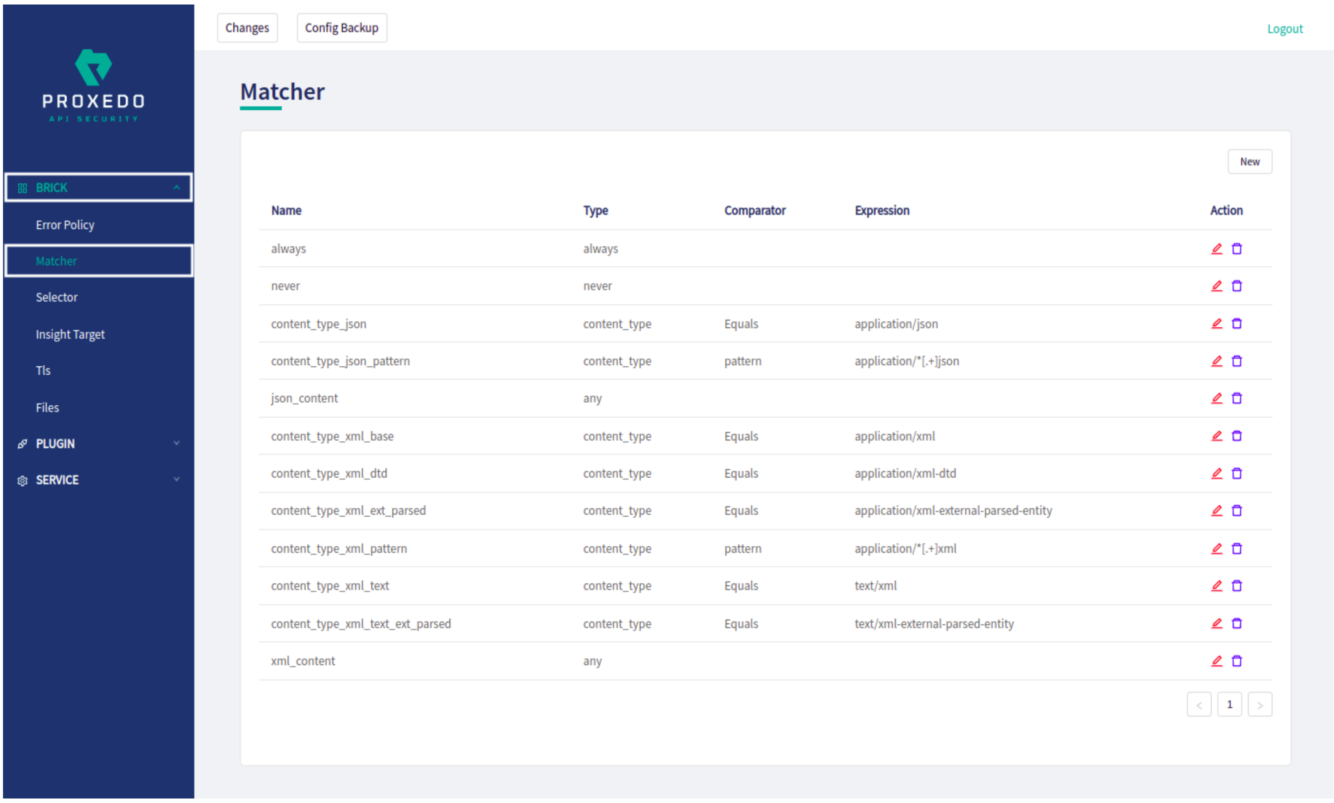
If you want to use comparator parameters, the comparator name should be given even if the default comparator is used.

6.4.2.1. Configuring Matchers

Matchers can be configured from the *BRICK* main navigation item.

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

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



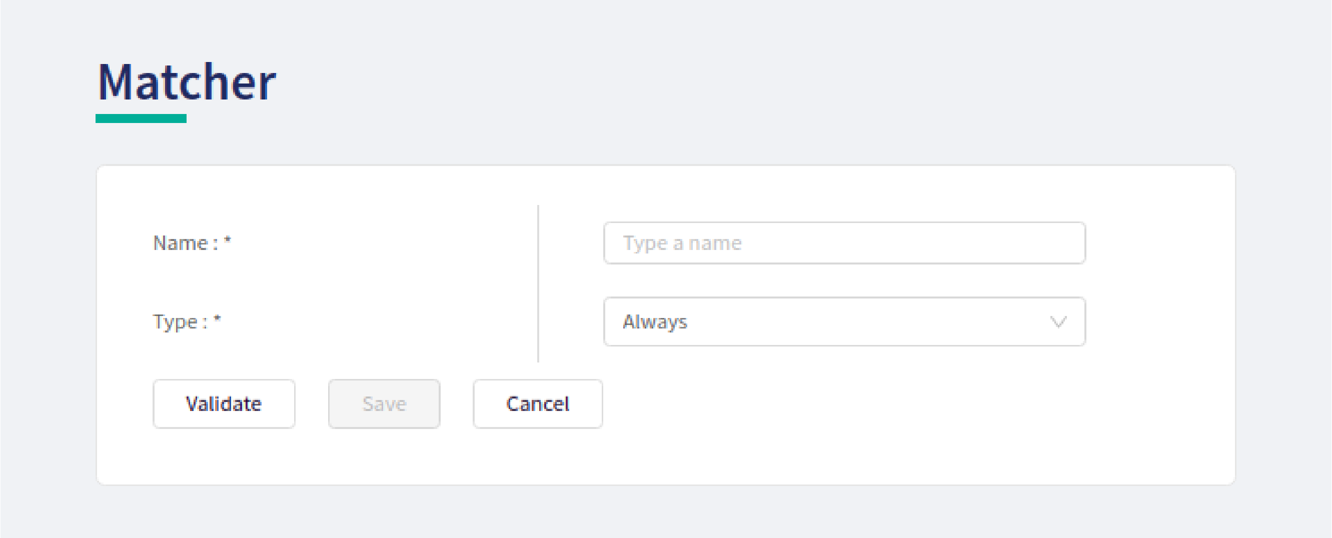
Matcher

Name	Type	Comparator	Expression	Action
always	always			✎ 🗑
never	never			✎ 🗑
content_type_json	content_type	Equals	application/json	✎ 🗑
content_type_json_pattern	content_type	pattern	application/*[.+]json	✎ 🗑
json_content	any			✎ 🗑
content_type_xml_base	content_type	Equals	application/xml	✎ 🗑
content_type_xml_dtd	content_type	Equals	application/xml-dtd	✎ 🗑
content_type_xml_ext_parsed	content_type	Equals	application/xml-external-parsed-entity	✎ 🗑
content_type_xml_pattern	content_type	pattern	application/*[.+]xml	✎ 🗑
content_type_xml_text	content_type	Equals	text/xml	✎ 🗑
content_type_xml_text_ext_parsed	content_type	Equals	text/xml-external-parsed-entity	✎ 🗑
xml_content	any			✎ 🗑

Figure 9. Matchers' main page in the Web User Interface

3. Click on the *New* navigation button to configure a matcher.

The generic configuration page for matchers provides the following settings:



Matcher

Name : *

Type : * Always

Figure 10. Configuring matchers in the Web User Interface

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

Table 17. Matcher configuration options

Key	Values	Default value	Description
Name*	It is a mandatory value. It can be defined in free text.	It can be defined in free text.	The **Name** of the matcher which can be referenced in Plugins.

Key	Values	Default value	Description
Type*	It is a mandatory value. For the available values, see Matcher types.		The preferred matcher type has to be selected from the drop-down list.

4. Provide the name of the matcher.
5. Choose the type of the matcher 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	It matches the direction of the message (request or response).
Method	It matches the HTTP method of the request. Note that the method is case insensitive by definition, therefore the case will always be ignored. 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.

Matcher type	Description

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.

Content type	It 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	It matches the status code of the response. 💡 See the default Status class comparator which allows convenient matching on HTTP status classes. The available values for the *Expression* parameter are: Informational response, Successful response, Redirects, Client errors, Server Errors. 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	It matches the original content of the message. If the content type is JSON, the body will be decompressed but not parsed. When choosing the *Raw content* matcher from the drop-down list, additional parameters appear. For more information on the configuration of these parameters, see Matcher types' additional configuration options.
Client_address	It 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	It matches the client's port (TCP).
Server_address	It 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	It mathces the server's port (TCP).
xpath	It 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	It 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: main 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.

Matcher type	Description
Fraud_detector_score	It 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`
	ℹ 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	It 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.

Matcher type	Description
URI netloc	It 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.
URI origin	It 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	It 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	It 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.

Matcher type	Description
URI password	It 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	It 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	It 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.
URI path	It 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	It matches the *path* part of the URI, without the normalization of URI path matcher carried out. ℹ 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.

Matcher type	Description
URI raw query	It 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	It 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	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	It 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.

Matcher type	Description
Soap header	It 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	It 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	It 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	Soap 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. It 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.
Soap fault detail	It 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 11 fault faultstring	It matches the soap fault 'faultstring'. This matcher only works with soap version 1.1. When choosing the *Soap 11 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.

Matcher type	Description
Soap 11 fault faultactor	It matches the soap fault 'faultactor'. This matcher only works with soap version 1.1. When choosing the *Soap 11 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 12 fault reason	It matches the soap fault 'Reason'. This matcher only works with soap version 1.2. When choosing the *Soap 12 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 12 fault node	It matches the soap fault 'Node'. This matcher only works with soap version 1.2. When choosing the *Soap 12 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 12 fault role	It matches the soap fault 'Role'. This matcher only works with soap version 1.2. When choosing the *Soap 12 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.
Ignorecase		Off (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 ignorcase field.

Key	Values	Default value	Description
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 complete explanation on how to write JMESPath expressions is not in the scope of this document. To learn more about it visit the: main 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			It 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			It defines the XML namespaces.
Xpath Expression*			The expression to extract the node from the call to match against.
Multiline			It sets the Multiline flag for the *Regex* comparator.
Minimum*			It matches if the pattern is larger or equal to the value.
Maximum*			It matches if the pattern is smaller or equal to the value.

Key	Values	Default value	Description
Source Plugin	Fraud Detector Plugins can be referenced here by selecting them from the drop-down list.	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.	This parameter defines which Fraud Detector plugin shall be used in case there are more than one defined.

6. Configure the necessary parameters with the help of the above tables.
7. Click the *Validate* button to check if the defined parameters are suitable and adequate for configuring the 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. If the configuration of the component is satisfactory, after clicking the *Validate* button, the user receives the 'Component Validation successful' notification.
8. Click the *Save* button to save the configured matcher.

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.

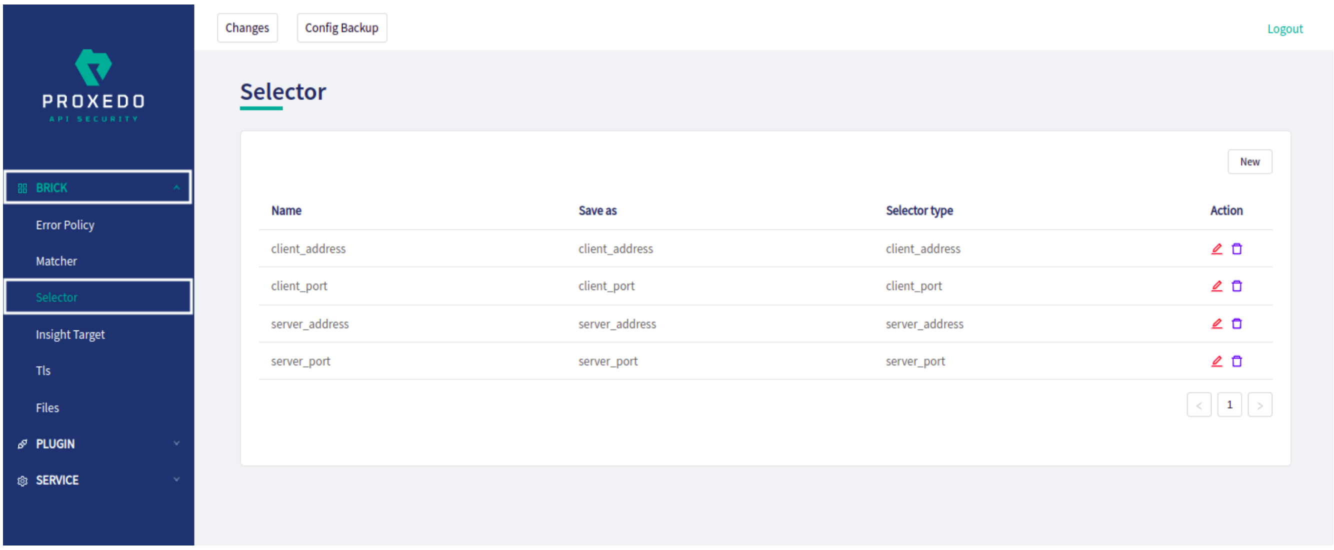
They are used by Insight.

6.4.3.1. Configuring Selectors

The selector can be configured from the *BRICK* main navigation item.

1. Click on the *BRICK* main configuration item in the left navigation area. Alternatively you can also click on the ⌄ sign to open up the sub-navigation items of *BRICK*.
2. Select *Selectors*.

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:



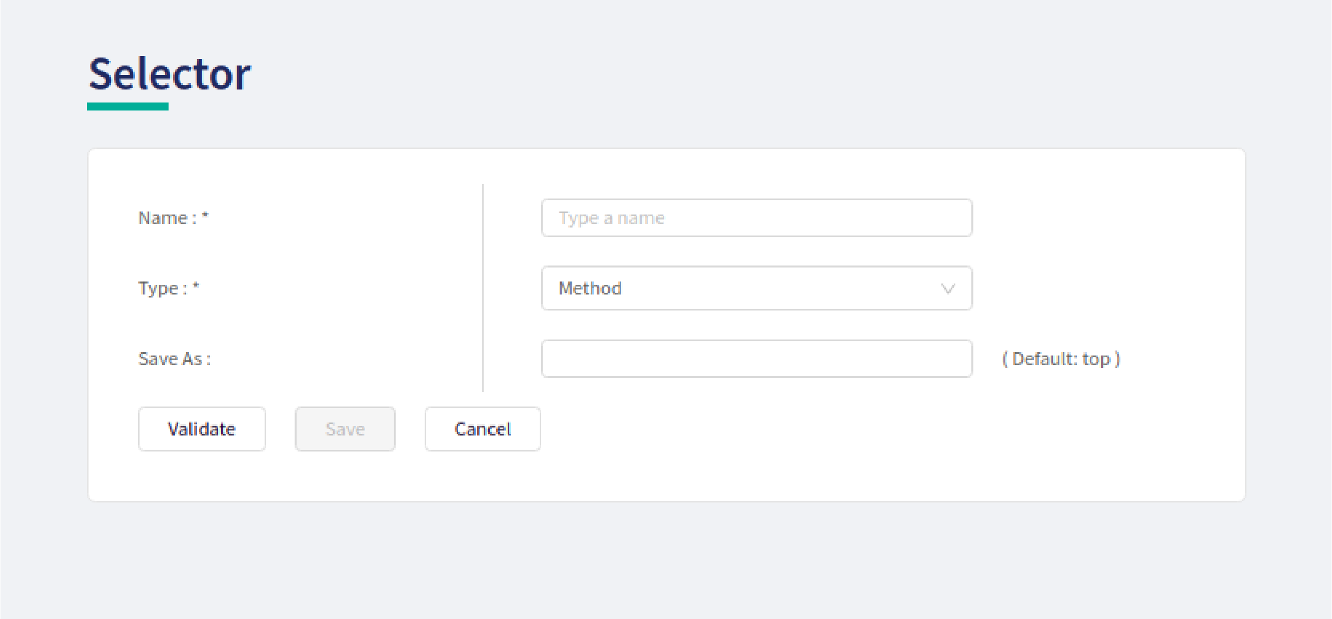
Selector

Name	Save as	Selector type	Action
client_address	client_address	client_address	
client_port	client_port	client_port	
server_address	server_address	server_address	
server_port	server_port	server_port	

Figure 11. Selector main page in the Web User Interface

3. Click on the *New* navigation button to configure the *Selector*.

The following configuration options appear for *Selector*:



Selector

Name : * Type a name

Type : * Method

Save As : (Default: top)

Figure 12. Configuring Selector in the Web User Interface

The selector accepts the following configuration options:

Table 23. Selector configuration options

Key	Values	Default value	Description
Name*	It is a mandatory value. It can be defined in free text.		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.

Key	Values	Default value	Description
Save as	The key under which the results of a selector are saved in the *Insight plugin's* dictionary.	Top	If it is omitted, the result will be directly merged as top level keys. 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.

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
Clear text	It can be switched On or Off.		
Namespaces	It defines the XML namespaces.		
Xpath Expression			The expression to extract the node from the call to match against.
JmesPath Expression			A complete explanation on how to write JMESPath expressions is not in the scope of this document. To learn more about it visit the: main 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.
Time format	'YYYY-MM-DDTHH:mm:ss.SSSSSSZ Z'	Set the format. See: Timestamp format options	
Time zone		UTC	It is 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).

Key	Values	Default value	Description
Source Plugin	Fraud Detector Plugins can be referenced here by selecting them from the drop-down list.	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.	This parameter defines which Fraud Detector plugin shall be used in case there are more than one defined.

4. Name the *Selector* key.
5. Fill in any more desired parameters.
6. Click the *Validate* button to check if the defined parameters are suitable and adequate for configuring the 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. If the configuration of the component is satisfactory, after clicking the *Validate* button, the user receives the 'Component Validation successful' notification.
7. Click the *Save* button if you have configured all the required parameters.

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.

> ⚠ Unlike other bricks, *Insight Target* configurations cannot be put inline into a *Plugin's* configuration, they must always be configured here.

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* are 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 forward slash character ("/").
- Keys in nested dictionaries are added to the path by name.
- List items are added to the path by their index.

> You can control the separator with the **Flatten separator** configuration key that every *Insight Target* accepts.

6.4.4.2. Configuring Insight Targets

The *Insight Target* can be configured from the *BRICK* main navigation item.

1. Click on the *BRICK* main configuration item in the Left navigation area. Alternatively you can also click on the ⌄ sign to open up the sub-navigation items of *BRICK*.
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:

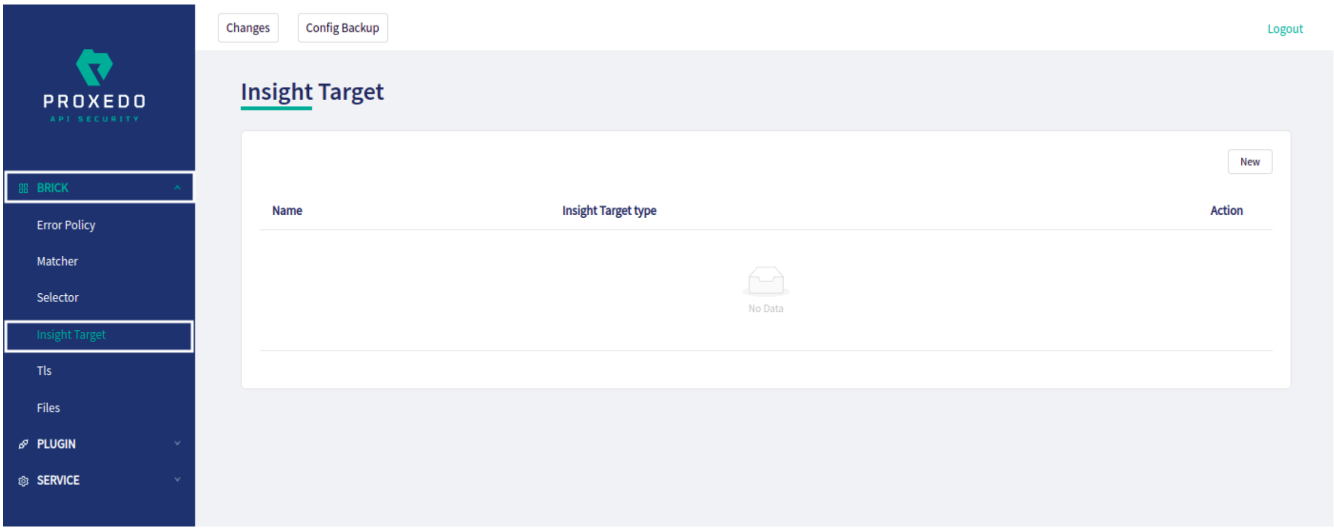
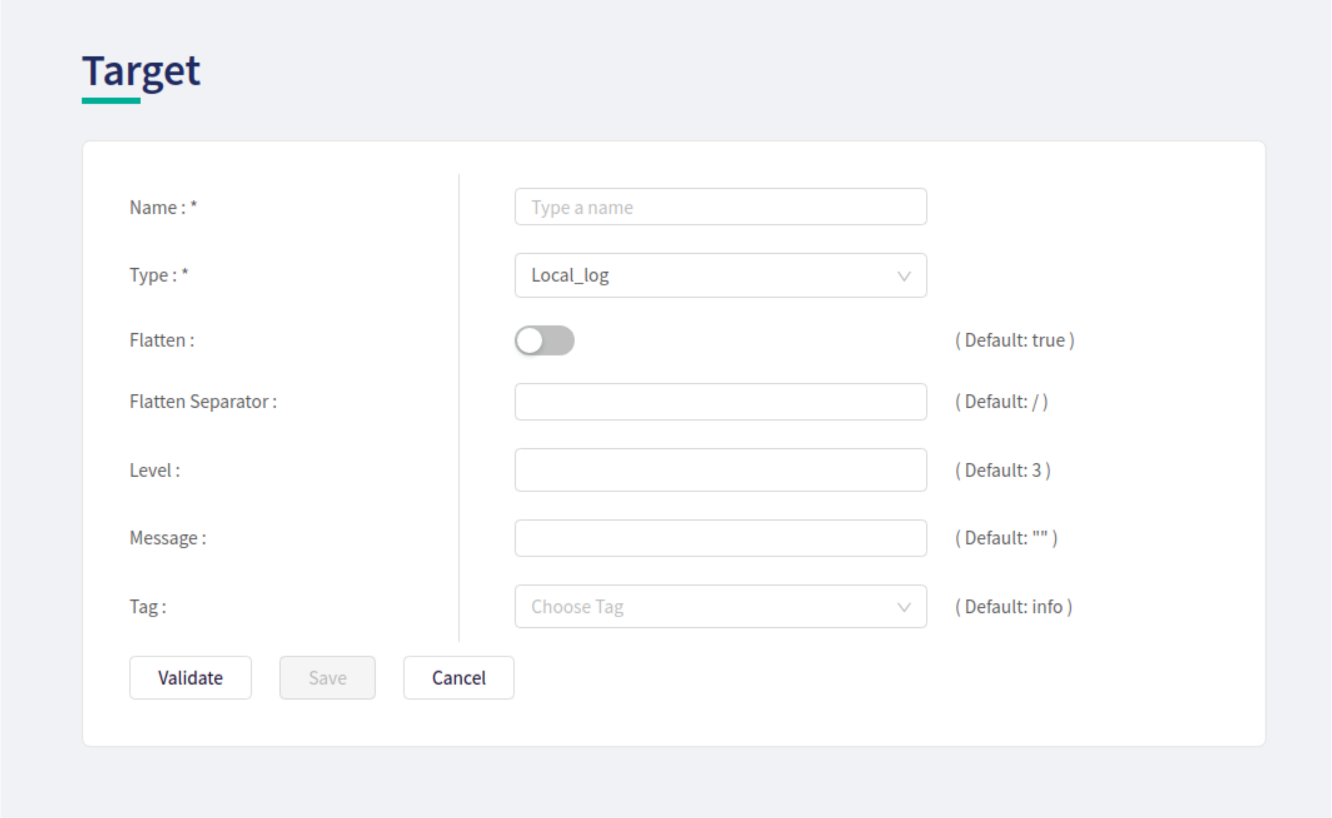


Figure 13. Insight Target main page in the Web User Interface

3. Click on the *New* navigation button to configure the *Insight Target*.



Name : * Type a name

Type : * Local_log

Flatten : (Default: true)

Flatten Separator : (Default: /)

Level : (Default: 3)

Message : (Default: "")

Tag : Choose Tag (Default: info)

Figure 14. Configuring Insight Target in the Web User Interface

The *Insight Target* accepts the following configuration options:

Table 25. Insight Target configuration options

Key	Values	Default value	Description
Name*	It is a mandatory value. It can be defined in free text.		It is the name identifying the *Insight Target*. This name of the *Insight Target* can be referenced from other parts of the configuration.
Type*	It is a mandatory value. The values can be selected from the drop-down list. The available values are: • Local log • Syslog • Elastic		• Local log: It 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: It 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: It sends the insight to an *Elasticsearch* engine in JSON. For more details on configuration settings with syslog, see Elastic *Insight Target* configuration parameters.

Key	Values	Default value	Description
Flatten	This parameter can be switched 'on' or 'off'.	On (True)	Flatten the *Insight Target* message.
Flatten separator		/	It is the separator in the flattened message.
Level		3	It is the log level for the logged message.
Message		It is the message of the insight if present, otherwise it is empty.	It is the message part of the log message.
Tag	The value can be selected from a drop-down list.	info	It is the log tag for the logged message.

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.

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 separator		/	It is the separator in the flattened message.
Level		3	It provides the log level for the logged message.
Message		The message of the insight if present, otherwise it is empty.	It is the message part of the log message.
Tag		info	It is 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
Data format	The possible values are: sdata, json.	sdata	This is the data format of the insight.
Enable heartbeat		False	It enables sending heartbeat (-- MARK --) messages to the *Insight Target*.
Flatten		True	It flattens the *Insight Target* message.
Flatten Separator		/	It is the separator in the flattened message.

Key	Values	Default value	Description
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.
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.
Host*			It is the hostname or the IP address of the syslog server.
IP protocol	The possible values are 4 and 6, corresponding to IPv4 and IPv6.		This determines the internet protocol version of the given driver.
Mask credit card numbers		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.
Remote Connection	• Protocol: The available values are: TCP and UDP. • Port: The available values are integers. • Use TLS: The available values are True or False. • **Syslog TLS***: Select the *Syslog TLS* brick you want to use for the *Insight Target*.	• Protocol: TCP, Port 601 • Protocol: UDP, Port: 514 • Use TLS: False	• Protocol: Add the transport protocol to send messages over. The available values are: TCP and UDP. • Port: Add the port number here to connect to the remote system. • Use TLS: It enables using TLS for the Syslog communication. • **Syslog TLS***: It is mandatory if the *Use TLS* option is set to True.
Report config load		False	It reports the event of a configuration being loaded with a cryptographic hash of the loaded configuration. This ingorms the *Insight Target* about changes in the configuration.
Second fraction digits	Integer between 0 and 6 inclusive	3	The number of digits representing the fractions of seconds in the Syslog timestamp.

Key	Values	Default value	Description
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).

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
Doc type		_doc	The doc type is used when sending the data.
Flatten		True	It flattens the *Insight Target* message.
Flatten Separator		/	It is the separator in the flattened message.
Host*			It is the hostname of the Elastic search instance.
Index*			It is the name of the index in the Elastic search instance.
Mask credit card numbers		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.
Port		9200	Add the port number here to connect to the remote system.

7. Configure any more desired parameter details.
8. Click the *Validate* button to check if the defined parameters are suitable and adequate for configuring the 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. If the configuration of the component is satisfactory, after clicking the *Validate* button, the user receives the 'Component Validation successful' notification.
9. Click *Save* to save your configuration settings for the *Insight Target*.

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 Insight Target*.

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

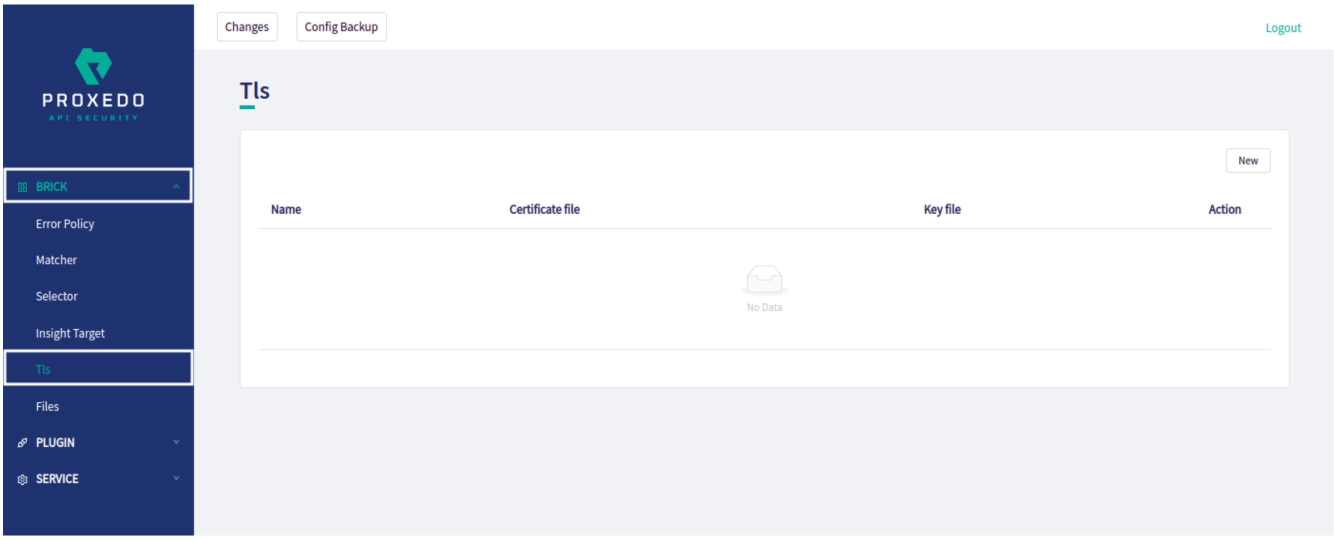
> These parameters are used by the *Insight Director* and the *Transport Director*. For options that reference a file the path is relative to `/opt/balasys/var/persistent/` inside the *Transport Director* container. This directory is a docker volume and by default mounted from the `/opt/balasys/var/persistent/transport-director`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

6.4.5.1. Configuring the TLS

TLS can be configured from the *BRICK* main navigation item.

1. Click on the *BRICK* main configuration item in the Left navigation area. Alternatively you can also click on the ⌄ sign to open up the sub-navigation items of *BRICK*.
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:

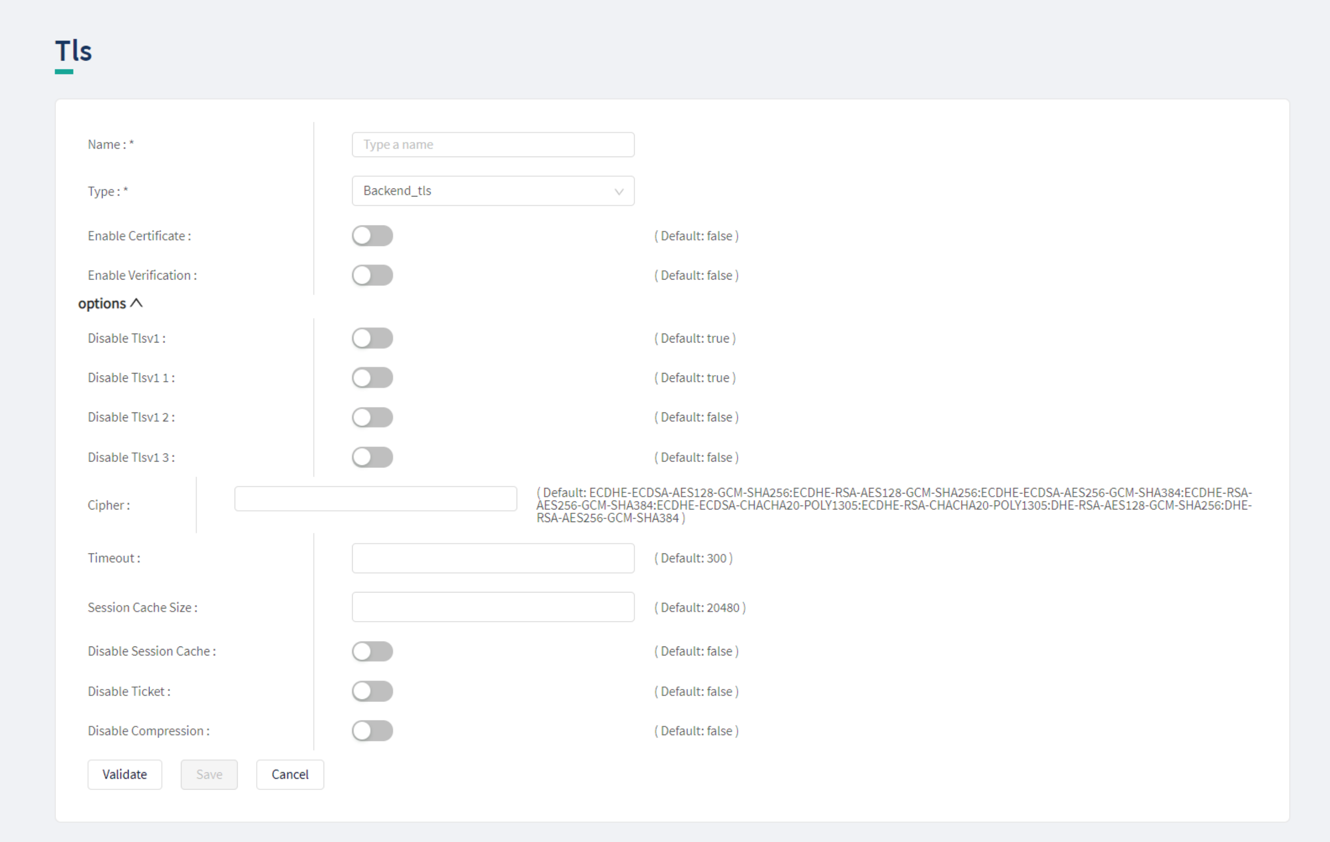


Name	Certificate file	Key file	Action
No Data			

Figure 15. TLS main page in the Web User Interface

3. Click on the *New* navigation button to configure TLS.

TLS contains the following settings:



Tls

Name : *	Type a name	
Type : *	Backend_tls	
Enable Certificate :		(Default: false)
Enable Verification :		(Default: false)

options ^

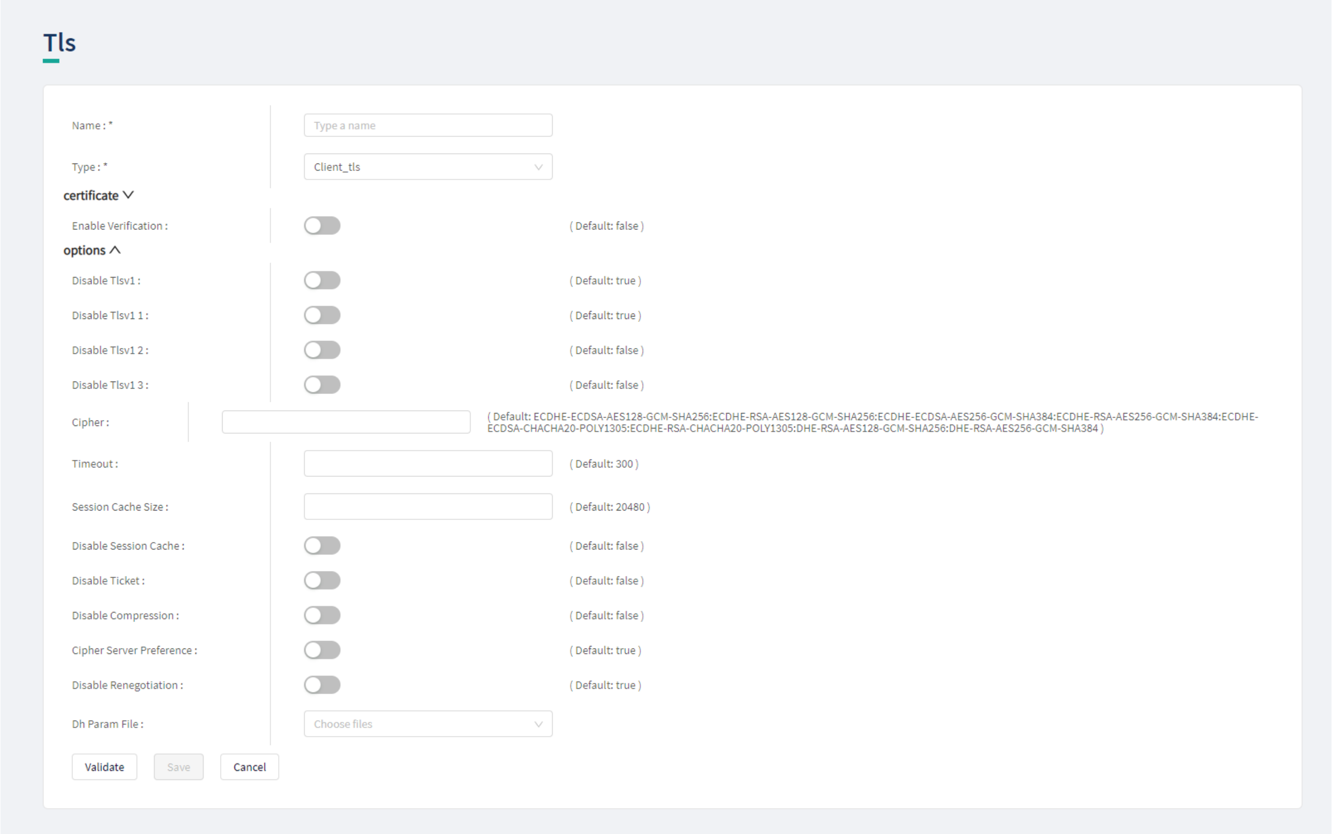
Disable Tlsv1 :		(Default: true)
Disable Tlsv1 1 :		(Default: true)
Disable Tlsv1 2 :		(Default: false)
Disable Tlsv1 3 :		(Default: false)
Cipher :		(Default: ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-CHACHA20-POLY1305:ECDHE-RSA-CHACHA20-POLY1305:DHE-RSA-AES128-GCM-SHA256:DHE-RSA-AES256-GCM-SHA384)
Timeout :		(Default: 300)
Session Cache Size :		(Default: 20480)
Disable Session Cache :		(Default: false)
Disable Ticket :		(Default: false)
Disable Compression :		(Default: false)

Figure 16. 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 or a *Syslog TLS* configuration.

6.4.5.1.1. Configuring the Client TLS

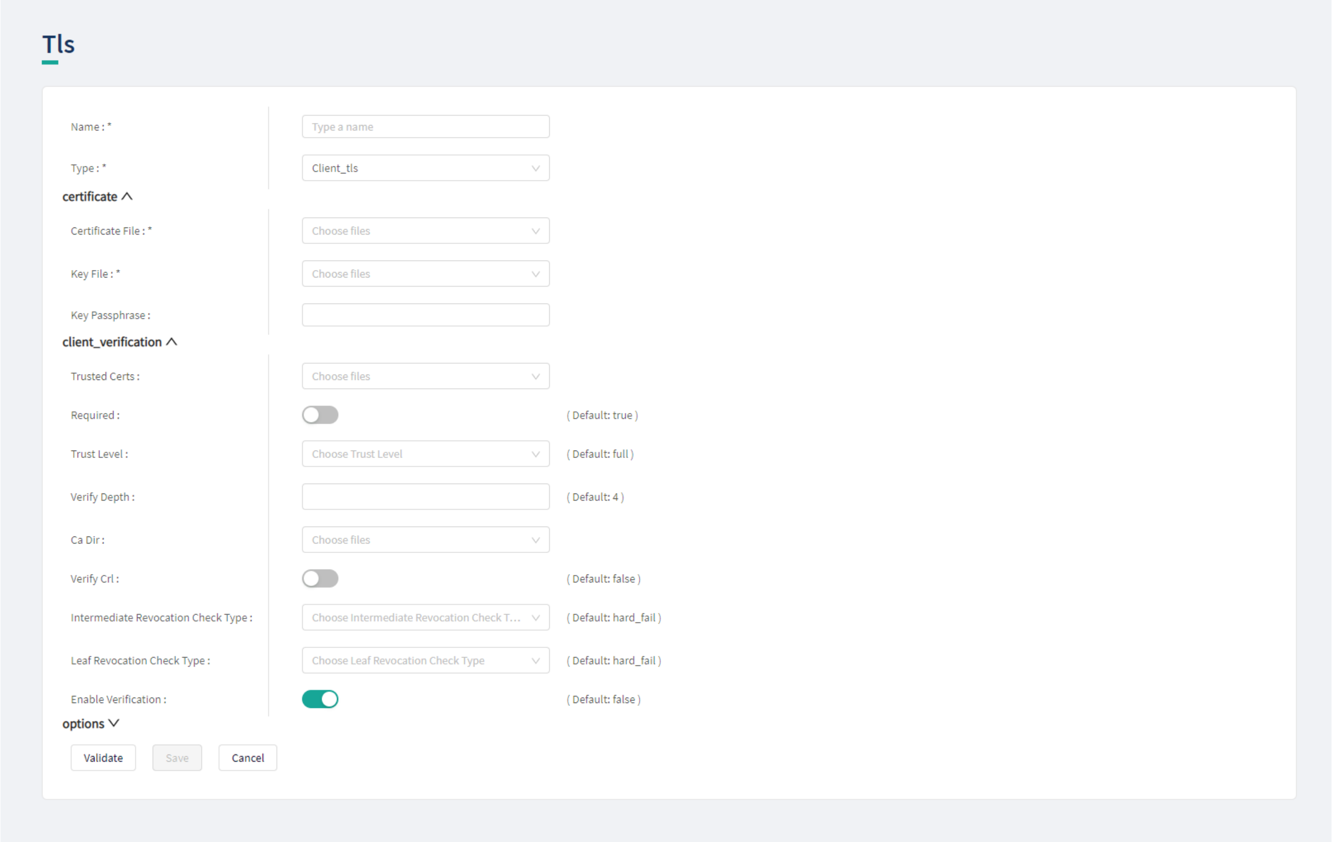
The following parameters need to be configured for *Client TLS*:



Tls

Name : *	Type a name	
Type : *	Client_tls	
certificate ∨		
Enable Verification :	○	(Default: false)
options ∧		
Disable Tlsv1 :	○	(Default: true)
Disable Tlsv1 1 :	○	(Default: true)
Disable Tlsv1 2 :	○	(Default: false)
Disable Tlsv1 3 :	○	(Default: false)
Cipher :		(Default: 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)
Timeout :		(Default: 300)
Session Cache Size :		(Default: 20480)
Disable Session Cache :	○	(Default: false)
Disable Ticket :	○	(Default: false)
Disable Compression :	○	(Default: false)
Cipher Server Preference :	○	(Default: true)
Disable Renegotiation :	○	(Default: true)
Dh Param File :	Choose files	

Figure 17. Configuring Client TLS in the Web User Interface, TLS options



Tls

Name : *	Type a name	
Type : *	Client_tls	
certificate ∧		
Certificate File : *	Choose files	
Key File : *	Choose files	
Key Passphrase :		
client_verification ∧		
Trusted Certs :	Choose files	
Required :	○	(Default: true)
Trust Level :	Choose Trust Level	(Default: full)
Verify Depth :		(Default: 4)
Ca Dir :	Choose files	
Verify Crl :	○	(Default: false)
Intermediate Revocation Check Type :	Choose Intermediate Revocation Check T...	(Default: hard_fail)
Leaf Revocation Check Type :	Choose Leaf Revocation Check Type	(Default: hard_fail)
Enable Verification :	●	(Default: false)
options ∨		

Figure 18. 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*	It is a mandatory value. It can be defined in free text.		The name of the parameter can be referenced.
Type*	It is a mandatory value. Choose the required value from the drop-down list.		Client TLS, Backend TLS and Syslog 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*	It is a mandatory value. You can upload the certificate file.		Provide the path and filename for the certificate file. The certificate file must be in PEM format.
Key file*	It is a mandatory value. You can upload the key file.		Provide the path and filename to the private key file. The private key must be in PEM format.
Key passphrase	You can upload the file.		Provide the passphrase used to access the private key specified in the Key file.
Enable Verification		Off (False)	It is an option for verifying client side X.509 certificates. By default no client verification takes place.
Client verification			Client verification options
Trusted Certs	You can upload trusted certificates in a ZIP file.		This is a Certificate *File* element from among the *Brick* components.
Required	The parameter can be switched on or off.	On (true)	If it is set to True, PAS requires a certificate from the peer.

Key	Values	Default value	Description
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	It defines the length of the longest accepted CA verification chain. PAS will automatically reject longer CA chains.
Ca Dir	You can upload the trusted CAs in a ZIP file.		This is a Certificate *File* element from among the *Brick* components.
Verify Crl	The parameter can be switched on or off.	Off (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
Options			TLS protocol options used on the listener.
Disable TLS v1	The parameter can be switched on or off. If it is set *ON* it does not allow using TLSv1 in the connection.	On (true)	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.

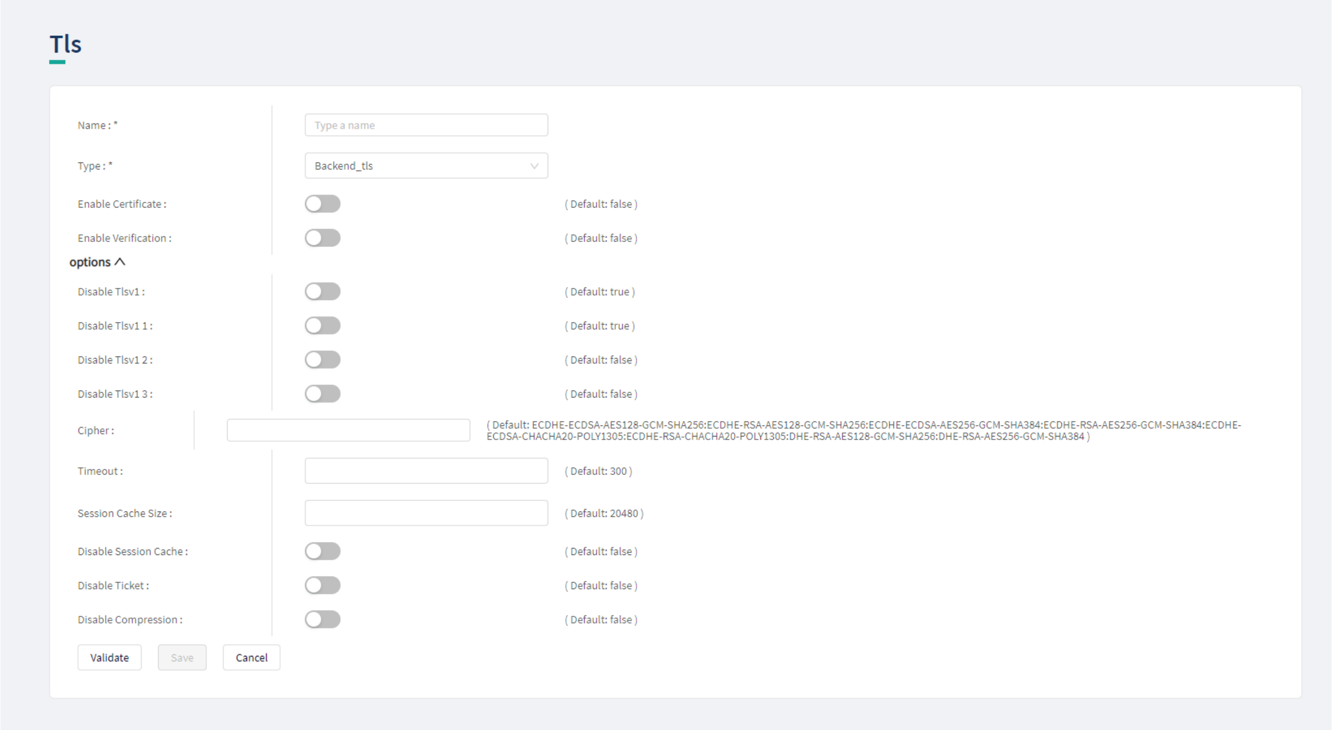
Key	Values	Default value	Description
Disable TLS v1.1	The parameter can be switched on or off. If it is set *ON* it does not allow using TLSv1.1 in the connection.	On (true)	It does not allow the usage of TLSv1.1 in the connection.
Disable TLS v1.2	The parameter can be switched on or off. If it is set *ON* it does not allow using TLSv1.2 in the connection.	Off (false)	It does not allow the usage of TLSv1.2 in the connection.
Disable TLS v1.3	The parameter can be switched on or off. If it is set *ON* it does not allow using TLSv1.3 in the connection.	Off (false)	It does not allow the usage of TLSv1.3 in the connection.
Cipher		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.
Timeout		300	It drops idle connection if the timeout value (in seconds) expires.
Session Cache Size		20480	It defines the number of sessions stored in the session cache for SSL session reuse
Disable Session Cache	The parameter can be switched on or off.	Off (false)	Do not store session information in the session cache. Set this option to 'on' to disable SSL session reuse.
Disable Ticket	The parameter can be switched on or off.	Off (false)	Session tickets are a method for SSL session reuse, described in RFC 5077. Set this option to ON to disable SSL session reuse using session tickets.

Key	Values	Default value	Description
Disable Compression	The parameter can be switched on or off.	Off (false)	Set the parameter *On* to disable support for SSL/TLS compression. Set the parameter *Off* to enable support for SSL/TLA compression.
Cipher Server Preference	The parameter can be switched on or switched off.	On (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	The parameter can be switched on or off.	On (true)	Set this parameter *On* to disable client-initiated renegotiation.
Dh Parameter File			You can upload the DH parameter file. The DH parameter file must be in PEM format.

4. Click the *Validate* button to check if the defined parameters are suitable and adequate for configuring the 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. If the configuration of the component is satisfactory, after clicking the *Validate* button, the user receives the 'Component Validation successful' notification.
5. Save the *Client TLS* configuration by clicking *Save*.

6.4.5.1.2. Configuring Backend TLS

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



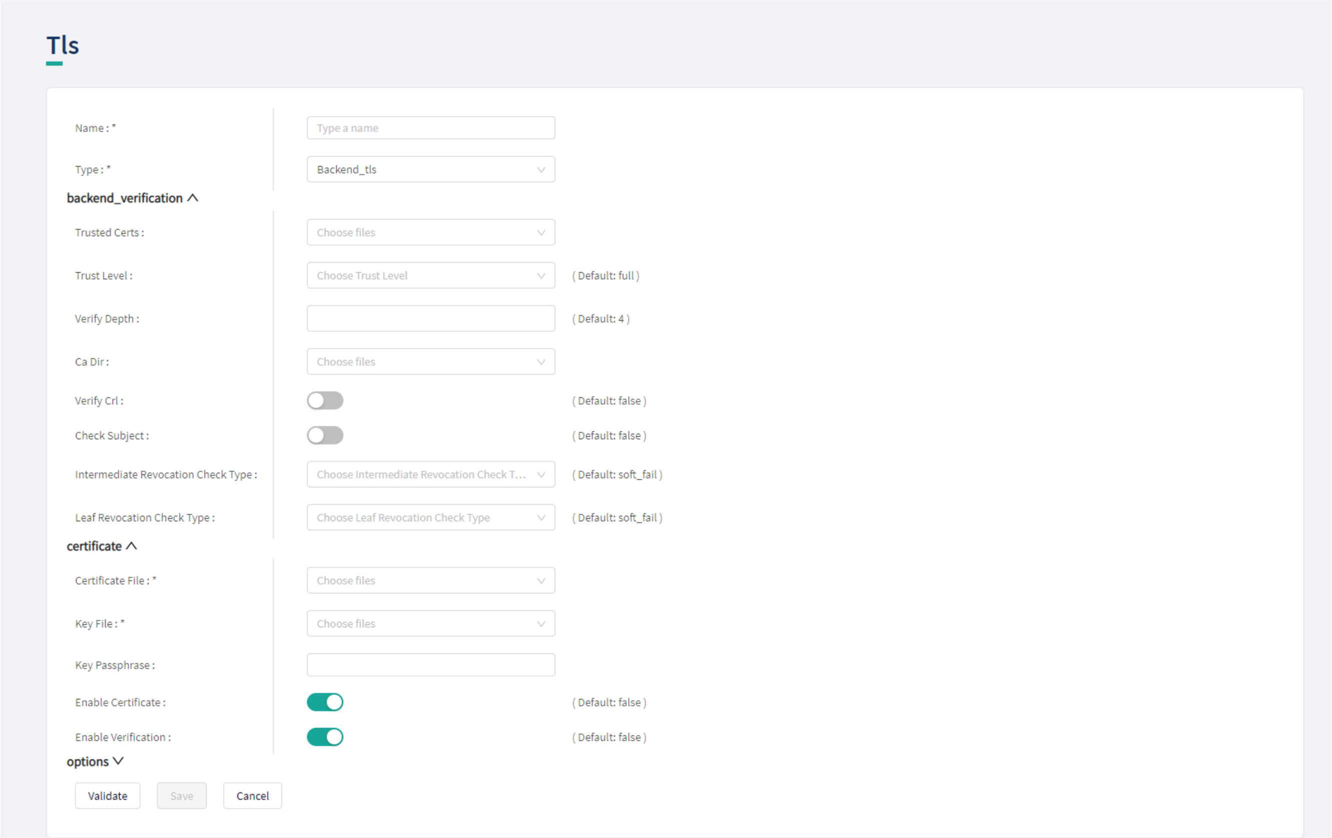
Tls

Name : *	Type a name	
Type : *	Backend_tls	
Enable Certificate :	○	(Default: false)
Enable Verification :	○	(Default: false)

options

Disable Tlsv1 :	○	(Default: true)
Disable Tlsv1 1 :	○	(Default: true)
Disable Tlsv1 2 :	○	(Default: false)
Disable Tlsv1 3 :	○	(Default: false)
Cipher :		(Default: 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)
Timeout :		(Default: 300)
Session Cache Size :		(Default: 20480)
Disable Session Cache :	○	(Default: false)
Disable Ticket :	○	(Default: false)
Disable Compression :	○	(Default: false)

Figure 19. Configuring Backend TLS in the Web User Interface, TLS options



Tls

Name : * Type a name

Type : * Backend_tls

backend_verification

Trusted Certs : Choose files

Trust Level : Choose Trust Level (Default: full)

Verify Depth : (Default: 4)

Ca Dir : Choose files

Verify Crl : (Default: false)

Check Subject : (Default: false)

Intermediate Revocation Check Type : Choose Intermediate Revocation Check T... (Default: soft_fail)

Leaf Revocation Check Type : Choose Leaf Revocation Check Type (Default: soft_fail)

certificate

Certificate File : * Choose files

Key File : * Choose files

Key Passphrase :

Enable Certificate : (Default: false)

Enable Verification : (Default: false)

options

Figure 20. 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*	It is a mandatory value. It can be defined in free text.		The name of the parameter can be referenced.
Type*	It is a mandatory value. Choose the required value from the drop-down list.		Client TLS, Backend TLS and Syslog 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
Certificate			Configuration for the X.509 certificate used for TLS connections on the listener.

Key	Values	Default value	Description
Enable Certificate		Off/False	It is an option for enabling backend side X.509 certificates. By default no backend verification takes place.
Enable Verification		Off/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	You can upload trusted certificates in a ZIP file.		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	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	It defines the length of the longest accepted CA verification chain. PAS will automatically reject longer CA chains.
Ca Dir	You can upload the trusted CAs in a ZIP file.		It is a directory where the trusted CA certificates are stored. CA certificates are loaded on-demand from this directory when PAS verifies the certificate of the peer.
Verify Crl	The parameter can be switched on or off.	Off (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.

Key	Values	Default value	Description
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.
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.
Check Subject	The parameter can be switched on or off.	Off (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).
Options			TLS protocol options used on the listener.
Disable TLS v1	The parameter can be switched on or off. If it is set *ON* it does not allow using TLSv1 in the connection.	On (true)	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.
Disable TLS v1.1	The parameter can be switched on or off. If it is set *ON* it does not allow using TLS v. 1.1 in the connection.	On (true)	It does not allow the usage of TLS v. 1.1 in the connection.
Disable TLS v1.2	The parameter can be switched on or off. If it is set *ON* it does not allow using TLS v. 1.2 in the connection.	false	It does not allow the usage of TLS v. 1.2 in the connection.

Key	Values	Default value	Description
Disable TLS v1.3	The parameter can be switched on or off. If it is set to *ON* it does not allow using TLS v. 1.3 in the connection.	false	It does not allow the usage of TLS v. 1.3 in the connection.
Cipher		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.
Timeout	The parameter can be switched on or off. If it is set *ON* it does not allow using TLSv1 in the connection.	300	It drops idle connection if the timeout value (in seconds) expires.
Session cache size	The parameter can be switched on or off. If it is set *ON* it does not allow using TLSv1 in the connection.	20480	It defines the number of sessions stored in the session cache for SSL session reuse
Disable session cache	The parameter can be switched on or off.	Off (false)	Do not store session information in the session cache. Set this option to 'On' to disable SSL session reuse.
Disable ticket	The parameter can be switched on or off.	Off (false)	Do not store session information in the session cache. Set this option to 'On' to disable SSL session reuse.
Disable compression	The parameter can be switched on or off.	Off (false)	Set the parameter *On* to disable support for SSL/TLS compression. Set the parameter *Off* to enable support for SSL/TLA compression.

4. Click the *Validate* button to check if the defined parameters are suitable and adequate for configuring the 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. If the configuration of the component is satisfactory, after clicking the *Validate* button, the user receives the 'Component Validation successful' notification.

5. Click the *Save* button if you have configured all the required parameters.

6.4.5.1.3. Revocation checks for certificates

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

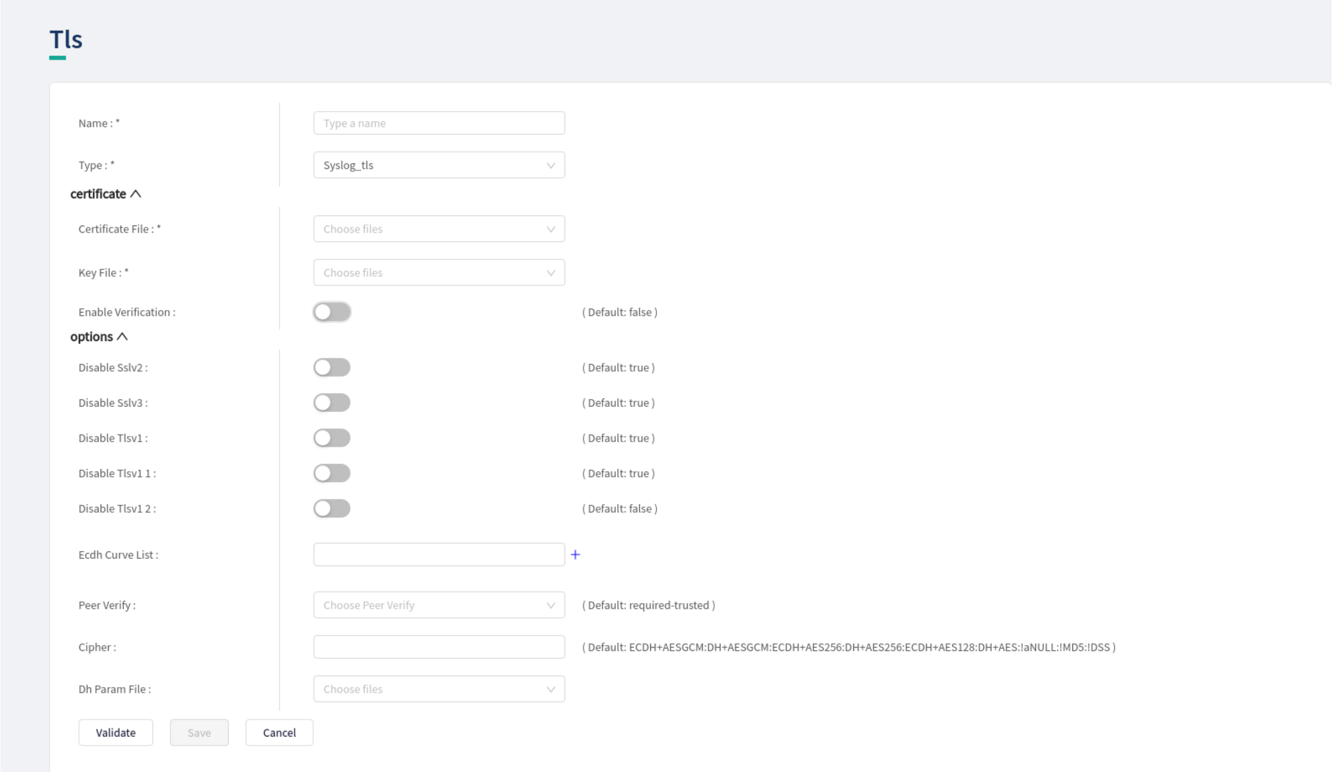
The result for a certificate, according to the revocation check types is as follows:

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

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



Tls

Name : * Type a name

Type : * Syslog_tls

certificate

Certificate File : * Choose files

Key File : * Choose files

Enable Verification : (Default: false)

options

Disable Sslv2 : (Default: true)

Disable Sslv3 : (Default: true)

Disable Tlsv1 : (Default: true)

Disable Tlsv1 1 : (Default: true)

Disable Tlsv1 2 : (Default: false)

Ecdh Curve List :

Peer Verify : Choose Peer Verify (Default: required-trusted)

Cipher : (Default: ECDH+AESGCM:DH+AESGCM:ECDH+AES256:DH+AES256:ECDH+AES128:DH+AES:!aNULL:!MD5:!DSS)

Dh Param File : Choose files

Figure 21. 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*	It is a mandatory value. It can be defined in free text.		The name of the parameter can be referenced.
Type*	It is a mandatory value. Choose the required value from the drop-down list.		Client TLS, Backend TLS and Syslog 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
Certificate			It is the configuration for the X.509 certificate used for TLS connections on the *Insight Target*.
Certificate File*	It is a mandatory value. You must select a *File* brick of type *generic* that represents the uploaded certificate.		Provide the name of the selected *File brick*. The certificate must be in PEM format.
Key file*	It is a mandatory value. You can select a *generic* file type that represents the uploaded private key.		Provide the name of the selected *File brick*. The private key must be in PEM format.
Enable Verification		Off (false)	It is an option for enabling the verification of server side X.509 certificates.
Options			TLS protocol options used on the *Syslog Insight target*.
Disable TLS v1	The parameter can be switched on or off. If it is set *ON* it does not allow using TLSv1 in the connection.	On (true)	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.
Disable TLS v1.1	The parameter can be switched on or off. If it is set *ON* it does not allow using TLSv1.1 in the connection.	On (true)	It does not allow the usage of TLSv1.1 in the connection.

Key	Values	Default value	Description
Disable TLS v1.2	The parameter can be switched on or off. If it is set *ON* it does not allow using TLSv1.2 in the connection.	Off (false)	It does not allow the usage of TLSv1.2 in the connection.
ECDH curve list	Add one or more names of ECDH curves. The possible values are the ones supported by OpenSSL 1.1.1.	empty list	This is 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	It 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).
Cipher	It is the colon-separated list of ciphers from the list supported by OpenSSL 1.1.1.	ECDH+AESGCM: DH+AESGCM:ECDH+AES256: DH+AES256:ECDH+AES128: DH+AES:!aNULL:!MD5: !DSS!aNULL: !MD5: !DSS	It specifies the allowed ciphers.
DH Parameter File	Select a *File* brick of type *generic* from the drop-down menu.		It specifies the file containing the Diffie-Hellman parameters, generated using the `openssl dhparam` utility. It must be in PEM format.
Server verification*			Server verification options are mandatory if *Enable Verification* is set to *True*.
CA dir	Select the *CA File brick* representing your CA directory.		CA directory containing the trusted CA and CRL files.
Verify CRL		Off (false)	It 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 suitable and adequate for configuring the 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. If the configuration of the component is satisfactory, after clicking the *Validate* button, the user receives the 'Component Validation successful' notification.
5. Save the *Syslog TLS* configuration by clicking *Save*.

6.4.6. Files

The *Files* configuration element enables the administrator to upload any certificate files.

6.4.6.1. Configuring Files

Files can be configured from the *BRICK* main navigation item.

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

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:

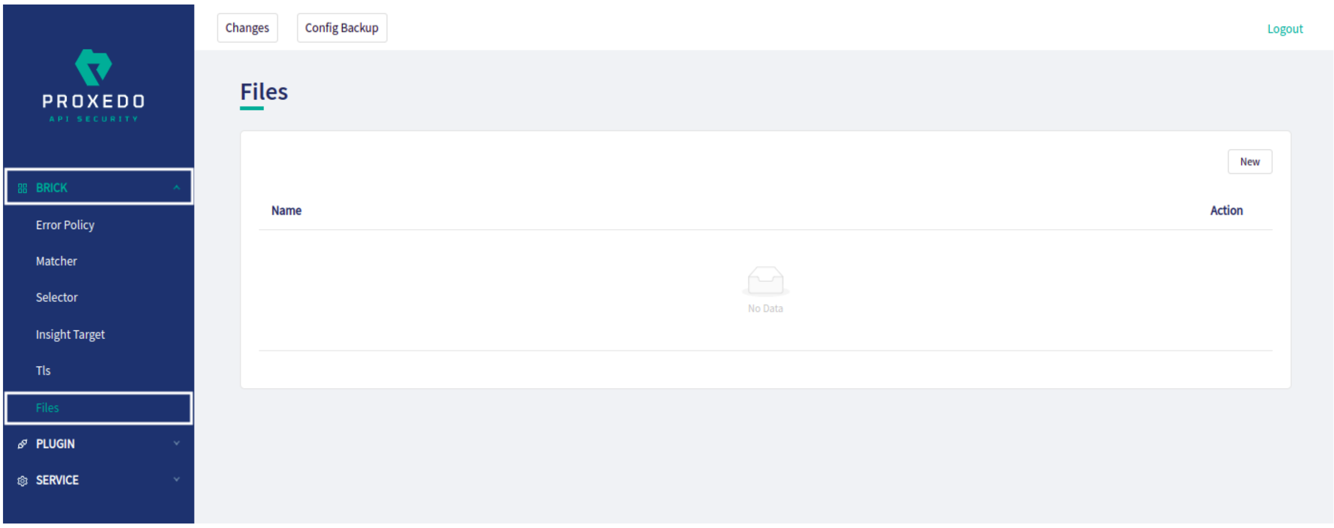


Figure 22. Files main page in the Web User Interface

3. Click on the *New* navigation button to configure Files.

Files contains the following settings:

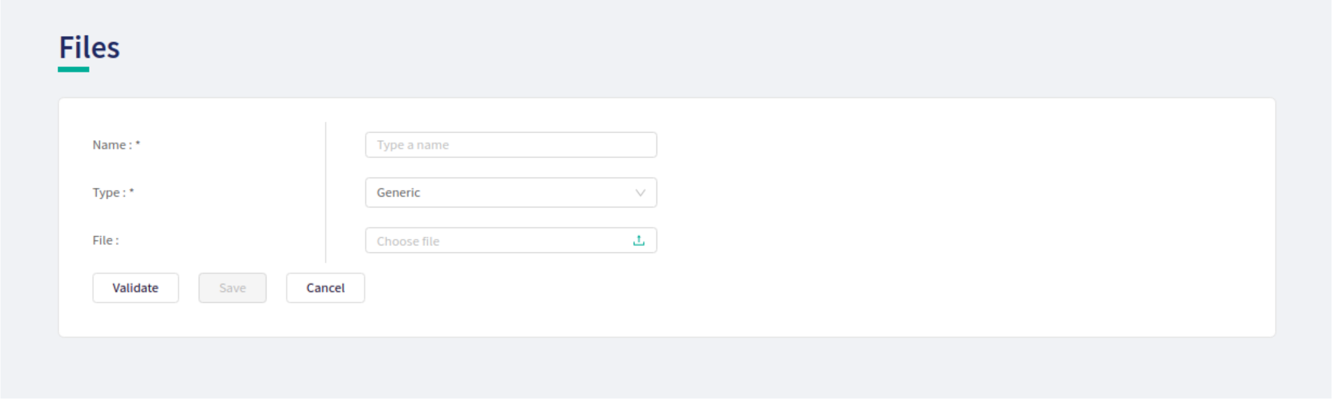


Figure 23. Configuring Files in the Web User Interface

Files has the following configuration parameters:

Table 36. Files configuration parameters

Key	Values	Default	Description
Name*	It is a mandatory value. The name can be provided in free text.		It defines the file-related configuration.

Key	Values	Default	Description
Type*	It is a mandatory value. The available values are: • Generic • Swagger • XSD • WSDL • CA • CRL See table Requirements for specific file types for specific requirements for each type.		The type selected here defines by which *PLUGIN* it can be used. The file uploaded here with the *Type Swagger*, for example, can be used by *Swagger* Plugins.
File*	It is a mandatory value. The required file can be uploaded here.		

4. Click the *Validate* button to check if the defined parameters are suitable and adequate for configuring the 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. If the configuration of the component is satisfactory, after clicking the *Validate* button, the user receives the 'Component Validation successful' notification.

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

Table 37. Requirements for specific file types

File type	Requirements
CA	1. It must be a flat ZIP file with the CA certificates inside. 2. It must contain not only the certificate files but also copies of them 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`. 3. It can contain CRL files, but then it also needs to contain the copies of them following the `<hash_of_the_related_ca_file.r0` format. The hash can be produced as described above.

6.4.7. Common configuration elements for *BRICKS*

6.4.7.1. 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 38. Extractor types

Key	Description
Method	It extracts the HTTP method of the request. It does not require configuration.
Status	It extracts the status code of the response. It does not require configuration.
JMESPath	It 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: main website: • There is a tutorial. • There are examples. • There is also a formal specification.
Header	It 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	It works like the Header extractor but it returns a list even if there is only a single extracted value.
Header first	It works like header extractor but it only returns the first extracted value even if there is a list of extracted values.
Headers	It extracts all the headers from the call. The results are stored as a dictionary, therefore it is recommended to omit the 'save as' key 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	It extracts the score value provided by the *Fraud Detector* plugin.
URI	It extracts the whole request URI as received from the client. It does not require configuration.

Key	Description
URI netloc	It 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	It 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	It extracts the *scheme* of the request (http or https). It does not require configuration.
URI username	It extracts the *username* in the request if present. It does not require configuration.
URI password	It extracts the *password* in the request if present. It does not require configuration.
URI host	It extracts the host in the request. It does not require configuration.
URI port	It 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	It 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	It 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	It extracts the query part of the URI as a string. It does not require configuration.
URI query	It extracts the query part of the URI. It does not require configuration.
URI query parameter	It 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 parameter force list	It works like *Uri query parameter* but it returns a list even if there is only a single extracted value.
URI query parameter first	It works like *Uri query parameter* but it only returns the **first** extracted value even if there is a list of extracted values.
Client_address	It extracts the client’s IP address.
Client_port	It extracts the client’s port (TCP).
Server_address	It extracts the server’s IP address.
Server_port	It extracts the server’s port (TCP).
Content	It extracts the content. It does not require configuration.
Raw content	It extracts the content as a string. It does not require configuration.
Content type	It extracts the content type from the HTTP header. It does not require configuration.
Content type charset	It extracts the charset from the content type HTTP header. It does not require configuration.
Call direction	It extracts the call direction (request, response). It does not require configuration.
Session ID	It 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.

Key	Description
Static	It extracts a string, integer, number, object, array, boolean as string from the configuration.
Timestamp	It extracts the current time. Also see the tables on Configuring timestamps and Timestamp format options.
Xpath	It 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	It works like *xpath* but it returns a list even if there is only a single extracted value.
Xpath first	It works like *xpath* but it only returns the first extracted value even if there is a list of extracted values.
Soap version	This extractor extends the xpath extractor with predefined expressions. It 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	This extractor extends the xpath extractor with predefined expressions. It extracts the soap envelope.
Soap header	It extracts the soap header. This extractor extends the xpath extractor with predefined expressions.
Soap body	It extracts the soap body. This extractor extends the xpath extractor with predefined expressions.
Soap fault	It extracts the soap fault. This extractor extends the xpath extractor with predefined expressions.

Key	Description
Soap fault code	It extracts the soap fault 'code'. This extractor extends the xpath extractor with predefined expressions. This extractor expression depends on the soap version. • **faultcode** - it is the SOAP v1.1 node tag • **Code** - it is the SOAP v1.2 node tag
Soap fault detail	This extractor extends the xpath extractor with predefined expressions. It extracts the soap fault 'detail'. This matcher expression depends on the soap version. • **Detail** - it is the SOAP v1.1 node tag • **Detail** - it is the SOAP v1.2 node tag
Soap 1.1 fault faultstring	This extractor extends the xpath extractor with predefined expressions. It extracts the soap fault 'faultstring'. This extractor only works with soap version 1.1.
Soap 1.1 fault faultactor	This extractor extends the xpath extractor with predefined expressions. It extracts the soap fault 'faultactor'. This extractor only works with soap version 1.1.
Soap 1.2 fault reason	This extractor extends the xpath extractor with predefined expressions. It extracts the soap fault 'Reason'. This extractor only works with soap version 1.2.
Soap 1.2 fault node	This extractor extends the xpath extractor with predefined expressions. It extracts the soap fault 'Node'. This extractor only works with soap version 1.2.
Soap 1.2 fault role	This extractor extends the xpath extractor with predefined expressions. It extracts the soap fault 'Role'. This extractor only works with soap version 1.2.

You can still use **Save as** 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 39. 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-DDTHH:mm:ss.SSSSSSZZ'	Set the format. See: Timestamp format options

Table 40. 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

Name	Token	Output
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 41. Xpath extractor configuration options

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

6.4.7.2. Comparators

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

Table 42. Types of comparators

Key	Description	Parameters
Equals	It 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 ignorcase field.
Not equals	It 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 ignorcase field.
Starts with	It 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	It matches if the value ends exactly with the pattern.	Ignore case: Case differences are ignored. When the present **VaLuE** would match **given_value**.

Key	Description	Parameters
Substring	It 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*. ℹ 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	It matches if the pattern is larger or equal to the value.	
Maximum	It matches if the pattern is smaller or equal to the value.	
Range	It matches if the value is between the limits in the pattern, including boundaries. The format of the pattern must be minimum..maximum.	
Status class	Status class is a special matcher for conveniently matching HTTP status code classes. 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).	

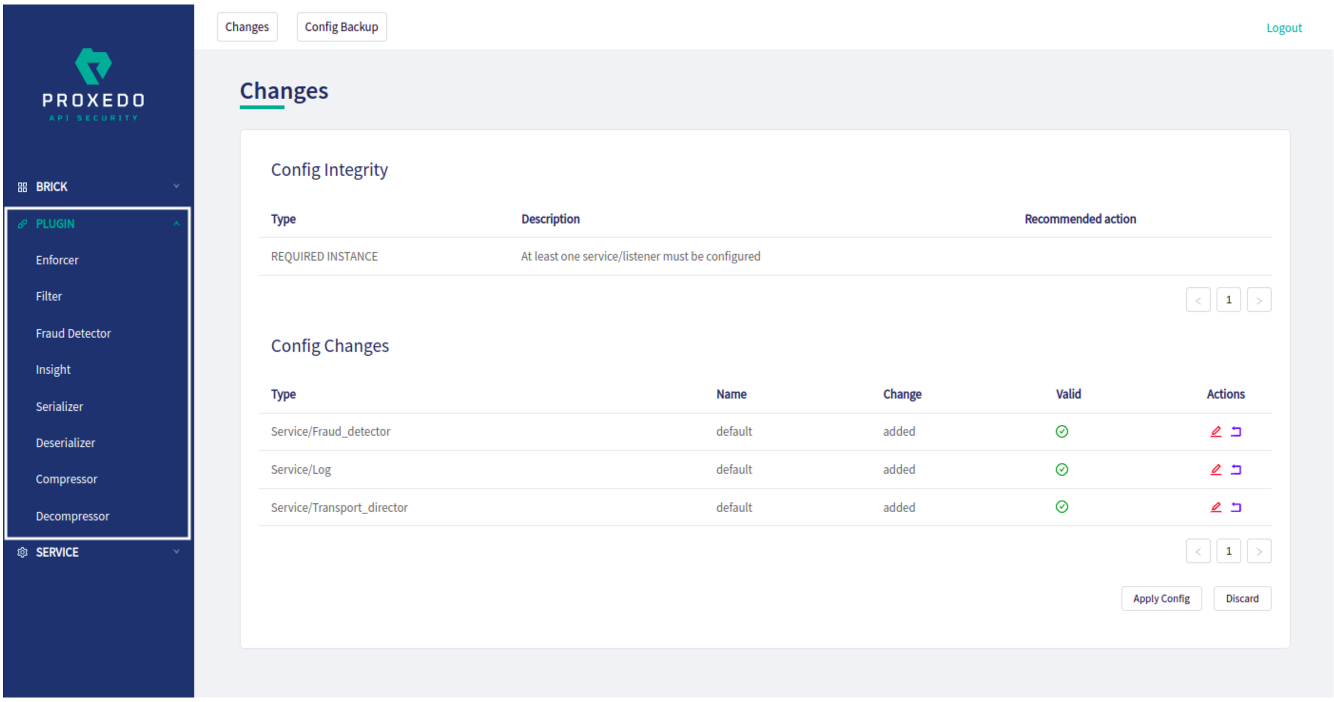
Table 43. Checking status code range

Pattern	Status code range	Description
info	1xx	Informational response
success	2xx	Successful response

Pattern	Status code range	Description
redirect	3xx	Redirects
client_error	4xx	Client Errors
server_error	5xx	Server Errors

6.5. PLUGIN - 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:



Changes

Config Integrity

Type	Description	Recommended action
REQUIRED INSTANCE	At least one service/listener must be configured	

Config Changes

Type	Name	Change	Valid	Actions
Service/Fraud_detector	default	added	✓	
Service/Log	default	added	✓	
Service/Transport_director	default	added	✓	

Figure 24. The PLUGIN's 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 44. Default objects - PLUGIN

Default object name	Key
default_json	Serializer
default_xml	Serializer
default_json	Deserializer

Default object name	Key
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 45. Plugins' common parameters

Key	Values	Default value	Description
Matcher	The Matchers configured under the *BRICK* main configuration unit are listed here and can be selected from the drop-down list.	Always: If the value is not defined, the plugin is always executed.	It is an 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.
Error policy	The Error Policy configured under the *BRICK* navigation item are listed here can be selected from the drop-down list.		It is an 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 OpenAPI2.0 (Swagger) schemas, XSD schemas or WSDL schema.

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
- XSD 1.1 Specification
- XSD Tutorial
- WSDL Tutorial
- WSDL 1.1 Specification
- WSDL 1.2 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 46. 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.

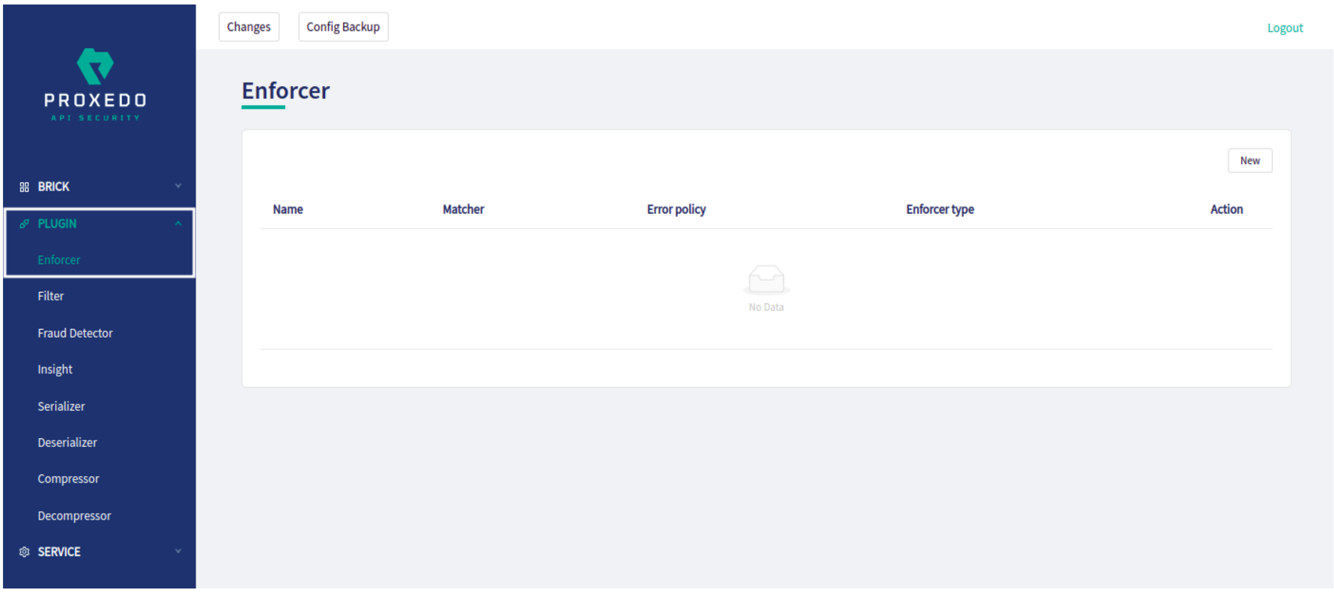
See Error Policy to understand how defaults are applied.

6.5.2.1. Configuring Enforcer Plugins

Enforcer plugins can be configured from the *PLUGIN* main navigation item.

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

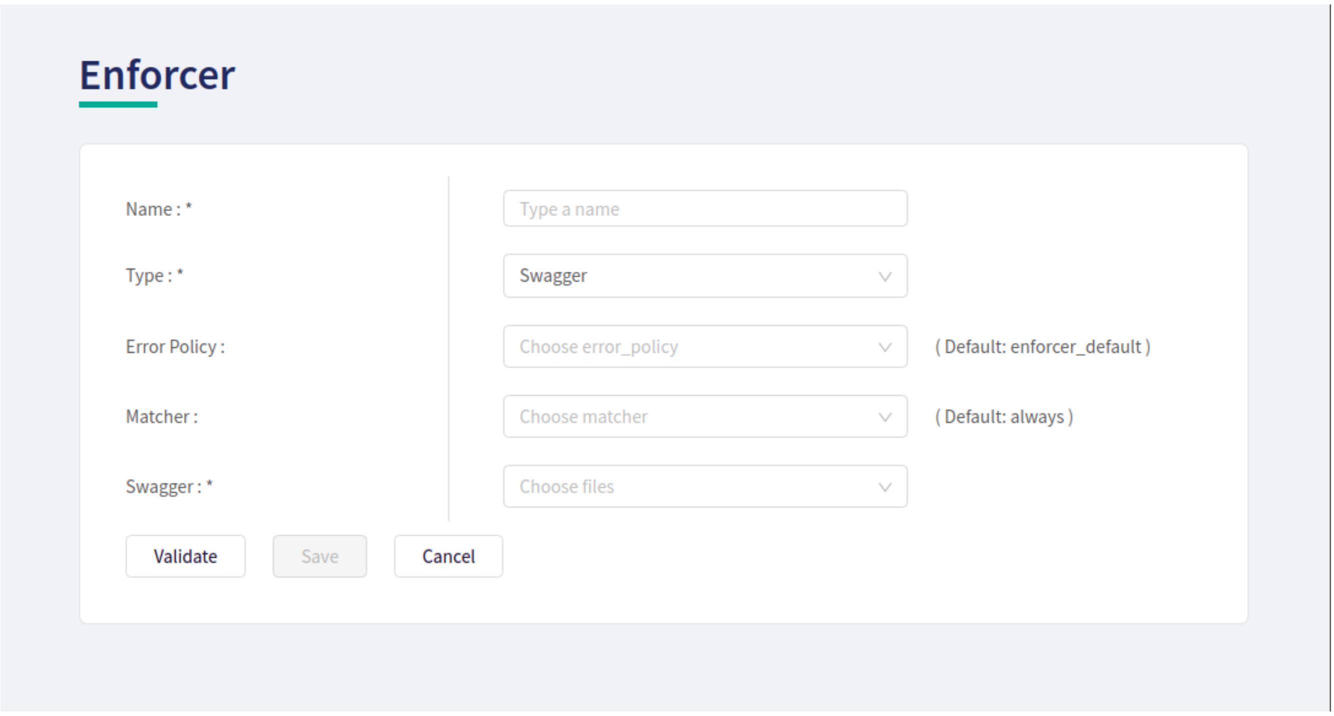
In the configuration window that appears, you can either see the empty parameter values that can be configured for the actual component or you can see already configured component(s) and their parameters. The already configured components with defined parameters can be default components available in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



Name	Matcher	Error policy	Enforcer type	Action

Figure 25. Enforcer Plugin's main page in the Web User Interface

The following values can be configured for the Filter Plugin:



Enforcer

Name : * Type a name

Type : * Swagger

Error Policy : Choose error_policy (Default: enforcer_default)

Matcher : Choose matcher (Default: always)

Swagger : * Choose files

Figure 26. Configuring an enforcer plugin in the Web User Interface

The *Enforcer Plugin* accepts the following configuration options:

Table 47. Enforcer Plugin's configuration options

Key	Values	Default value	Description
Name*	It is a mandatory value. It can be defined in free text.		This name identifies the Enforcer Plugin. The name of the plugin can be referenced from other parts of the configuration.
Type*	It is a mandatory value. It can be selected from the drop-down list. The available values are: • Swagger • XSD • WSDL		This identifies the type of the *Enforcer* plugin.
Error policy	The error policies configured under *BRICK - Configuration units* are listed here and can be selected from the drop-down list.	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.
Matcher	The matchers configured under *BRICK - Configuration units* are listed here and can be selected from the drop-down list.	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.

Key	Values	Default value	Description
Swagger*/WSDL*/Operations*	Depending on which type of the component was selected above, the following values are available: • The Swaggers defined under Files are listed here and can be selected from the drop-down list. • The WSDL files defined under Files are listed here and can be selected from the drop-down list. • XSD enforcer plugin 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 OpenApi2.0 schemas. WSDL enforcer *Plugin* validates against WSDL 1.0-1.1 schemas. XSD enforcer *Plugin* validates against XSD schemas.

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

Table 48. XSD enforcer plugin configuration options for Operations

Key	Default	Description
uri_path	*	It defines the pattern for uri_path.

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

3. Name the *Enforcer* Plugin.
4. Choose the type of the *Enforcer* plugin.
5. Choose an *Error policy* from the drop-down list. The drop-down list will offer the error policy options configured under *BRICK*.
6. Choose a *Matcher* from the drop-down list. The drop-down list will offer the matcher options configured under *BRICK*.
7. Depending on the choice of the *Enforcer plugin* type selected earlier, different fields appear here for further configuration:
   - Swagger - Upload the Swagger file if the Enforcer type selected at *Type* field was Swagger.
   - WSDL - Upload the WSDL file if the Enforcer type selected earlier was WSDL.
   - Operations - Fill in the *Operations* fields according to XSD enforcer plugin configuration options for Operations if the Enforcer type selected earlier was XSD.
8. Click the *Validate* button to check if the defined parameters are suitable and adequate for configuring the 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. If the configuration of the component is satisfactory, after clicking the *Validate* button, the user receives the 'Component Validation successful' notification.
9. Click the *Save* button, when all required configuration fields have been defined.

6.5.2.2. Swagger

The Swagger enforcer *Plugin* validates against OpenApi2.0 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. XSD

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

> ℹ 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.

6.5.2.4. WSDL

WSDL enforcer *Plugin* validates against WSDL 1.0-1.1 schemas.

> ℹ 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 49. 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.

> ℹ 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.

6.5.3. Filter

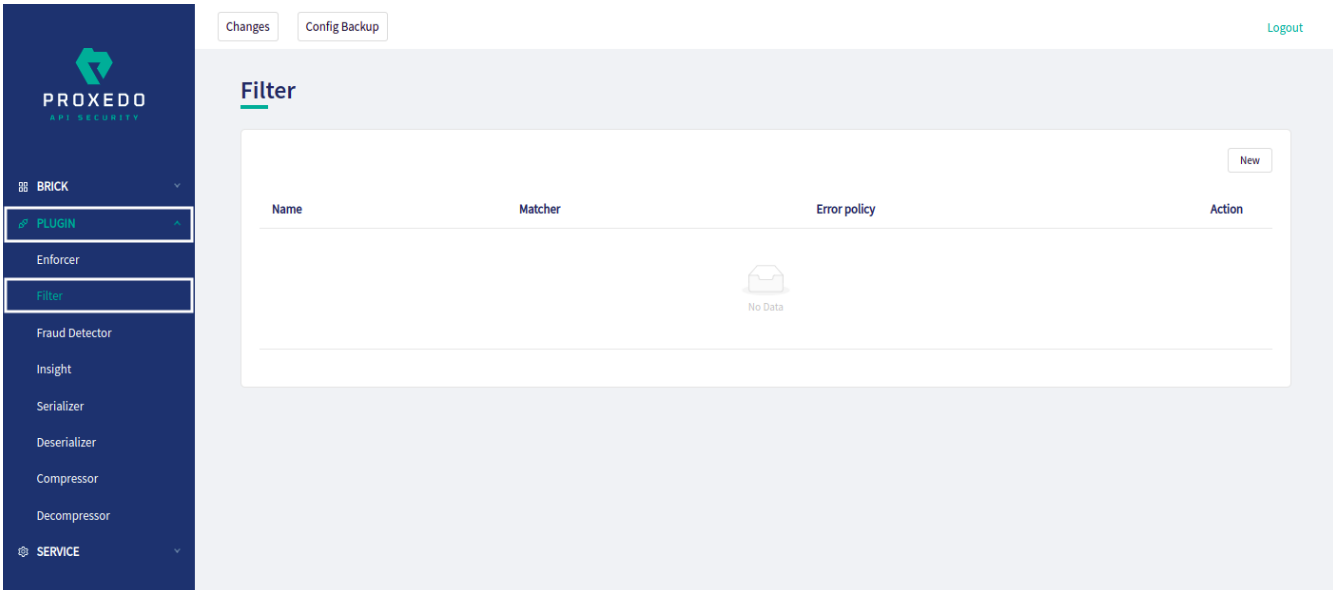
Filter Plugins are lightweight alternatives of *Enforcer Plugins* for filtering unwanted traffic. They only consist of a matcher and an error policy. If the matcher matches, the error policy is applied. This way you can use matchers inline, instead of creating a whole schema-based *Enforcer Plugin* for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

6.5.3.1. Configuring Filter Plugins

The Filter Plugin can be configured under the *PLUGIN* main navigation unit.

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

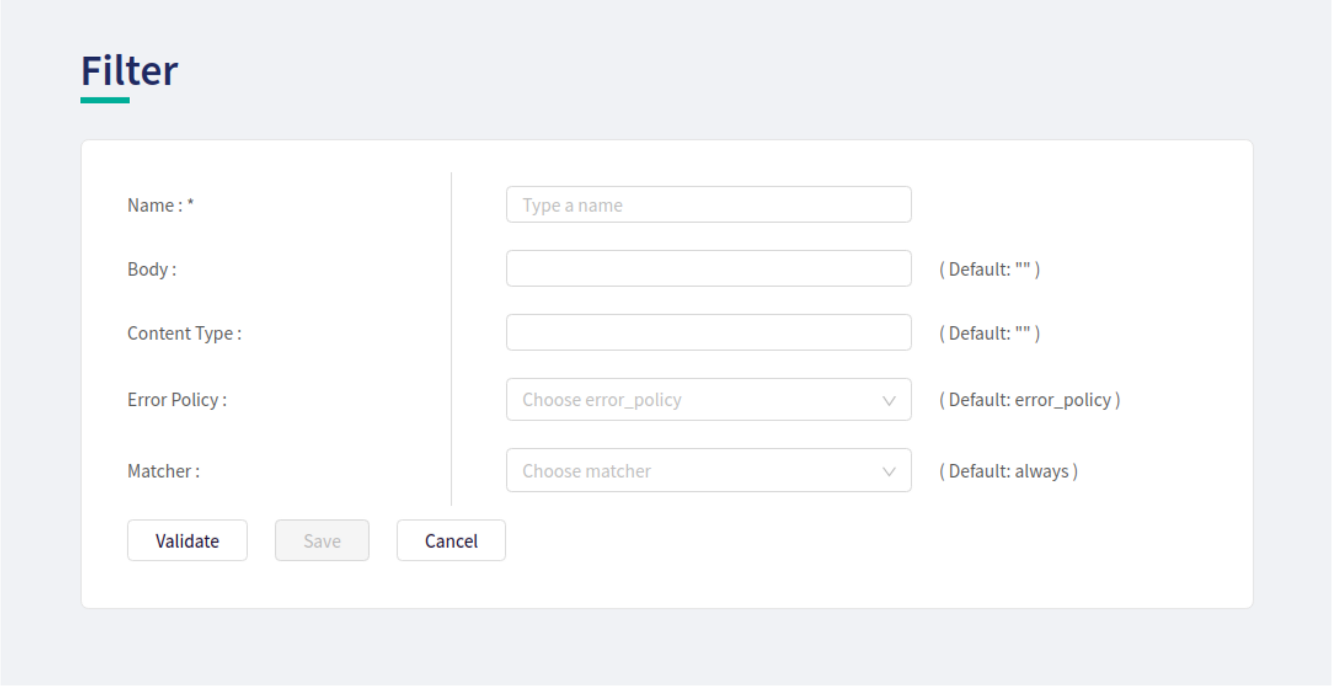
In the configuration window that appears, you can either see the empty parameter values that can be configured for the actual component or you can see already configured component(s) and their parameters. The already configured components with defined parameters can be default components available in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



Name	Matcher	Error policy	Action

Figure 27. Filter Plugin's main page in the Web User Interface

The following values can be configured for the Filter Plugin:



Filter

Name : * Type a name

Body : (Default: "")

Content Type : (Default: "")

Error Policy : Choose error_policy (Default: error_policy)

Matcher : Choose matcher (Default: always)

Figure 28. Configuring a filter plugin in the Web User Interface

The *Filter Plugin* accepts the following configuration options:

Table 50. Filter Plugin's configuration options

Key	Values	Default value	Description
Name*	It is a mandatory value. It can be defined in free text.		The name identifying the Filter Plugin. This name of the plugin can be referenced from other parts of the configuration.
Body	It can be defined in free text.		It is the body of the message sent in case an error policy is applied.
Content Type			This field defines the content type of HTTP error request sent, if the filter stops the call. It can be referenced by its name.
Error policy	The error policies configured under *BRICK - Configuration units* are listed here and can be selected from the drop-down list.	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.
Matcher	The matchers configured under *BRICK - Configuration units* are listed here and can be selected from the drop-down list.	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.

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. Add the name of the Filter Plugin.
4. Add the Body content for the error policy. (Optional)
5. Define the Content type.
6. Choose an error policy from the drop-down list. (Optional)
7. Choose a matcher from the drop-down list. (Optional)
8. Click the *Validate* button to check if the defined parameters are suitable and adequate for configuring the 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. If the configuration of the component is satisfactory, after clicking the *Validate* button, the user receives the 'Component Validation successful' notification.
9. Click the *Save* button, when all required configuration fields have been defined.

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.

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

The Fraud Detector Plugin can be configured under the *PLUGIN* main navigation unit.

1. Click on the *PLUGIN* main configuration item in the Left navigation area. Alternatively you can also click on the ⌄ sign to open up the sub-navigation items of *PLUGIN*.
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:

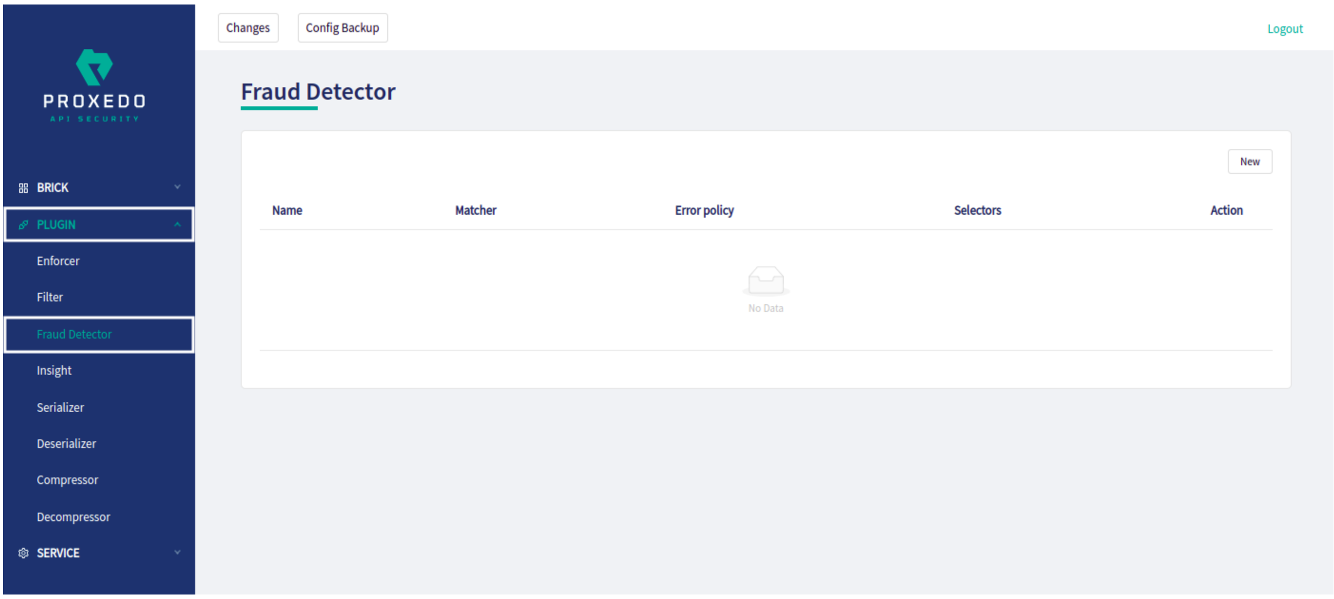


Figure 29. Fraud Detector's main page in the Web User Interface

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

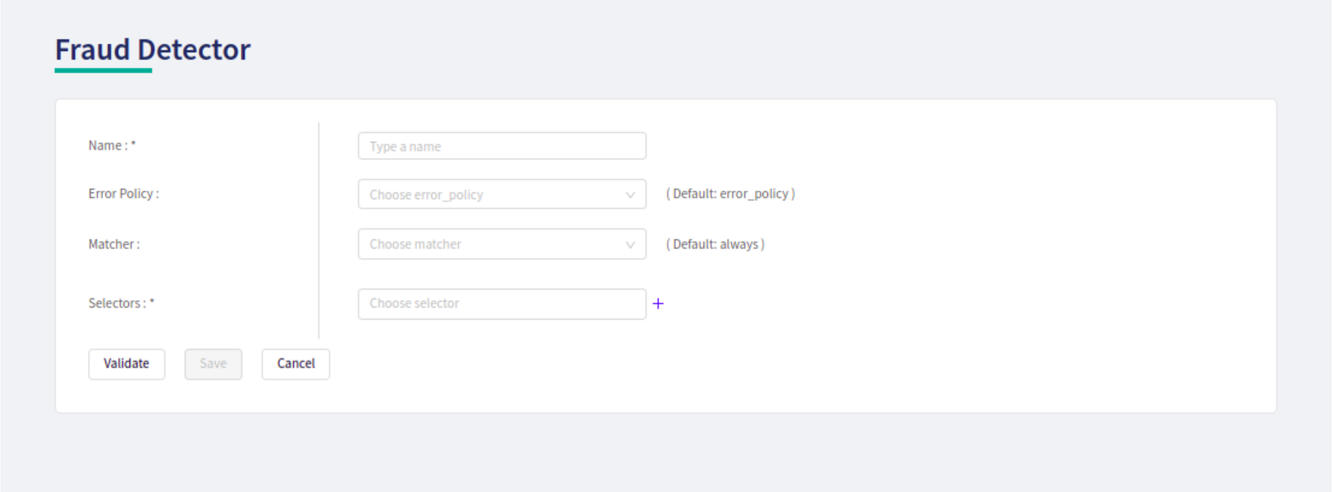


Figure 30. Configuring the Fraud Detector plugin in the Web User Interface

The *Fraud Detector Plugin* accepts the following configuration options:

Table 51. Fraud Detector Plugin's configuration options

Key	Values	Default value	Description
Name*	It is a mandatory value. It can be defined in free text.		It is the name identifying the Fraud Detector. This name of the plugin can be referenced from other parts of the configuration.
Error Policy	The error policies configured under *BRICK - Configuration units* are listed here and can be selected from the drop-down list.	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
Matcher	The matchers configured under *BRICK - Configuration units* are listed here and can be selected from the drop-down list.	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.
Selectors*			A list of Selectors that collect information from the call. They can be referenced by their name or can be defined inline. Fraud Detector plugin takes the following selector types into account: • IP address • email address • phone number ℹ There is a default matcher for the Client's IP address, namely the Client_address matcher. As there are no defaut matchers for the other two data type, use 'phone' name in the 'Save as' field for the phone number data and 'email' for the email address. ℹ It is possible to add more types of data from the selectors to the Fraud Detector Plugin using custom fields, apart from the above recommended three data types. In such cases contact the Balasys Support team.

⚠ 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. Add the name of the Fraud Detector.
4. Choose an error policy from the drop-down list. (Optional)
5. Choose a matcher from the drop-down list. (Optional)
6. Choose a *Selector* from the drop-down list. When it is selected click on the plus sign to add it to the configuration.
7. Click the *Validate* button to check if the defined parameters are suitable and adequate for configuring the

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. If the configuration of the component is satisfactory, after clicking the *Validate* button, the user receives the 'Component Validation successful' notification.

8. Click the *Save* button, when all required configuration fields have been defined.

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

6.5.5. Insight

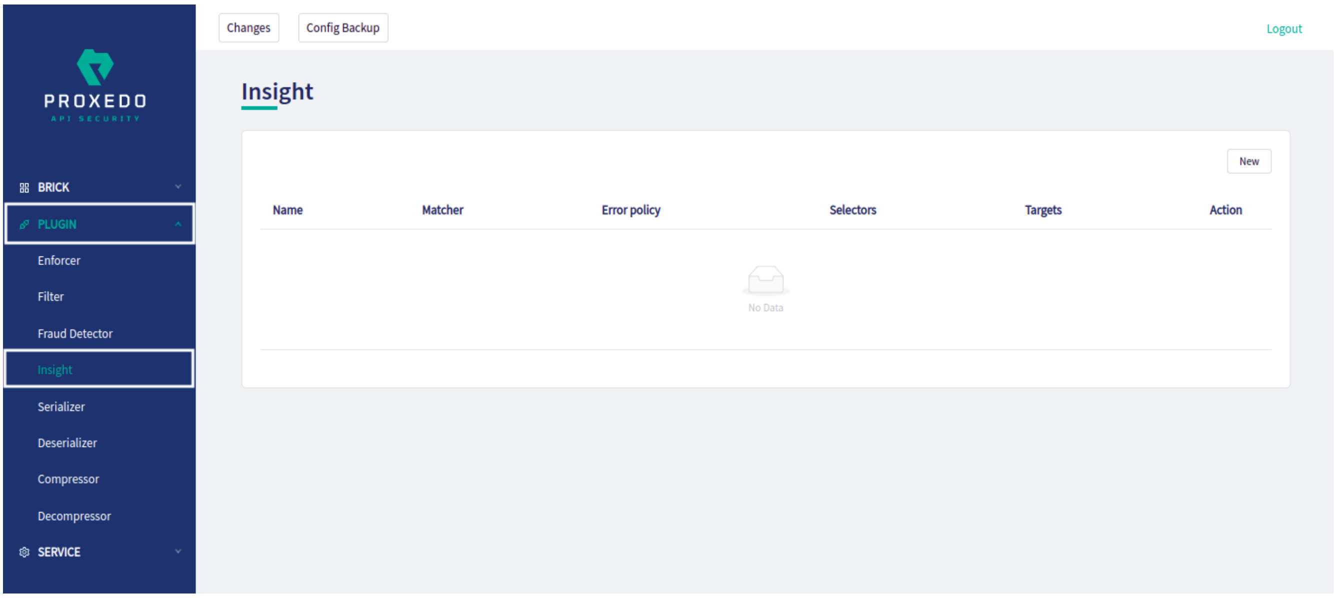
It is a Plugin that extracts various data from the call and sends it to external systems (log servers, SIEMs, and other data analysis tools).

6.5.5.1. Configuring Insight Plugins

The Insight Plugin can be configured under the *PLUGIN* main navigation unit.

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

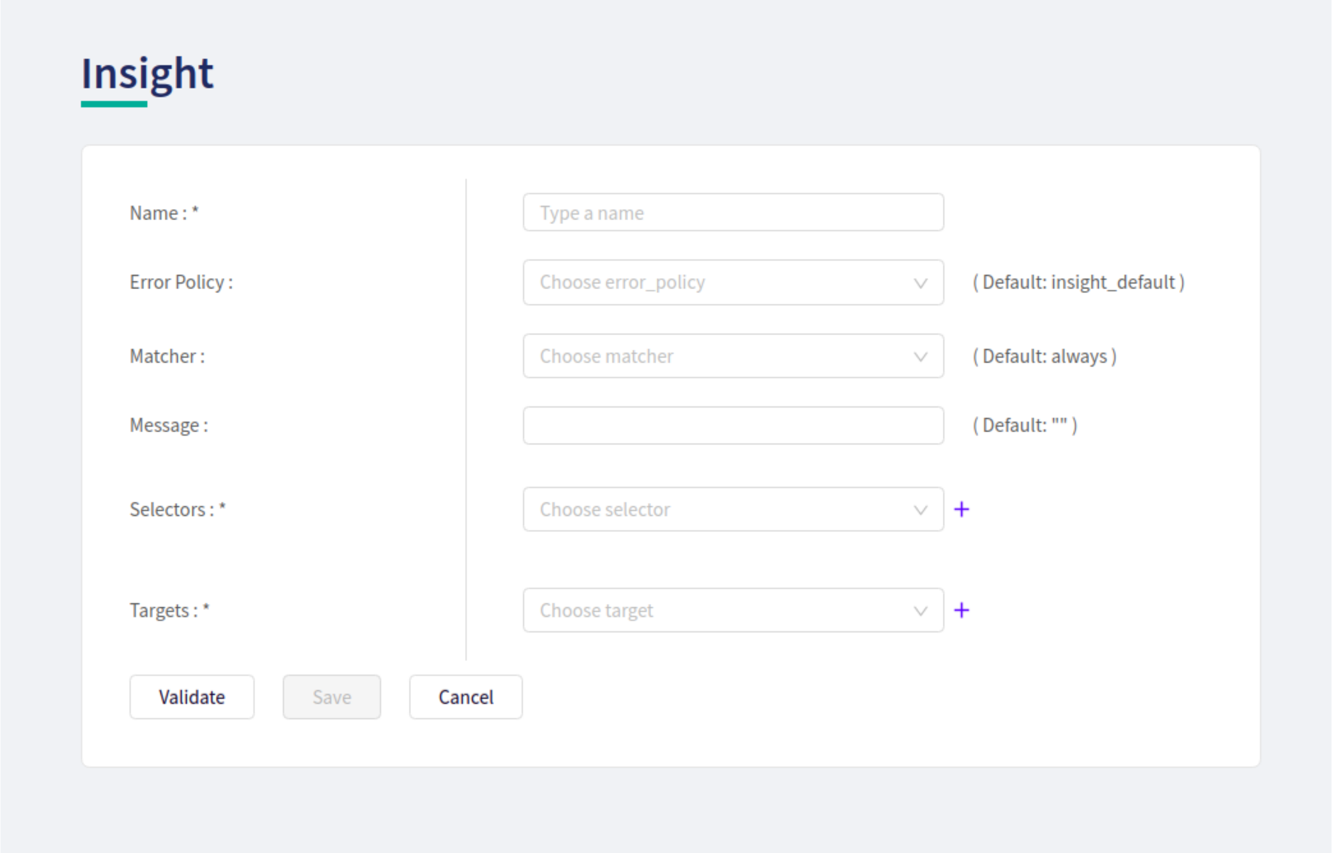
In the configuration window that appears, you can either see the empty parameter values that can be configured for the actual component or you can see already configured component(s) and their parameters. The already configured components with defined parameters can be default components available in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



Name	Matcher	Error policy	Selectors	Targets	Action

Figure 31. Insight Plugin's main page in the Web User Interface

3. Click the *New* button to create an Insight Plugin configuration. The following values can be configured for the Insight Plugin:



Insight

Name : *	Type a name	
Error Policy :	Choose error_policy	(Default: insight_default)
Matcher :	Choose matcher	(Default: always)
Message :		(Default: "")
Selectors : *	Choose selector	+
Targets : *	Choose target	+

Figure 32. Configuring an insight plugin in the Web User Interface

Table 52. Insight Plugin's configuration options

Key	Values	Default value	Description
Name*	It is a mandatory value. It can be defined in free text.		The name identifying the insight. This name of the insight can be referenced from other parts of the configuration.
Error policy	The error policies configured under *BRICK - Configuration units* are listed here and can be selected from the drop-down list.	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.
Matcher	The matchers configured under *BRICK - Configuration units* are listed here and can be selected from the drop-down list.	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.
Message	It can be defined in free text.	""	It is the message part of the log message.
Selectors*			A list of Selectors is provided here that collect information from the call. They can be referenced by their name or can be defined inline. 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.

Key	Values	Default value	Description
Targets*			A list of Insight Targets where the collected information will be sent to.

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 53. 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 suitable and adequate for configuring the 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. If the configuration of the component is satisfactory, after clicking the *Validate* button, the user receives the 'Component Validation successful' notification.
11. Click the *Save* button, when all required configuration fields have been defined.

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.

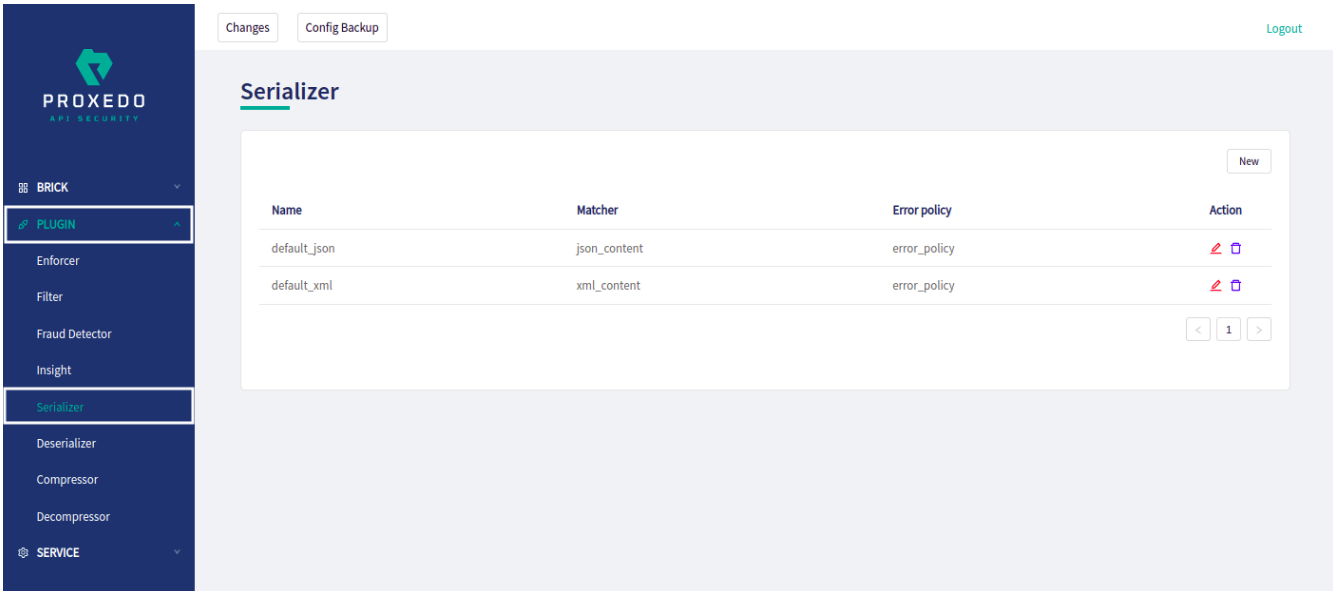
The Serializer Plugin understands the Content-Type HTTP header and can work with JSON and XML content.

6.5.6.1. Configuring Serializer Plugins

The Serializer can be configured under the *PLUGIN* main navigation unit.

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

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

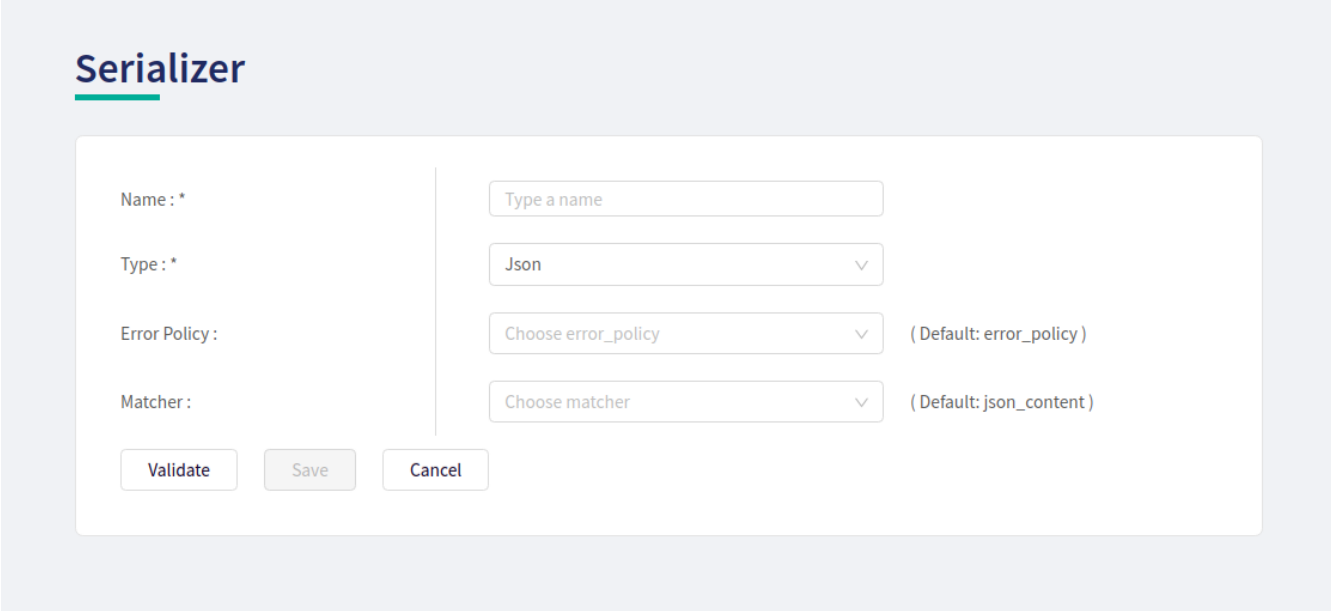


Serializer

Name	Matcher	Error policy	Action
default_json	json_content	error_policy	✎ 🗑
default_xml	xml_content	error_policy	✎ 🗑

Figure 33. The serializer main page in the Web User Interface

3. Click the *New* button to create a serializer configuration. The following values can be configured for the Serializer Plugin:



Serializer

Name : *	Type a name	
Type : *	Json	
Error Policy :	Choose error_policy	(Default: error_policy)
Matcher :	Choose matcher	(Default: json_content)

Figure 34. Configuring a serializer in the Web User Interface

The table describes some more details on the serializer configuration parameters.

Table 54. Serializers' configuration options

Key	Values	Default value	Description
Name*	It is a mandatory value. It can be defined in free text.		It is 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*	It is a mandatory value. 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.
Error policy	The Error Policy configured under the *BRICK* navigation item are listed here can be selected from the drop-down list.	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.
Matcher	The Matchers configured under the *BRICK* main configuration unit are listed here and can be selected from the drop-down list.	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.

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 suitable and adequate for configuring the 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. If the configuration of the component is satisfactory, after clicking the *Validate* button, the user receives the 'Component Validation successful' notification.
9. Click the *Save* button, when all required configuration fields have been defined.

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.

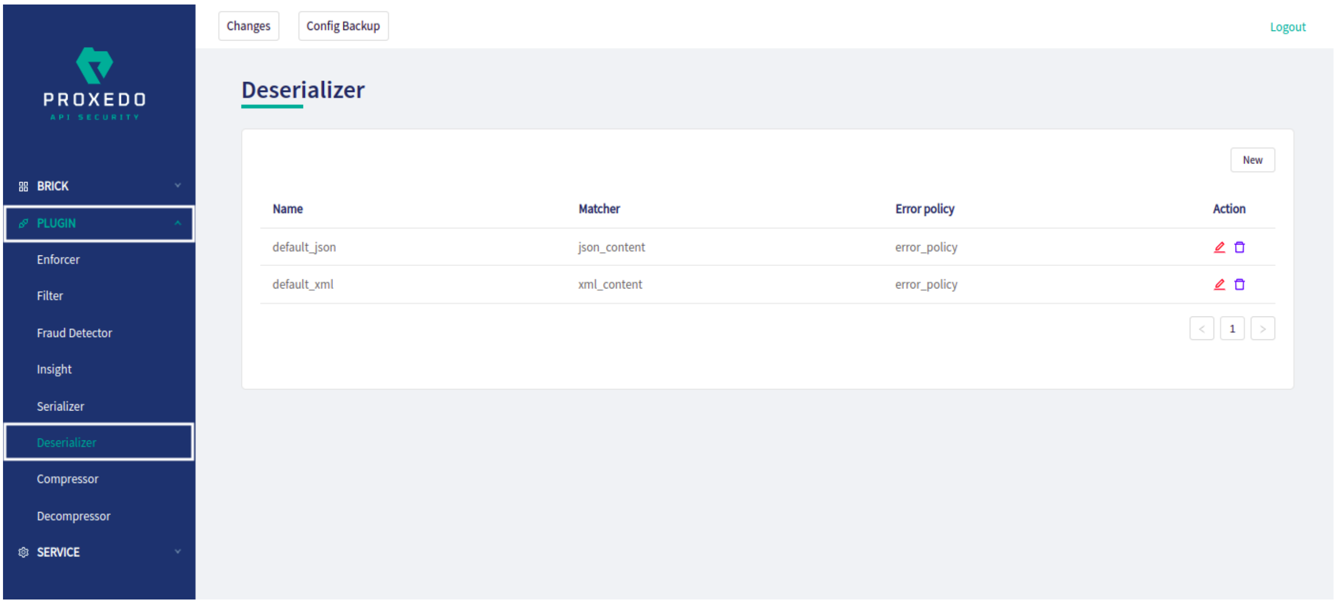
A typical Security Flow configuration starts with a *Decompressor* followed by a *Deserializer* and finishes with a *Serializer* followed by a *Compressor*. This ensures that transferred HTTP bodies are syntactically correct and that they are reconstructed to avoid transferring potentially crafted content.

6.5.7.1. Configuring Deserializer Plugins

The Deserializer can be configured under the *PLUGIN* main navigation unit.

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

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



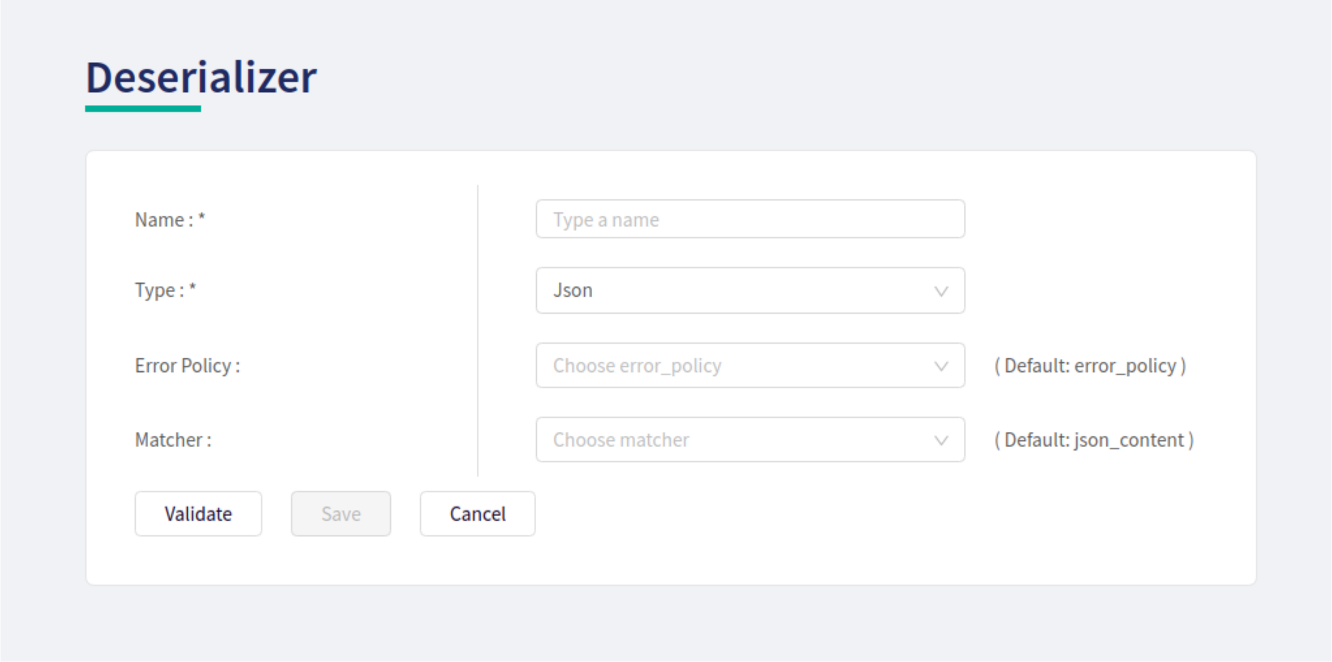
Deserializer

Name	Matcher	Error policy	Action
default_json	json_content	error_policy	✎ 🗑
default_xml	xml_content	error_policy	✎ 🗑

Figure 35. The deserializer's main page in the Web User Interface

2. Click the *New* navigation button to create a deserializer configuration.

The following values can be configured for the Deserializer Plugin:



Deserializer

Name : * Type a name

Type : * Json

Error Policy : Choose error_policy (Default: error_policy)

Matcher : Choose matcher (Default: json_content)

Figure 36. Configuring a deserializer in the Web User Interface

The following table describes the deserializer configuration parameters in details:

Table 55. Deserializers' configuration options

Key	Values	Default value	Description
Name*	It is a mandatory value. It can be defined in free text.		The name identifying the deserializer. This name of the deserializer can be referenced from other parts of the configuration.
Type*	It is a mandatory value. 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.
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.

Key	Values	Default value	Description
Error policy	The error policies configured under *BRICK - Configuration units* are listed here can be selected from the drop-down list.	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.
Matcher	The matchers configured under *BRICK - Configuration units* are listed here can be selected from the drop-down list.	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.

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 suitable and adequate for configuring the 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. If the configuration of the component is satisfactory, after clicking the *Validate* button, the user receives the 'Component Validation successful' notification.
8. Click the *Save* button, when all required configuration fields have been defined.

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 Compressors

The Compressor can be configured under the *PLUGIN* main navigation unit.

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

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

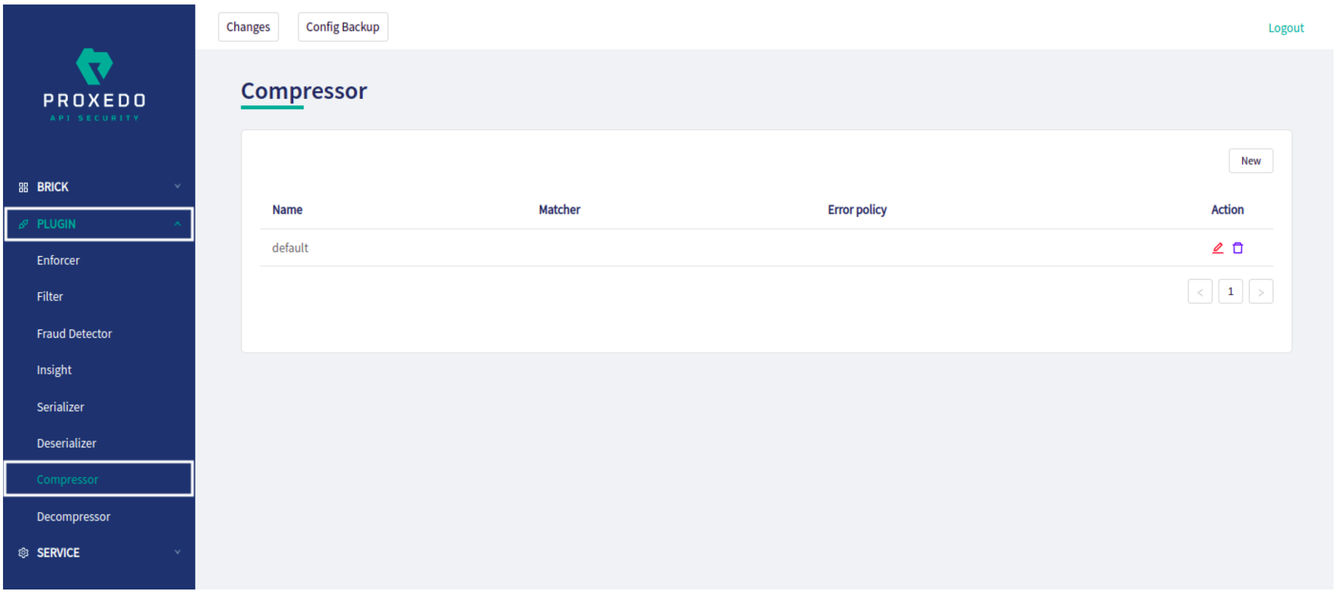


Figure 37. The compressor main page in the Web User Interface

2. Click the *New* button to create a Compressor configuration. The following values can be configured for the Compressor Plugin:

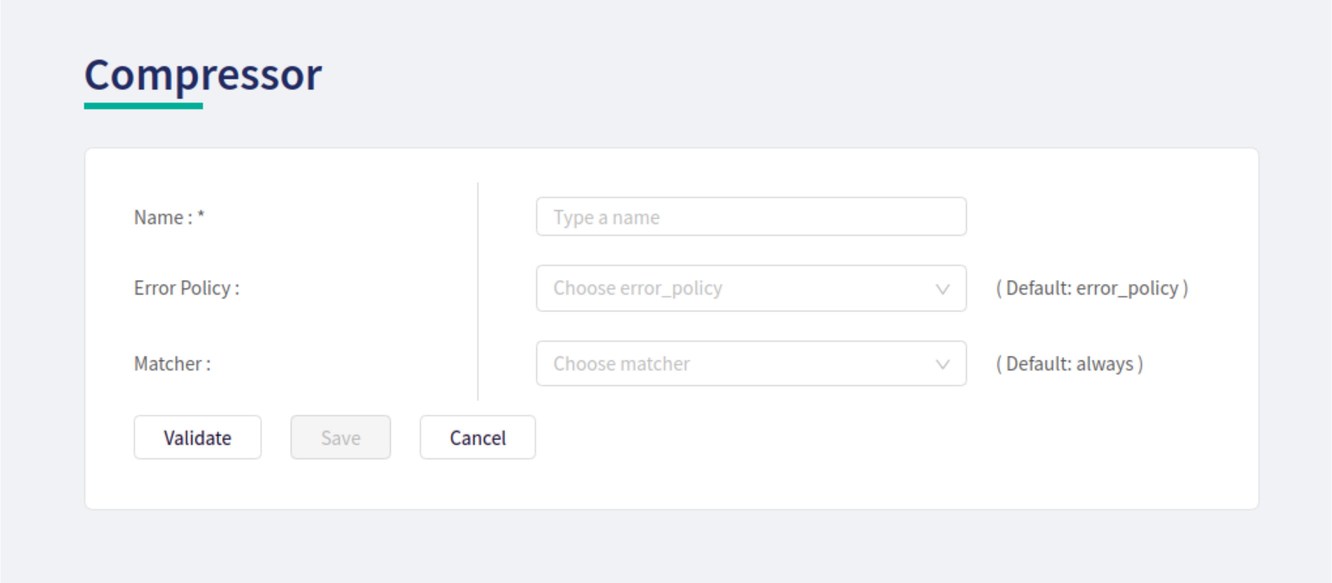


Figure 38. Configuring a compressor in the Web User Interface

The table describes some more details on the Compressor's configuration parameters.

Table 56. The Compressors' configuration options

Key	Values	Default value	Description
Name*	It is a mandatory value. It can be defined in free text.		It is 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.
Error policy	The Error Policy configured under the *BRICK* navigation item are listed here can be selected from the drop-down list.	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.

Key	Values	Default value	Description
Matcher	The Matchers configured under the *BRICK* main configuration unit are listed here and can be selected from the drop-down list.	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.

Continue configuring the compressor with the following steps:

3. Add the name of the compressor.
4. Choose an Error policy from the drop-down list.
5. Choose a Matcher from the drop-down list.
6. Click the *Validate* button to check if the defined parameters are suitable and adequate for configuring the 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. If the configuration of the component is satisfactory, after clicking the *Validate* button, the user receives the 'Component Validation successful' notification.
7. Click the *Save* button, when all required configuration fields have been defined.

6.5.9. Decompressor

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

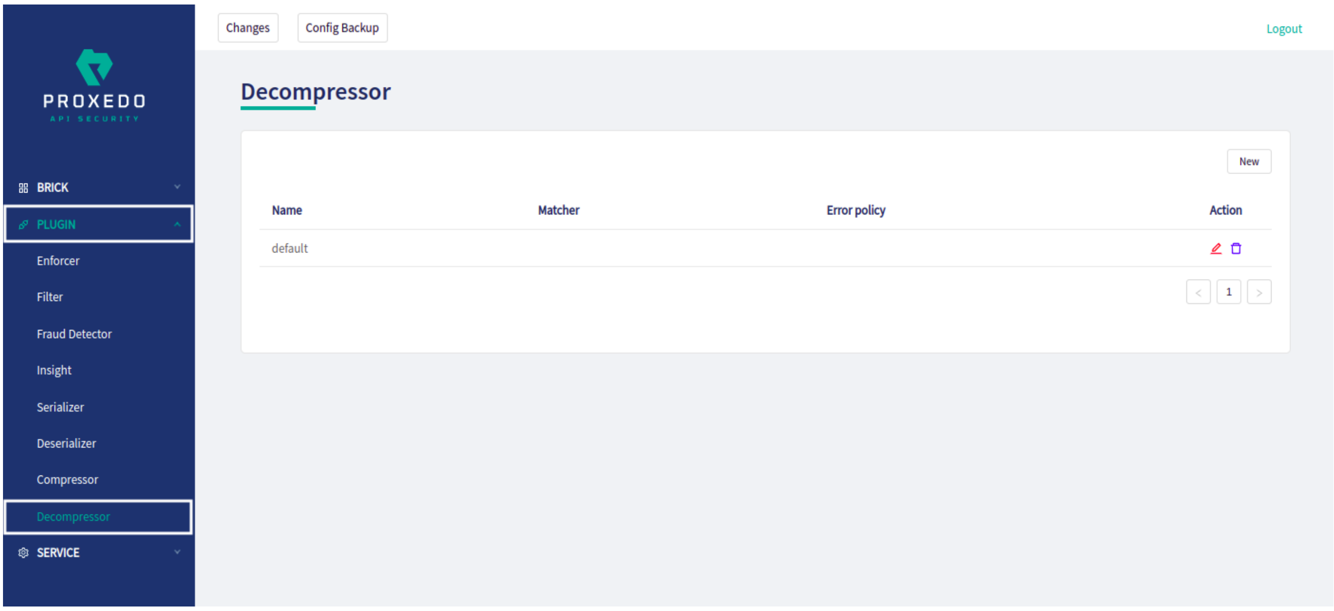
Decompressors understand the *Transfer-Encoding* HTTP header and can work with content optionally compressed by the *gzip*, *deflate* and *brotli* algorithms.

6.5.9.1. Configuring Decompressors

The Decompressor can be configured under the *PLUGIN* main navigation unit.

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

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



Name	Matcher	Error policy	Action
default			

Figure 39. The Decompressor’s main page in the Web User Interface

2. Click the *New* button to create a Decompressor configuration. The following values can be configured for the Decompressor Plugin:

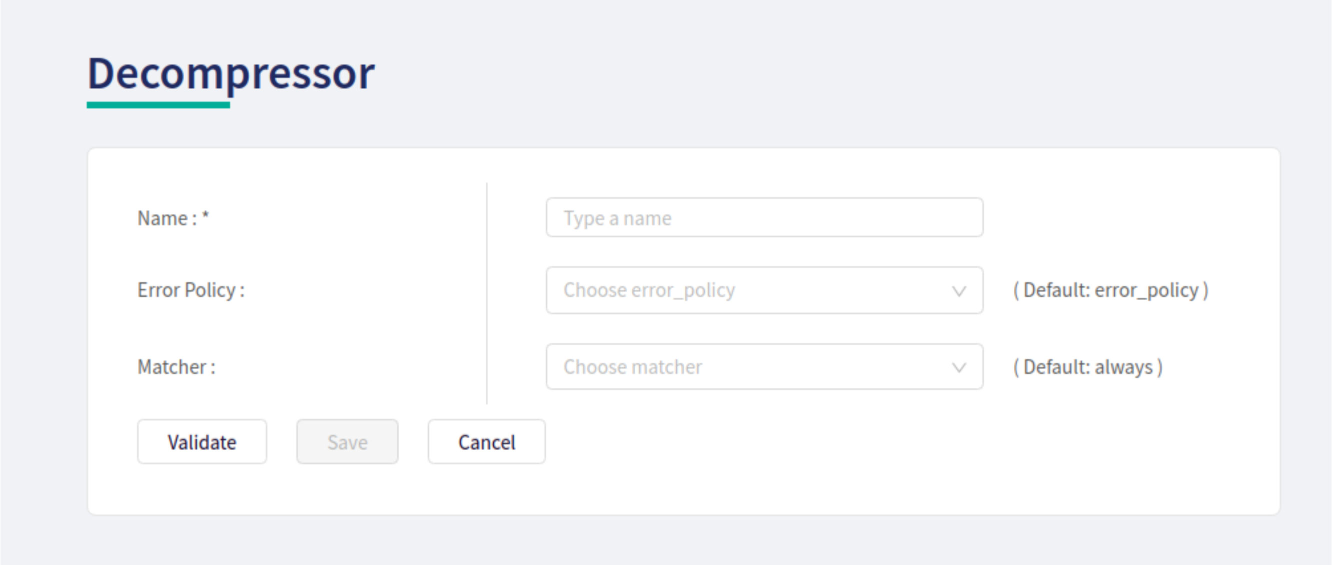


Figure 40. Configuring a decompressor in the Web User Interface

The table describes some more details on the Decompressor’s configuration parameters.

Table 57. The Decompressors' configuration options

Key	Values	Default value	Description
Name*	It is a mandatory value. It can be defined in free text.		It is 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.

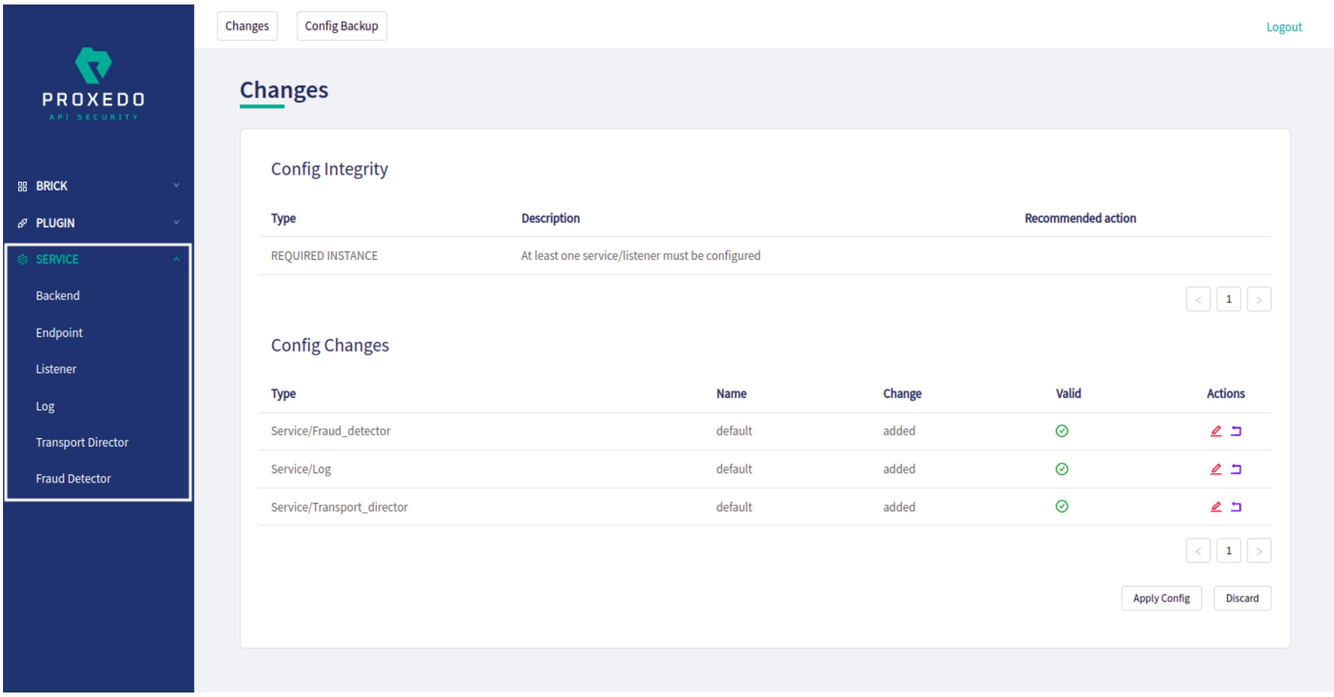
Key	Values	Default value	Description
Error policy	The Error Policy configured under the *BRICK* main configuration unit are listed here can be selected from the drop-down list.	The Plugin has a default error policy.	It defines a custom error policy to be applied if the Plugin reports an error. The settings 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.
Matcher	The Matchers configured under the *BRICK* main configuration unit are listed here and can be selected from the drop-down list.	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.

Continue configuring the decompressor with the following steps:

3. Add the name of the decompressor.
4. Choose an Error policy from the drop-down list.
5. Choose a Matcher from the drop-down list.
6. Click the *Validate* button to check if the defined parameters are suitable and adequate for configuring the 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. If the configuration of the component is satisfactory, after clicking the *Validate* button, the user receives the 'Component Validation successful' notification.
7. Click the *Save* button, when all required configuration fields have been defined.

6.6. SERVICE - Configuration units

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



Changes

Config Integrity

Type	Description	Recommended action
REQUIRED INSTANCE	At least one service/listener must be configured	

Config Changes

Type	Name	Change	Valid	Actions
Service/Fraud_detector	default	added	✓	
Service/Log	default	added	✓	
Service/Transport_director	default	added	✓	

Figure 41. The SERVICE 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 the Backend

Backend can be configured under the **SERVICE** main navigation item.

1. Click on the *SERVICE* main configuration item in the Left navigation area. Alternatively you can also click on the ⌄ sign to open up the sub-navigation items of *SERVICE*.
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:

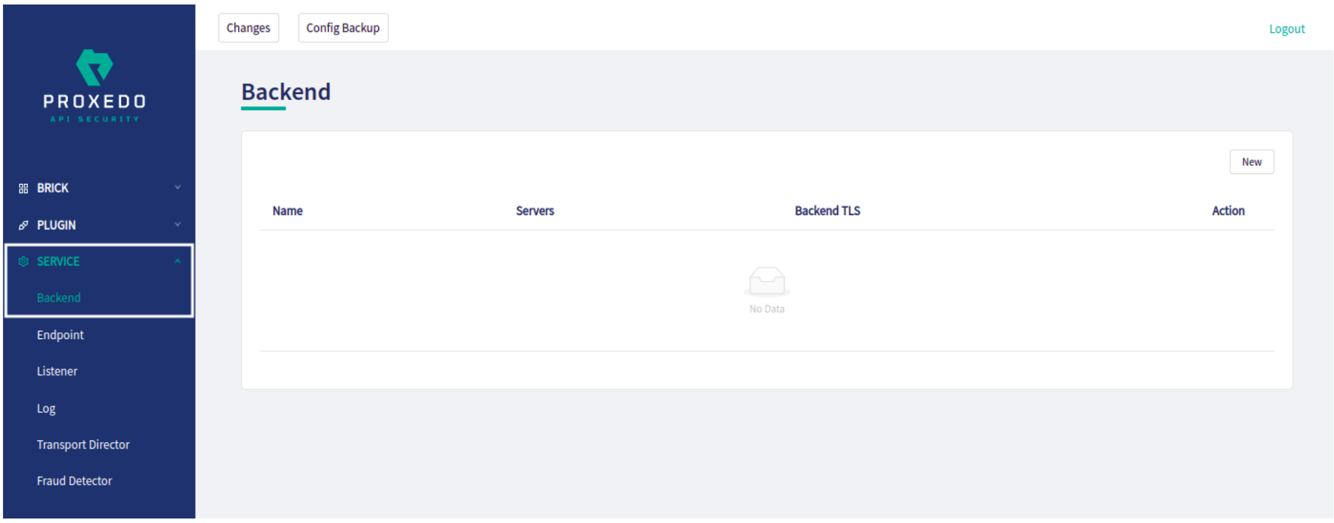
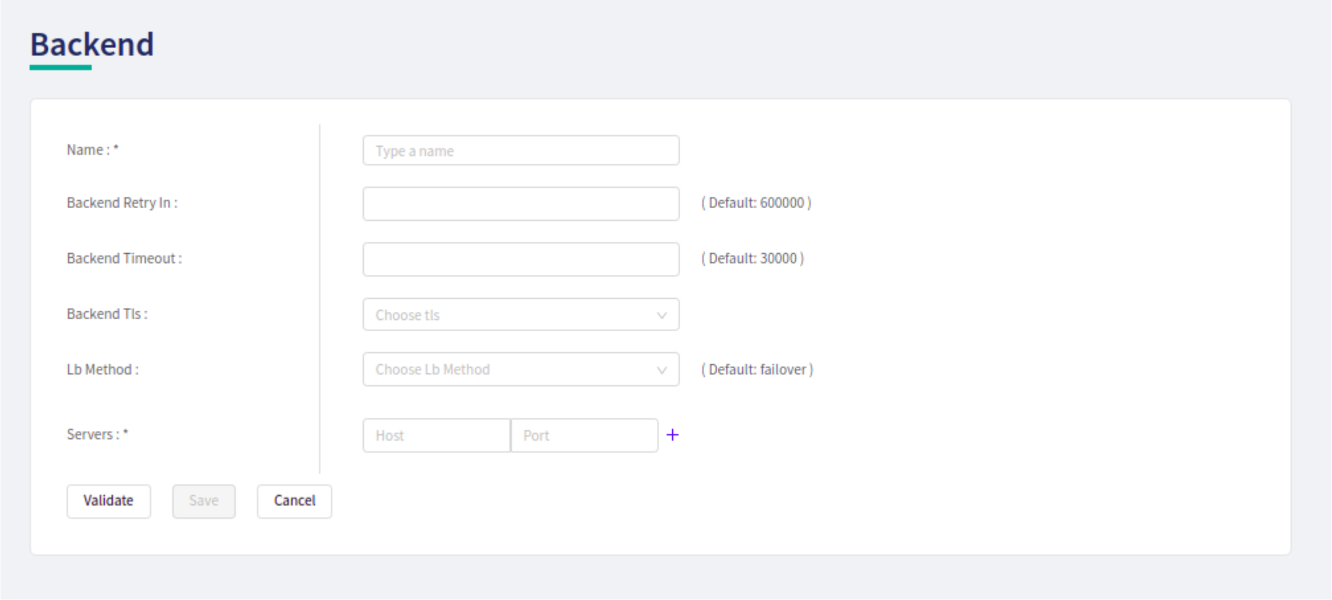


Figure 42. The main page for Backend

3. Click the *New* navigation button to create a Backend configuration.

The following keys are available for Backend configuration:



Backend

Name : * Type a name

Backend Retry In : (Default: 600000)

Backend Timeout : (Default: 30000)

Backend Tls : Choose tls

Lb Method : Choose Lb Method (Default: failover)

Servers : * Host Port +

Figure 43. Configuring backend in the Web User Interface

Backends take the following configuration options:

Table 58. Backend configuration

Key	Values	Default value	Description
Name	It is a mandatory value. It can be defined in free text.		The name identifying the backend. This name of the backend can be referenced from other parts of the configuration.
Backend retry in	If the value is not configured the default value will be added.	600000	It is the timeout in milliseconds before a server -that was down- is restarted again.
Backend timeout	If the value is not configured the default value will be added.	30000	It is the connection timeout in milliseconds of a server that is down.
Backend TLS	The value can be selected from a drop-down list. The drop-down list presents the Backend TLS configurations defined under *BRICK/TLS*. If the value is not set, no TLS will be used in this backend.	none	You can define the TLS configuration towards the backend servers. See Configuring Backend TLS for details.
LB method	One of the following methods can be used: • Failover: use the first server while available, then fail over to the next • RR: 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.

Key	Values	Default value	Description
Servers*	It is a mandatory value. 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.

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 suitable and adequate for configuring the 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. If the configuration of the component is satisfactory, after clicking the *Validate* button, the user receives the 'Component Validation successful' notification.
7. Click the *Save* button, when all required configuration fields have been defined.

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 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 the Endpoint

Endpoints can be configured under the **SERVICE** navigation item.

1. Click on the *SERVICE* main configuration item in the Left navigation area. Alternatively you can also click on the ⌄ sign to open up the sub-navigation items of *SERVICE*.
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:

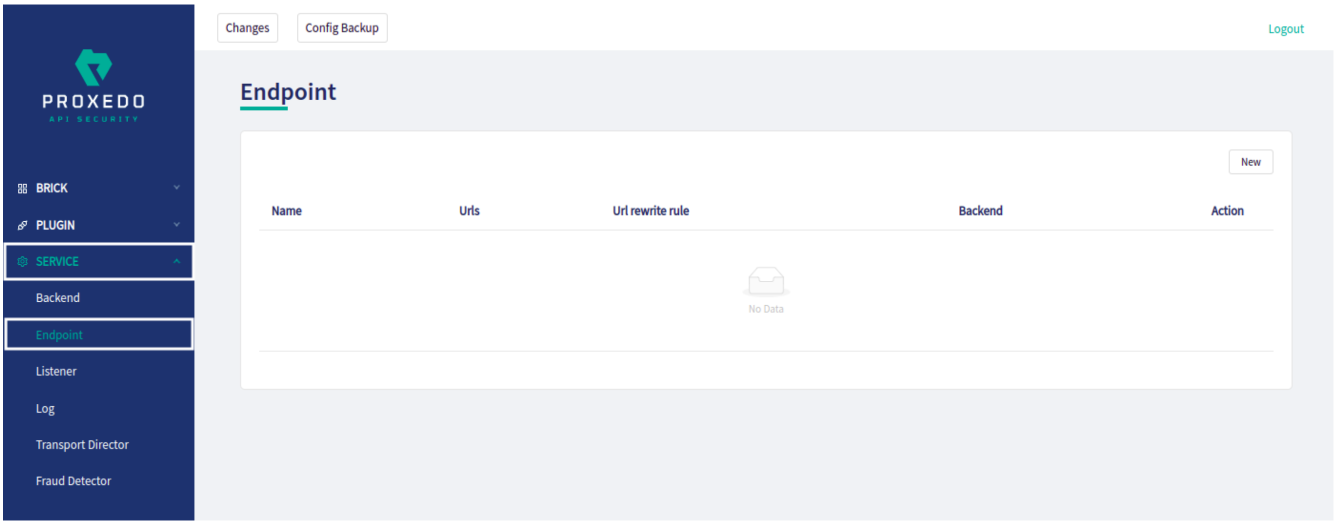
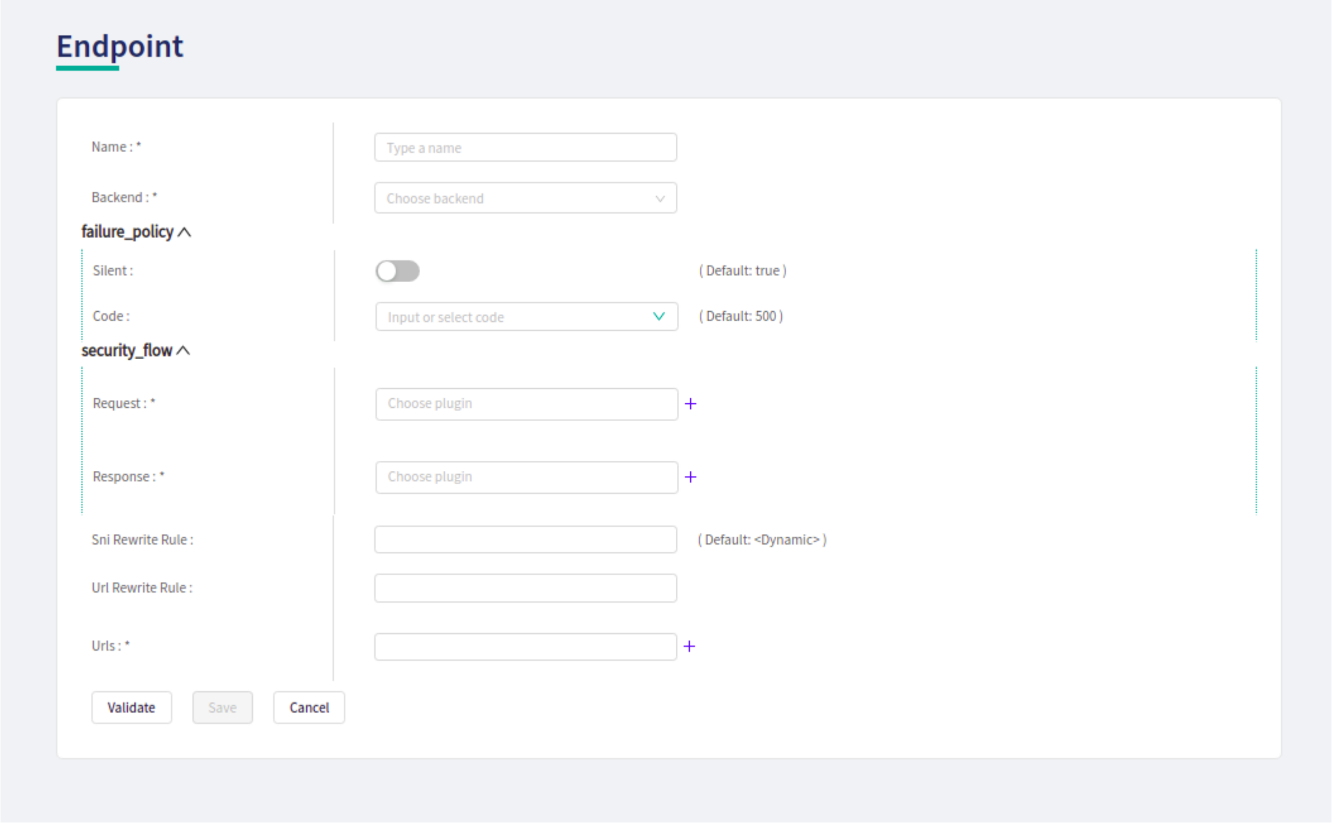


Figure 44. The main page for Endpoint

3. Click the *New* navigation button to create an Endpoint configuration.

The following keys are available for endpoint configuration on the main page of endpoint:



Endpoint

Name : * Type a name

Backend : * Choose backend

failure_policy

Silent : (Default: true)

Code : Input or select code (Default: 500)

security_flow

Request : * Choose plugin +

Response : * Choose plugin +

Sni Rewrite Rule : (Default: <Dynamic>)

Url Rewrite Rule :

Urls : * +

Figure 45. Configuring endpoint in the Web User Interface

Each endpoint has the following configuration options. The elements marked with * are mandatory to be configured.

Table 59. Endpoint configuration

Key	Values	Default value	Description
Name*	It is a mandatory value. It can be defined in free text.		The name identifying the endpoint. This name of the endpoint can be referenced from other parts of the configuration.
Backend*	It is a mandatory value.		Backends are a set of servers for a given API endpoint. For more details, see Backend.
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*. Both values are mandatory values. • Request* • Response*		The values in details are as follows: • Request: It is a mandatory value. 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: It is a mandatory value. 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.
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.
URL rewrite rule			It is 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.
Urls*			It denotes the URLs which the clients use to address the API endpoint.

4. Name the *Endpoint* Service.
5. Select the *Backend* parameter from the drop-down list. Backend servers are configured under the *SERVICE* 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 *PLUGIN* 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 *PLUGIN* main navigation item.

7. Provide the URL to address the API endpoint.
8. Click the *Validate* button to check if the defined parameters are suitable and adequate for configuring the 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. If the configuration of the component is satisfactory, after clicking the *Validate* button, the user receives the 'Component Validation successful' notification.
9. Click the *Save* button, when all required configuration fields have been defined.

ℹ **Backend** and **Backend url** needs to be the same as for all endpoints configured to the same listener.

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

ℹ 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.

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

Listeners can be configured under the **SERVICE** navigation unit.

1. Click on the *SERVICE* main configuration item in the Left navigation area. Alternatively you can also click on the ⌄ sign to open up the sub-navigation items of *SERVICE*.
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:

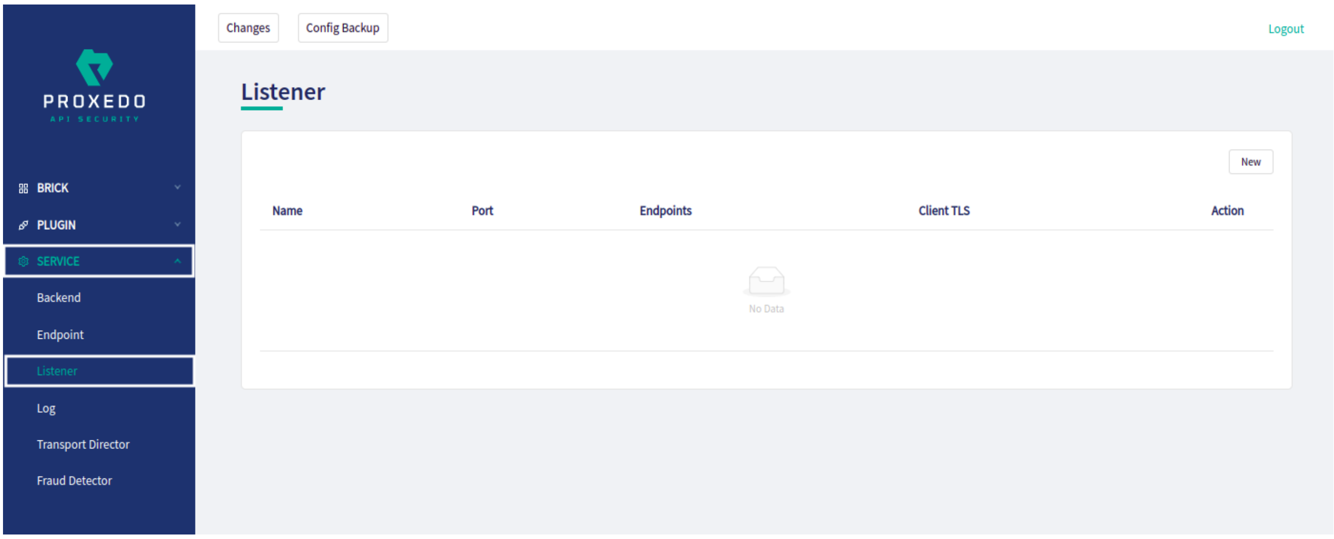
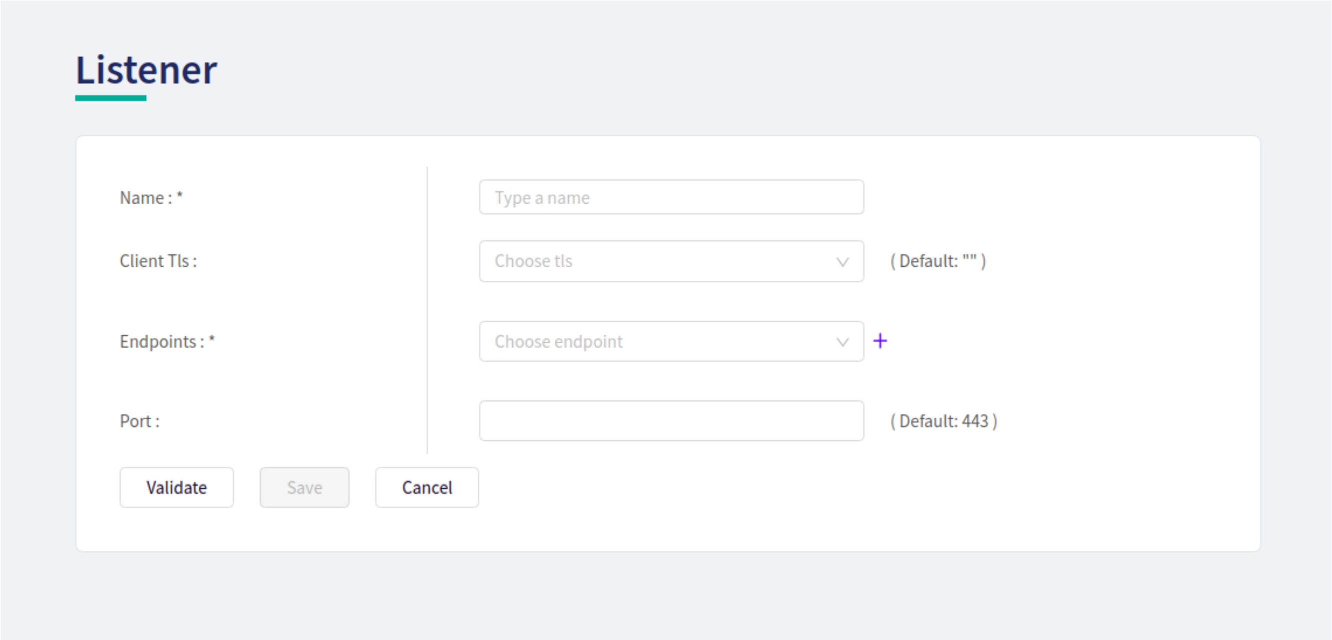


Figure 46. Listener's main page in the Web User Interface

3. Click the *New* button to create a Listener configuration.

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

The following keys are available for listener configuration on the main page of the listener:



Listener

Name : * Type a name

Client Tls : Choose tls (Default: "")

Endpoints : * Choose endpoint +

Port : (Default: 443)

Figure 47. Configuring a listener in the Web User Interface

The listener’s key elements are described in the following table. The elements marked with * are mandatory to be configured.

Table 60. Listeners’ configuration options

Key	Values	Default value	Description
Name*	It is a mandatory value. It can be defined in free text.		It is the name identifying the listener. This name of the listener can be referenced from other parts of the configuration.
Client TLS	The default value is ‘none’, which means, TLS is not used (and therefore HTTPS). You can alternatively select a *Client TLS*, the values of which have to be defined first under BRICK/Client TLS.	None	It is the TLS configuration towards the clients. See TLS for details.
Endpoints*	It is a mandatory value. You can choose the endpoint values from a drop-down list. The endpoint values have to be defined previously under SERVICE/Endpoint.		It is the list of endpoint(s), as defined under Endpoint that serve traffic coming in on the listener.
Port	It is a mandatory value. Any port value can be defined. Note that the port value has to be within the range configured in the docker.	49000	It is the number of the port the listener binds to.

Also consider the followings:

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

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.

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 *BRICK*.
6. Select the *Endpoint* from the drop-down list. The endpoint values have to be defined previously under SERVICE/Endpoint.
7. Fill in the *Port* information. If it is not configured, the default value will be applied.
8. Click the *Validate* button to check if the defined parameters are suitable and adequate for configuring the 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. If the configuration of the component is satisfactory, after clicking the *Validate* button, the user receives the 'Component Validation successful' notification.
9. Click the *Save* button, when all required configuration fields have been defined.

6.6.4. Log

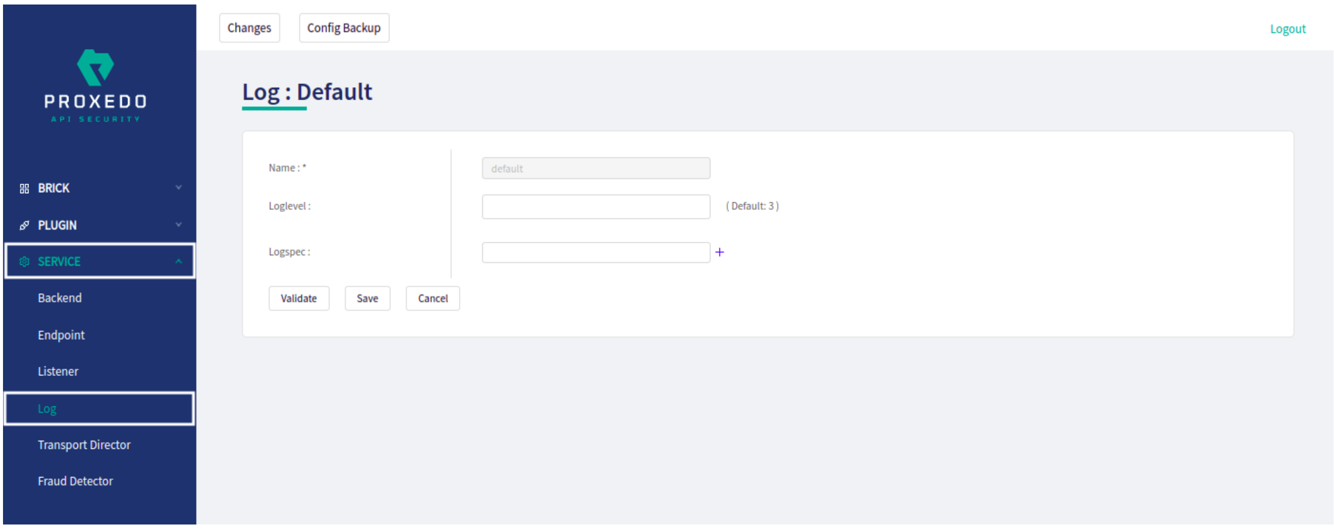
If at any point an error occurs during the Security Flow, the error policy is applied and logging takes place if configured so.

6.6.4.1. Configuring Logs

Logging can be configured under the **SERVICE** main navigation item.

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

The following keys are available for configuration on the main page of Log:



Log : Default

Name : * default

Loglevel : (Default: 3)

Logspec : +

Figure 48. The main page for Logs

> Changes in these settings do not take effect during configuration application. For these changes to take effect, restarting the `{pas_systemd_service_core}` service is necessary.

Table 61. Log configuration

Key	Values	Default value	Description
Name*	Log has a default name 'default', that cannot be changed.		The name identifying the log configuration.
Log level	The value can take number format.	3	It configures the log level to logging. It must be between 1-9.
Log specification	A single log specification consists of a wildcard matching log category, a colon, and a number specifying the verbosity level of that given category. Categories match from left to right. For example: `http.*:5,core: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 *loglevel* 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.

3. Click the *Validate* button to check if the defined parameters are suitable and adequate for configuring the 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. If the configuration of the component is satisfactory, after clicking the *Validate* button, the user receives the 'Component Validation successful' notification.
4. Click the *Save* button, when all required configuration fields have been defined.

6.6.5. Transport Director

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

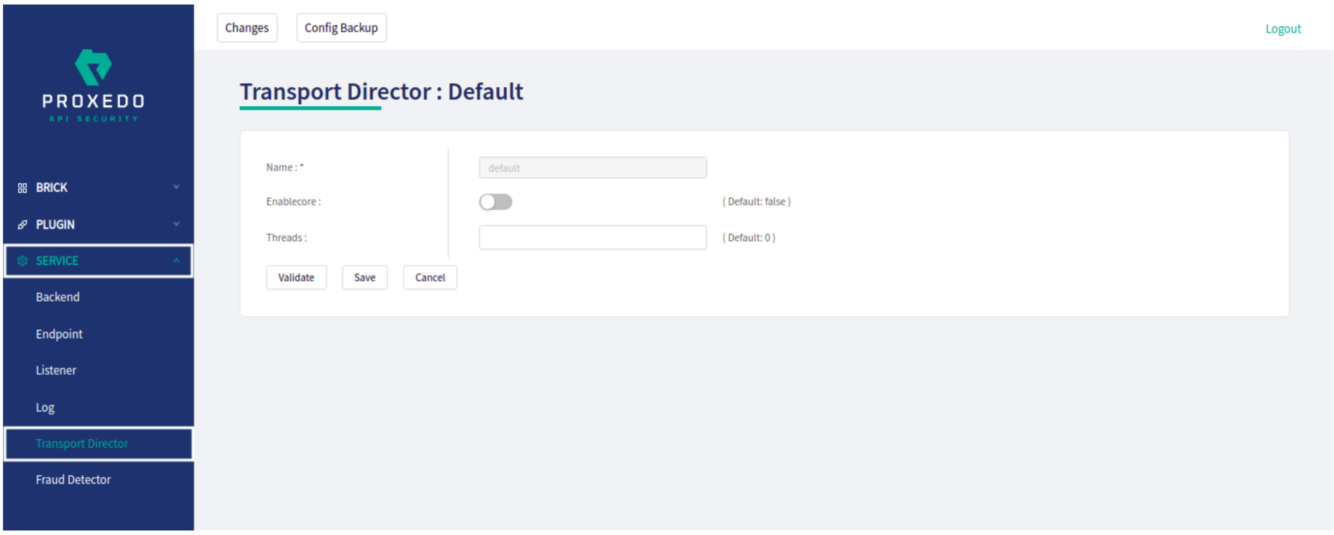
- handles network connections from the client
- handles network connections towards the backends
- handles TLS on these connections
- load-balances between multiple backend servers
- load-balances between multiple *Flow Directors*
- enforces HTTP protocol validity in calls

6.6.5.1. Configuring the Transport Director

The *Transport Director* can be configured under the **SERVICE** main navigation item.

1. Click on the *SERVICE* main configuration item in the Left navigation area. Alternatively you can also click on the ⌄ sign to open up the sub-navigation items of *SERVICE*.
2. Select *Transport Director*.

The following main window appears for the *Transport Director*:



Transport Director : Default

Name : * default

Enablecore : (Default: false)

Threads : (Default: 0)

Figure 49. The main page for Transport Director

The following settings control the *Transport Director* container's startup.

> Changes in these settings do not take effect during configuration application. For these changes to take effect, restarting the `{pas_systemd_service_core}` service is necessary.

Table 62. Transport Director configuration

Key	Values	Default value	Description
Name*	The *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	It can be configured active or inactive.	false	It 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.

3. Click the *Validate* button to check if the defined parameters are suitable and adequate for configuring the 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. If the configuration of the component is satisfactory, after clicking the *Validate* button, the user receives the 'Component Validation successful' notification.
4. Click the *Save* button, when all required configuration fields have been defined.

6.6.6. 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 droppped.

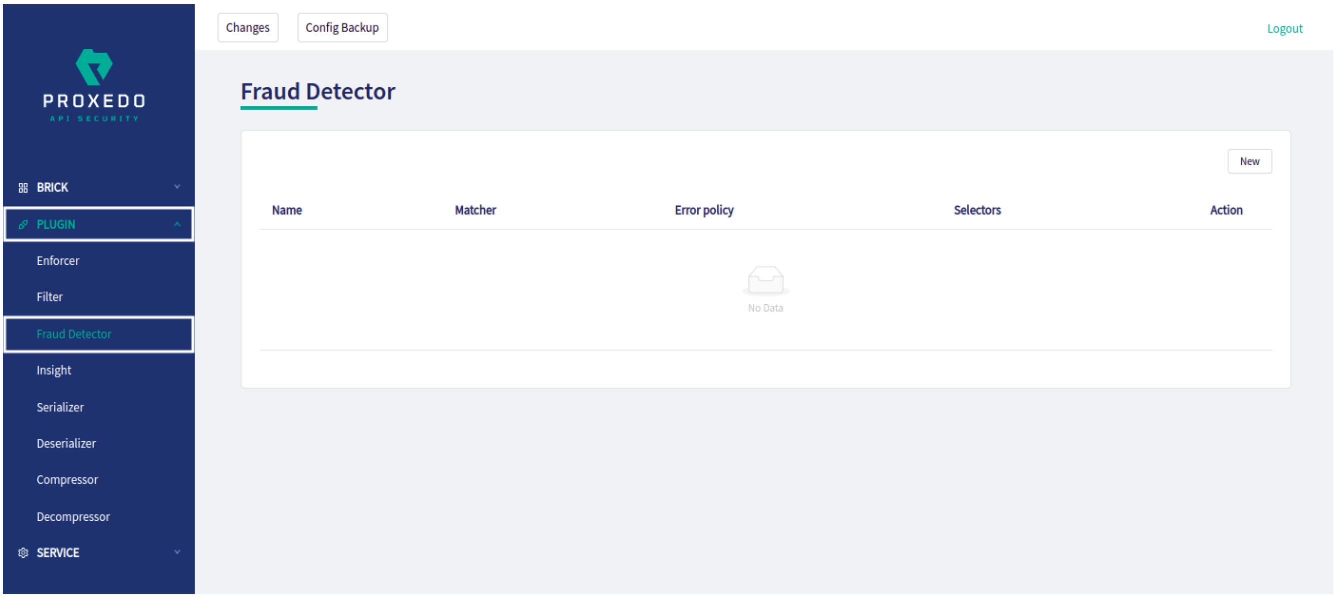
There are three recommended data types to be configured as selectors when configuring the Fraud Detector plugin, namely the IP address, the phone number and the e-mail address. For more details on how to configure Fraud Detector plugin, see Fraud Detector Plugin’s configuration options.

6.6.6.1. Configuring the Fraud Detector

The Fraud Detector can be configured under the **SERVICE** navigation unit.

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

The Fraud Detector’s main configuration window appears:



Fraud Detector

Name	Matcher	Error policy	Selectors	Action

Figure 50. The Fraud Detector's main page in the Web User Interface

The following parameters are available by default on the Fraud Detector's main page. The elements marked with * are mandatory to be configured.

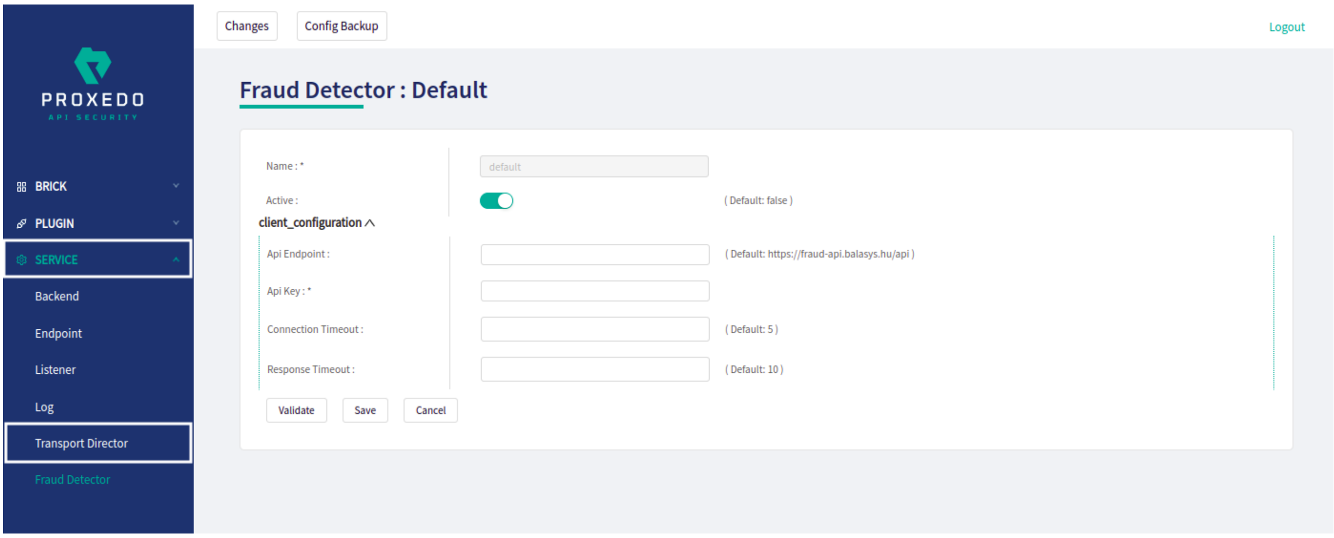
Table 63. Fraud Detector's configuration options

Key	Values	Default value	Description
Name*	The Fraud Detector has a predefined mandatory value, 'default', that cannot be changed.	default	It is the name identifying the Fraud Detector. This name of the Fraud Detector can be referenced from other parts of the configuration.
Active	The Fraud Detector can be active, or inactive.	The default value is 'false', which means, the Fraud Detector is not activated.	If the license for the Fraud Detector is purchased, the service can be activated, if the license for the service is not purchased the service can be set to inactive.

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

3. Set the Fraud Detector service to active state. The Fraud Detector is set to 'inactive' state by default, as for the 'active' state license is required.

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



Fraud Detector : Default

Name : * default

Active : ● (Default: false)

client_configuration

Api Endpoint : (Default: https://fraud-api.balasys.hu/api)

Api Key : *

Connection Timeout : (Default: 5)

Response Timeout : (Default: 10)

Validate Save Cancel

Figure 51. Configuring an active Fraud Detector in the Web User Interface

The Fraud Detector’s additional key elements in active state are described in the following table. The elements marked with * are mandatory to be configured.

Table 64. 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.	This parameter identifies the API endpoint.
Api Key*	It is a mandatory value. 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	This value defines the time limit for establishing connection with the provided url.
Response Timeout	The value can be provided in seconds.	10	This value defines the time limit for how long the PAS awaits the answer from the Fraud API after an established connection.

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 suitable and adequate for configuring the 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. If the configuration of the component is satisfactory, after clicking the *Validate* button, the user receives the 'Component Validation successful' notification.

9. Click the *Save* button.

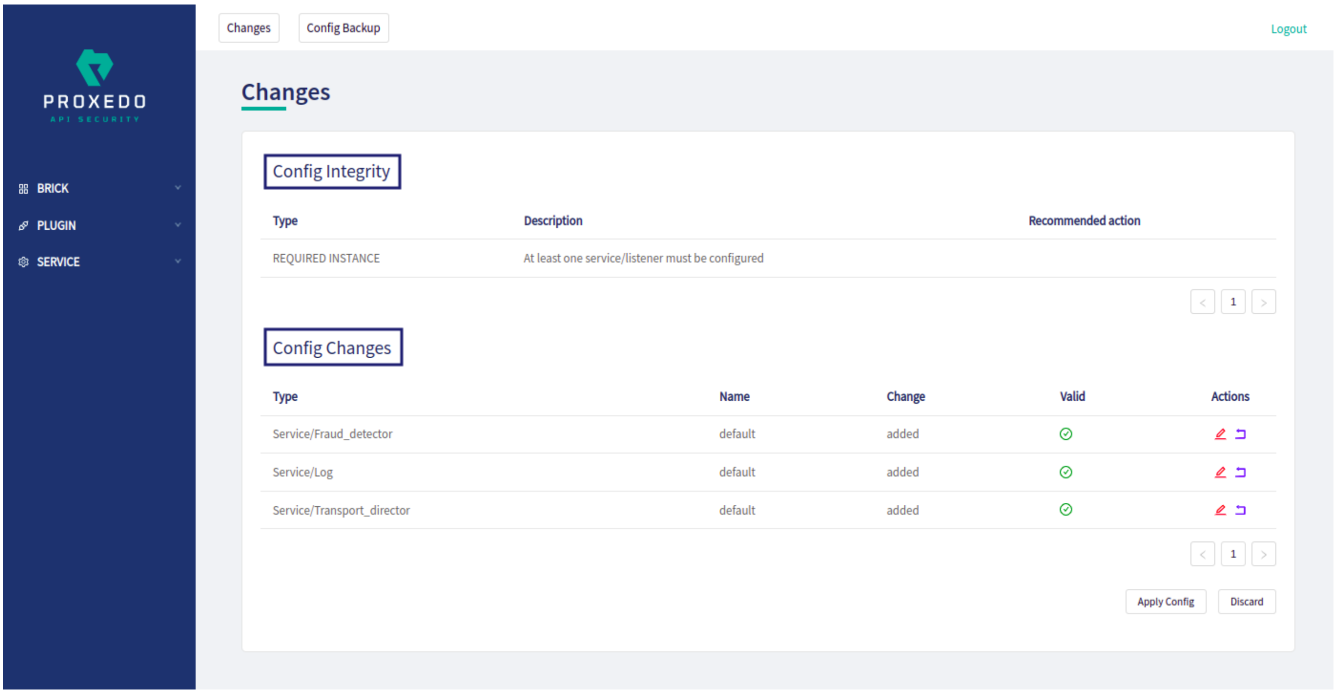
6.7. Checking and finalizing changes in Proxedo API Security configuration

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



Changes

Config Integrity

Type	Description	Recommended action
REQUIRED INSTANCE	At least one service/listener must be configured	

Config Changes

Type	Name	Change	Valid	Actions
Service/Fraud_detector	default	added	✓	
Service/Log	default	added	✓	
Service/Transport_director	default	added	✓	

Figure 52. Checking changes made to the configuration

Configuration Integrity

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

- **Type**: It denotes the type of the integrity problem, for example cycle detection.
- **Description**: Description provides details on the nature of the integrity change.
- **Recommended action**: A recommended action might be displayed here for the configuration integrity problem.

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

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) 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 *no* (no change).
- **Valid**: 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 possibilty to edit the configuration data for a component or to undo any configuration changes to a component. By selecting the undo icon, all changes made to the actual component will be deleted.

> If the edit button is disabled, that is, it is not active, it means that the instance has been deleted. If the undo button is disabled, that is, it is not active, no changes have been made to the actual component.

By selecting the *Discard* button, it is possible to discard all changes made to the configuration. However, the default elements that are created by the system to ease configuration, or the changes that have been applied to the configuration already cannot be discarded.

6.8. Applying and validating Proxedo API Security configuration

PAS configuration can be checked and validated on two levels:

- component-level validation
- validating the whole configuration by using the *Apply Config* button

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.

Component validation failed.

Figure 53. Component validation failed

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

Component validation successful.

Figure 54. Component validation successful

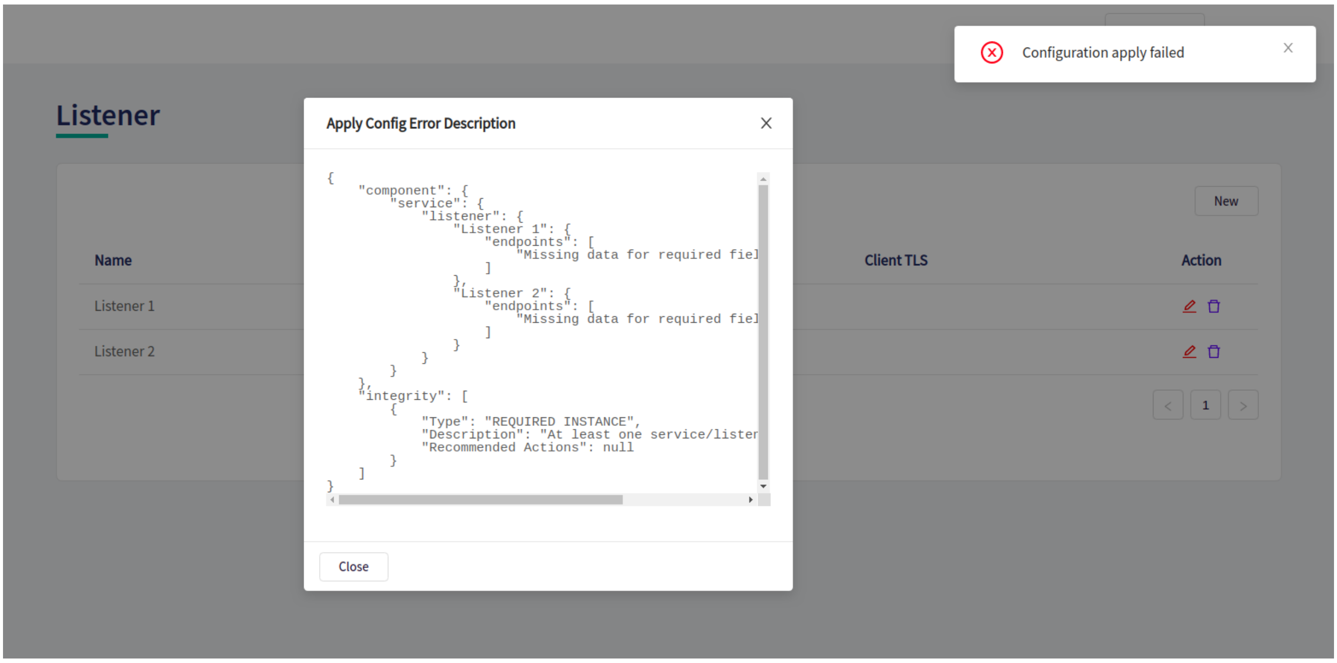
Validating the whole configuration

The *Apply Config* 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 Config* button. Apply the configuration only in case all changes are valid. When applying the configuration by using the *Apply Config* button, the Web UI provides either of the following messages:

- Configuration applied successfully.
- The configuration failed.
  Note, that in case the configuration could not be applied, the Web UI also provides an additional pop-up window with the description of the problem.

Configuration applied successfully

Figure 55. Apply Config result - successful



Configuration apply failed

Apply Config Error Description

```
{
    "component": {
        "service": {
            "listener": {
                "Listener 1": {
                    "endpoints": [
                        "Missing data for required fiel
                    ]
                },
                "Listener 2": {
                    "endpoints": [
                        "Missing data for required fiel
                    ]
                }
            }
        }
    },
    "integrity": [
        {
            "Type": "REQUIRED INSTANCE",
            "Description": "At least one service/lister
            "Recommended Actions": null
        }
    ]
}
```

Figure 56. Apply Config result - failed

6.8.1. 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.

6.8.1.1. Component-related errors

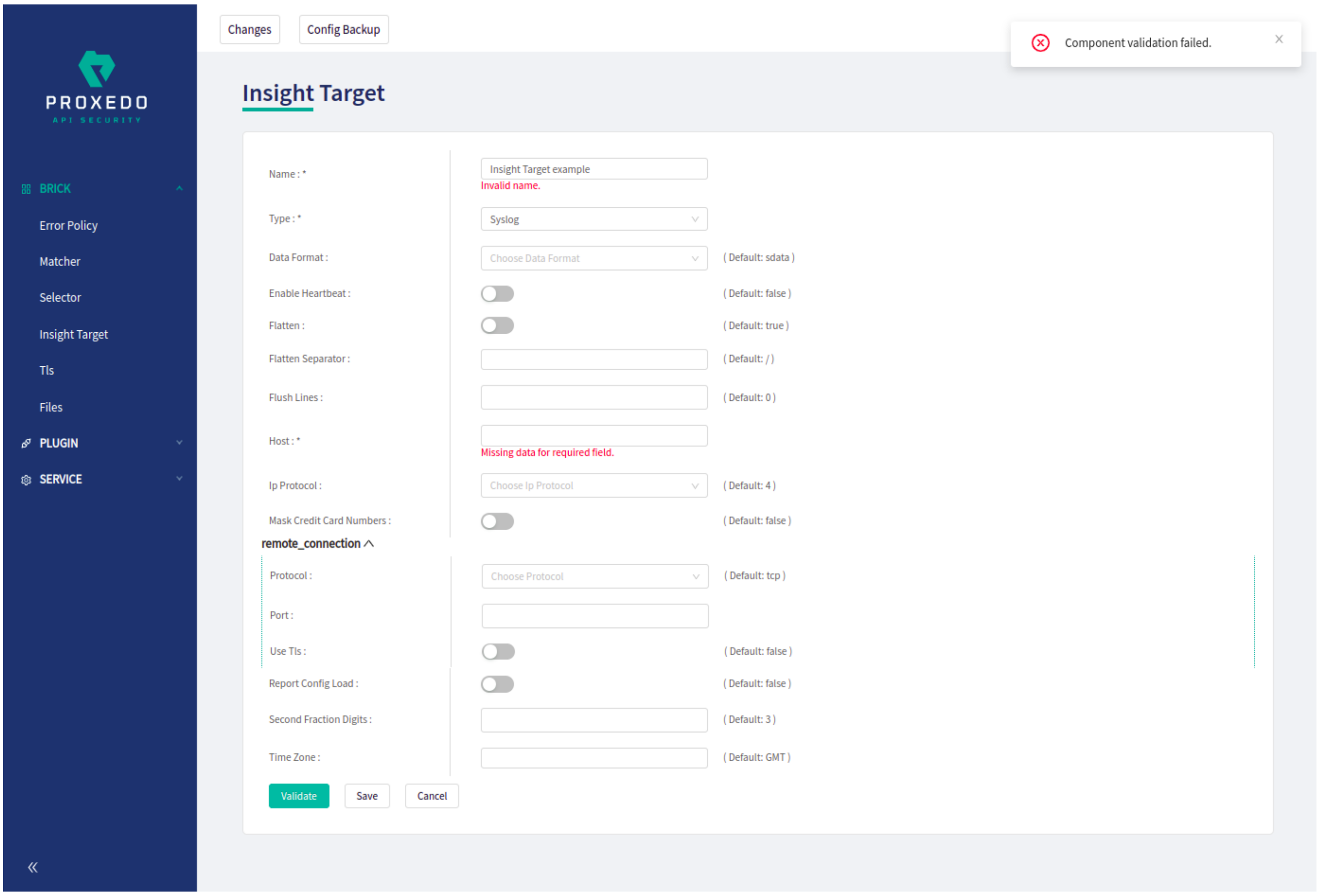
These errors are the results of the validation of the actual components. By correcting these the user can achieve a functioning configuration.

6.8.1.1.1. 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.**



Component validation failed.

Insight Target

Name : * Insight Target example
Invalid name.
Type : * Syslog
Data Format : Choose Data Format (Default: sdata)
Enable Heartbeat : (Default: false)
Flatten : (Default: true)
Flatten Separator : (Default: /)
Flush Lines : (Default: 0)
Host : *
Missing data for required field.
Ip Protocol : Choose Ip Protocol (Default: 4)
Mask Credit Card Numbers : (Default: false)
remote_connection
Protocol : Choose Protocol (Default: tcp)
Port :
Use Tls : (Default: false)
Report Config Load : (Default: false)
Second Fraction Digits : (Default: 3)
Time Zone : (Default: GMT)

Figure 57. Missing required field - Insight Target

6.8.1.1.2. Missing reference

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

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

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

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.

6.8.1.1.3. 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 A uses the same port as Listener B.**

6.8.1.2. Integrity errors

6.8.1.2.1. Cycle detection

This error indicates that there is a cycle of references between the instances. The cycle of references can only be configured in between compound matchers.

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

Error message: **Cyle detected in configuration: BRICK/Matcher/Matcher A→BRICK/Matcher/Matcher B→BRICK/Matcher/Matcher A.**

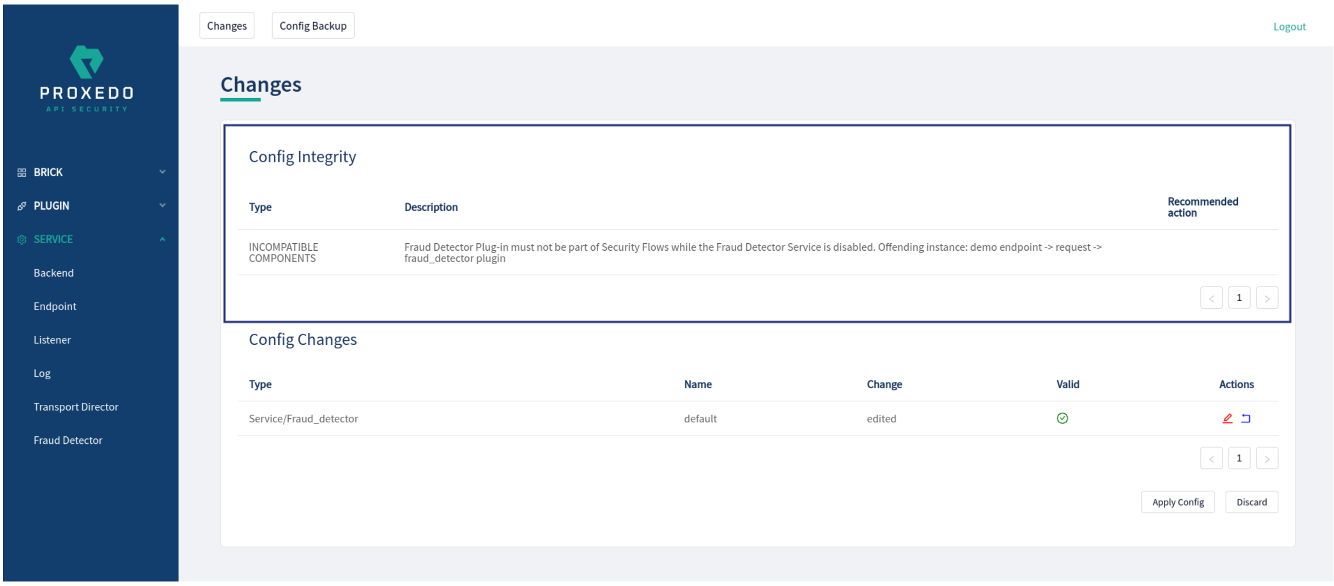
6.8.1.2.2. Required Instance is missing

This error indicates that a required instance is not configured. It is required that at least one Listener service must be configured.

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

6.8.1.2.3. Fraud Detector Plugin configured with the Fraud Detector in inactive state

The following integrity error is indicated:



Changes

Config Integrity

Type	Description	Recommended action
INCOMPATIBLE COMPONENTS	Fraud Detector Plug-in must not be part of Security Flows while the Fraud Detector Service is disabled. Offending instance: demo endpoint -> request -> fraud_detector plugin	

Config Changes

Type	Name	Change	Valid	Actions
Service/Fraud_detector	default	edited	✓	

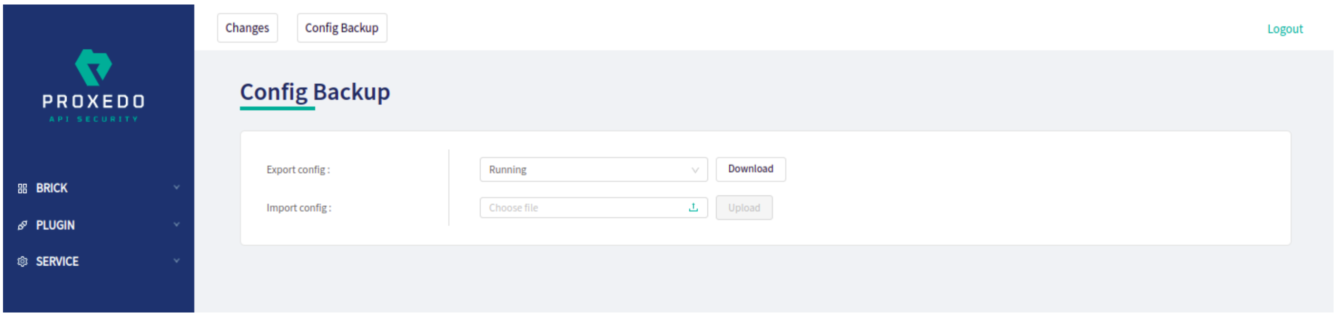
Apply Config Discard

Figure 58. Fraud detector endpoint integrity error

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

6.9. Backup and restore services for Proxedo API Security configuration

It is possible to backup and restore the Proxedo API Security configuration.



Config Backup

Export config : Running

Import config : Choose file

Figure 59. Backup and restore services with Proxedo API Security configuration

In order to export or import any configuration information from or to the system, complete the following steps:

1. Select the *Config backup* button in order to export or import any configuration files.
2. To export a configuration, select the type of the configuration to be exported at the *Export config* 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 config*. 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 in Kubernetes environment

This section introduces different methods of inspecting a PAS service state. For inspecting a PAS service state, it is recommended to use selectors, as selectors utilize all the three labels that are added to most objects of the PAS installation.

The three labels are as follows:

- **app**: This label is present on each object with the value of proxedo-api-security.
- **component**: This label is present on all objects that can be associated with any of the three main components of PAS, such as :
  - *mgmt* for the management object
  - *core* for the core objects
  - *storage* for the storage objects
- **subcomponent**: This label is attached to all objects that are directly and exclusively associated with one subcomponent (services, deployments, pods, network policies, etc.).
  The value of this label is always the name of the subcomponent, for example, *flow-director*, *blob-store*, *config-api*, etc. Since objects are named, using the *proxedo-api-security-<subcomponent-name>* convention, using the *proxedo-api-security-flow-director* object name is most often equivalent to using the *subcomponent=flow-director* selector. Using the selector can be more advantegous, especially with pods, if there are multiple running instances. Since pod names are suffixed with dynamically changing hashes, using a specific pod name can be both inconvenient and sometimes too narrow.

These labels are useful for semantically narrowing down the focus of queries about kubernetes objects.

7.1. Querying objects

By using the `kubectl get` command, objects can be queried with basic information about them.

Run the `kubectl get pods --selector=app=proxedo-api-security` command to get the list of pods related to PAS.

The output will be similar to the following example:

Example output for querying objects

```
NAME                                                      READY   STATUS
RESTARTS   AGE
proxedo-api-security-blob-store-768f54bddd-fpd2v          1/1     Running             0
20m
proxedo-api-security-config-api-5b8b845744-htswp          1/1     Running             0
20m
proxedo-api-security-consul-65f4c78f-26bsg                1/1     Running             0
20m
proxedo-api-security-flow-director-7459896d6c-k9ttm       0/1     ContainerCreating   0
20m
proxedo-api-security-frontend-84798447c4-svrvj            1/1     Running             0
20m
proxedo-api-security-insight-director-5756f4f4b4-jw4vv    0/1     ContainerCreating   0
20m
proxedo-api-security-transport-director-7d4f7fbdf-sh9kw   0/1     ContainerCreating   0
20m
```

In this example, the core components do not have configuration, as that is to be set on the Web UI, and for this reason they are not in *Running* state in the example.

To get PAS services, network policies, and so on, the relevant part of the command referring to 'pods' needs to be changed to the object type in question.

7.2. Inspecting objects

To get more detailed information about any specific kubernetes object, use the `kubectl describe` command. Selectors can also be used with this command, however it is recommended to use this command with a specific object name.

Based on the previous example where core pods were not in *Running* state, the kubectl `kubectl describe pod proxedo-api-security-flow-director-7459896d6c-k9ttm` command can be used to find out the reason behind its malfunction.

The output will be similar to the following example:

Example output for inspecting objects

```
Name:           proxedo-api-security-flow-director-7459896d6c-k9ttm
Namespace:      mate
Priority:       0
Node:           api-kube-node-2/10.90.31.63
Start Time:     Mon, 04 Jul 2022 10:40:56 +0200
Labels:         app=proxedo-api-security
                component=core
                pod-template-hash=7459896d6c
                subcomponent=flow-director
Annotations:    <none>
Status:         Pending
IP:
IPs:            <none>
Controlled By:  ReplicaSet/proxedo-api-security-flow-director-7459896d6c
Containers:
  flow-director:
    Container ID:
    Image:          docker.balasys.hu/api-security/flow-director:4.0.1
    Image ID:
    Ports:          1318/TCP, 8080/TCP
    Host Ports:     0/TCP, 0/TCP
    State:          Waiting
      Reason:       ContainerCreating
    Ready:          False
    Restart Count:  0
    Requests:
      cpu:      250m
      memory:   550Mi
    Readiness:  http-get http://:8000/health delay=5s timeout=1s period=10s #success=1
#failure=1
    Environment:
      INSIGHT_DIRECTOR_HOSTNAME:  proxedo-api-security-insight-director
      SERVICE_ADAPTOR_PORT:       8000
    Mounts:
      /opt/balasys/etc/pas from license (ro)
      /opt/balasys/etc/pas/k8s/configmap from config-configmap (ro)
      /opt/balasys/etc/pas/k8s/secret from config-secret (ro)
      /var/run/secrets/kubernetes.io/serviceaccount from kube-api-access-qbnnv (ro)
Conditions:
  Type              Status
  Initialized       True
  Ready             False
  ContainersReady   False
  PodScheduled      True
Volumes:
  license:
    Type:        Secret (a volume populated by a Secret)
    SecretName:  proxedo-api-security-license
    Optional:    false
  config-configmap:
    Type:      ConfigMap (a volume populated by a ConfigMap)
    Name:      proxedo-api-security-core-config
    Optional:  false
  config-secret:
    Type:        Secret (a volume populated by a Secret)
    SecretName:  proxedo-api-security-core-config
    Optional:    false
  kube-api-access-qbnnv:
    Type:                    Projected (a volume that contains injected data from
multiple sources)
    TokenExpirationSeconds:  3607
    ConfigMapName:           kube-root-ca.crt
    ConfigMapOptional:       <nil>
    DownwardAPI:             true
QoS Class:                   Burstable
Node-Selectors:              <none>
Tolerations:                 node.kubernetes.io/not-ready:NoExecute op=Exists for 300s
                             node.kubernetes.io/unreachable:NoExecute op=Exists for 300s
Events:
  Type     Reason       Age                   From               Message
  ----     ------       ----                  ----               -------
  Normal   Scheduled    37m                   default-scheduler  Successfully assigned
mate/proxedo-api-security-flow-director-7459896d6c-k9ttm to api-kube-node-2
  Warning  FailedMount  33m                   kubelet            Unable to attach or mount
volumes: unmounted volumes=[config-configmap config-secret], unattached volumes=[config-
configmap config-secret kube-api-access-qbnnv license]: timed out waiting for the
condition
  Warning  FailedMount  31m (x11 over 37m)    kubelet            MountVolume.SetUp failed
for volume "config-secret" : secret "proxedo-api-security-core-config" not found
  Warning  FailedMount  17m (x5 over 31m)     kubelet            Unable to attach or mount
volumes: unmounted volumes=[config-configmap config-secret], unattached volumes=[license
config-configmap config-secret kube-api-access-qbnnv]: timed out waiting for the
condition
```

```
  Warning  FailedMount  7m7s (x23 over 37m)  kubelet            MountVolume.SetUp failed
for volume "config-configmap" : configmap "proxedo-api-security-core-config" not found
  Warning  FailedMount  106s (x4 over 35m)   kubelet            Unable to attach or mount
volumes: unmounted volumes=[config-secret config-configmap], unattached volumes=[config-
secret kube-api-access-qbnnv license config-configmap]: timed out waiting for the
condition
```

In this example, the *Events* section of the output shows (among other details) that two necessary configuration objects do not exist, and therefore the pods cannot be started. It also describes the volumes, ports, environment variables and many more attributes that can be helpful for finding out the reason behind its malfunction.

7.3. Checking logs

Logs of PAS components are by default available through the `kubectl logs` command. An extract of the output of `kubectl logs pods/proxedo-api-security-frontend-84798447c4-svrvj` command is displayed in the following example:

Example output for checking logs

```
2022-07-04T09:36:50 config-webui 192.168.235.192 - - [04/Jul/2022:09:36:50 +0000] "POST
/api/v1/auth/login HTTP/1.1" 200 1005 "http://api-kube-node-3.dev.balasys:30001/login"
"Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/103.0.5060.53 Safari/537.36"
2022-07-04T09:36:50 config-webui 192.168.235.192 - - [04/Jul/2022:09:36:50 +0000] "GET
/api/v1/ui-adaptor/menu HTTP/1.1" 200 1942 "http://api-kube-node-
3.dev.balasys:30001/login" "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/103.0.5060.53 Safari/537.36"
2022-07-04T09:36:50 config-webui 192.168.235.192 - - [04/Jul/2022:09:36:50 +0000] "GET
/assets/outline/appstore.svg HTTP/1.1" 200 574 "http://api-kube-node-
3.dev.balasys:30001/" "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/103.0.5060.53 Safari/537.36"
2022-07-04T09:36:50 config-webui 192.168.235.192 - - [04/Jul/2022:09:36:50 +0000] "GET
/assets/outline/api.svg HTTP/1.1" 200 1134 "http://api-kube-node-3.dev.balasys:30001/"
"Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/103.0.5060.53 Safari/537.36"
2022-07-04T09:36:50 config-webui 192.168.235.192 - - [04/Jul/2022:09:36:50 +0000] "GET
/assets/images/proxedo_API_transparent.svg HTTP/1.1" 200 3975 "http://api-kube-node-
3.dev.balasys:30001/changes" "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/103.0.5060.53 Safari/537.36"
2022-07-04T09:36:50 config-webui 192.168.235.192 - - [04/Jul/2022:09:36:50 +0000] "GET
/assets/outline/setting.svg HTTP/1.1" 200 1873 "http://api-kube-node-
3.dev.balasys:30001/" "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/103.0.5060.53 Safari/537.36"
2022-07-04T09:36:50 config-webui 192.168.235.192 - - [04/Jul/2022:09:36:50 +0000] "GET
/SourceSansPro-SemiBold.43cc81b496222dc9ce3c.ttf HTTP/1.1" 200 268280 "http://api-kube-
node-3.dev.balasys:30001/styles.e68c8c26486c2eba6127.css" "Mozilla/5.0 (X11; Linux
x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/103.0.5060.53 Safari/537.36"
2022-07-04T09:36:51 config-webui 192.168.235.192 - - [04/Jul/2022:09:36:51 +0000] "GET
/api/v1/ui-adaptor/config/changes HTTP/1.1" 200 1969 "http://api-kube-node-
3.dev.balasys:30001/changes" "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/103.0.5060.53 Safari/537.36"
2022-07-04T09:36:51 config-webui 192.168.235.192 - - [04/Jul/2022:09:36:51 +0000] "GET
/assets/outline/rollback.svg HTTP/1.1" 200 265 "http://api-kube-node-
3.dev.balasys:30001/changes" "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/103.0.5060.53 Safari/537.36"
```

The `kubectl logs` command can also be used with *Selectors* and other object types like deployments or services. In this case, its scope is wider and can sometimes be more adequate.

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`, `pas-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
2021-04-20T09:15:34 pas-event-handler INFO:SupervisordEventDispatcher:Dispatching event;
processname='pre', eventname='PROCESS
2021-04-20T09:15:34 pas-event-handler INFO:SupervisordEventDispatcher:Process exited;
processname=pre, success=True
2021-04-20T09:15:34 pas-event-handler INFO:SupervisordEventDispatcher:Starting main
processes.
2021-04-20T09:15:34 pas-event-handler INFO:SupervisordEventDispatcher:Starting process;
process='flow-director'
[...]
2021-04-20T09:15:37 flow-director 2021-04-20T09:15:37+0200: flow_builder.info(3)
(nosession): Loaded plugin; [...]
2021-04-20T09:15:37 flow-director 2021-04-20T09:15:37+0200: flow_builder.info(3)
(nosession): Loaded plugin; [...]
2021-04-20T09:15:37 flow-director 2021-04-20T09:15:37+0200: flow_builder.info(3)
(nosession): Loaded plugin; [...]
2021-04-20T09:15:37 flow-director 2021-04-20T09:15:37+0200: flow_builder.info(3)
(nosession): Loaded plugin; [...]
2021-04-20T09:15:37 flow-director 2021-04-20T09:15:37+0200: flow_builder.info(3)
(nosession): Loaded plugin; [...]
2021-04-20T09:15:37 flow-director 2021-04-20T09:15:37+0200: flow_builder.info(3)
(nosession): Loaded plugin; [...]
2021-04-20T09:15:37 flow-director 2021-04-20T09:15:37+0200: flow_builder.info(3)
(nosession): Loaded plugin; [...]
2021-04-20T09:15:37 flow-director 2021-04-20T09:15:37+0200: flow_set.info(3) (nosession):
Start building flows
[...]
2021-04-20T09:15:39 pas-event-handler INFO:SupervisordEventDispatcher:Starting process;
process='service-adaptor'
[...]
2021-05-07T14:23:55 service-adaptor INFO:PASHealthCheck:All services are healthy.
2021-05-07T14:23:55 service-adaptor [pid: 47|app: 0|req: 223/223] 172.19.0.3 () {28 vars
in 350 bytes} [Fri May  7 14:23:55 2021] [...]
```

7.4. Changing bootstrap configuration

Since bootstrap configuration is provided during *Helm* installation, the parameters used there can be changed in the provided files. Moreover, all the input files may be changed. Changing the bootstrap configuration of the management and storage components (config.mgmt and config.storage) also requires manual rolling out of the relevant deployments. As soon as the changes are made, they can be made effective by running the installation command, as displayed in Providing the necessary files for *Helm* installation.

7.5. Backup and restore

7.5.1. Bootstrap configuration

As the whole bootstrap configuration is provided at the time of installation, the directory, in which the installation was carried out, needs to be saved, so that the installation procedure can be repeated.

7.5.2. Running configuration

To completely backup the running configuration, the storage component's *Persistent Volume* needs to be backed up. This can be done by directly backing up the *Persistent Volume* that is assigned to the `proxedo-api-security-storage` *Persistent Volume Claim*. This solution is specific to the Kubernetes Cluster and therefore it is the responsibility of the cluster administrator. In this case, the cluster administrator also needs to make sure that the restored *Persistent Volume* gets assigned to the new *Persistent Volume Claim* from the new PAS installation.

Another method to use the backup mechanism is made available with the Web UI. For more details on that, see Backup and restore services for Proxedo API Security configuration.

7.6. Factory reset

In case a factory reset is necessary, the simplest solution is to delete the namespace, PAS is installed in. If that is not feasible, an alternative is to explicitly delete Kubernetes objects related to PAS. To do so, two main steps are required:

1. Uninstall the PAS *Helm* chart using the `helm uninstall proxedo-api-security` command. This will remove all kubernetes objects managed by the *Helm* charts, including the *Persistent Volume Claim* associated with the storage components.

2. Delete the core configuration objects. These objects are not managed by the *Helm* chart but by the management component. To complete this, run the following commands:

    - `kubectl delete configmap proxedo-api-security-core-config`
    - `kubectl delete secrets proxedo-api-security-core-config proxedo-api-security-registry-credentials`

Following these steps, PAS shall be installed from scratch. For more details, see Installation of Proxedo API Security in Kubernetes environment.

Appendix A: 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
AQ	Antarctica/McMurdo
AQ	Antarctica/Casey
AQ	Antarctica/Davis
AQ	Antarctica/DumontDUrville
AQ	Antarctica/Mawson

Country Code	Time zone Name
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
AU	Australia/Darwin
AU	Australia/Perth
AU	Australia/Eucla
AW	America/Aruba
AX	Europe/Mariehamn

Country Code	Time zone Name
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
BR	America/Eirunepe
BR	America/Rio_Branco
BS	America/Nassau
BT	Asia/Thimphu
BW	Africa/Gaborone

Country Code	Time zone Name
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
CA	America/Dawson
CC	Indian/Cocos
CD	Africa/Kinshasa
CD	Africa/Lubumbashi
CF	Africa/Bangui

Country Code	Time zone Name
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
EG	Africa/Cairo
EH	Africa/El_Aaiun
ER	Africa/Asmara
ES	Europe/Madrid
ES	Africa/Ceuta

Country Code	Time zone Name
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
GU	Pacific/Guam
GW	Africa/Bissau
GY	America/Guyana
HK	Asia/Hong_Kong
HN	America/Tegucigalpa

Country Code	Time zone Name
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
KR	Asia/Seoul
KW	Asia/Kuwait
KY	America/Cayman
KZ	Asia/Almaty
KZ	Asia/Qyzylorda

Country Code	Time zone Name
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
MN	Asia/Choibalsan
MO	Asia/Macau
MP	Pacific/Saipan
MQ	America/Martinique
MR	Africa/Nouakchott

Country Code	Time zone Name
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
NU	Pacific/Niue
NZ	Pacific/Auckland
NZ	Pacific/Chatham
OM	Asia/Muscat
PA	America/Panama

Country Code	Time zone Name
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
RU	Europe/Saratov
RU	Europe/Ulyanovsk
RU	Europe/Samara
RU	Asia/Yekaterinburg
RU	Asia/Omsk

Country Code	Time zone Name
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
SN	Africa/Dakar
SO	Africa/Mogadishu
SR	America/Paramaribo
SS	Africa/Juba
ST	Africa/Sao_Tome

Country Code	Time zone Name
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
US	America/Kentucky/Monticello
US	America/Indiana/Indianapolis
US	America/Indiana/Vincennes
US	America/Indiana/Winamac
US	America/Indiana/Marengo

Country Code	Time zone Name
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
VN	Asia/Ho_Chi_Minh
VU	Pacific/Efate
WF	Pacific/Wallis
WS	Pacific/Apia
YE	Asia/Aden

Country Code	Time zone Name
YT	Indian/Mayotte
ZA	Africa/Johannesburg
ZM	Africa/Lusaka
ZW	Africa/Harare

Appendix B: values.yml examples

B.1. Minimal configuration

The configuration example is set as follows:

- Default TLS settings are used for storage-storage configuration
- Certificates and encryption key are generated by openssl commands
- INFO log level is defined
- If the parameters for the management configuration are not defined, the default values will be used.

Example values.yml file

```
config:
  storage:
    consul:
      gossip_encryption_key: MhstT80sqle63WC7knOak+c7GfK7k5OY2n/4Qk/fSXs=

    blob_store:
      access_key: your_access_key
      secret_key: your_secret_key
```

B.2. Management configuration with LDAP authentication

The configuration example is set as follows:

- LDAP authentication with NTLM and without TLS is used.

Example values.yml

```
config:
  mgmt:
    configapi:
      ldap:
        ldap_url: ldap://ad.example.com
        use_ntlm: on
        bind_user: administrator
        bind_password: your_administartor_password
        user_base_dn: CN=Users,DC=example,DC=com
        group_base_dn: CN=Users,CN=Builtin,DC=example,DC=com
        allowed_groups:
          - Users

  storage:
    consul:
      gossip_encryption_key: MhstT80sqle63WC7knOak+c7GfK7k5OY2n/4Qk/fSXs=

    blob_store:
      access_key: your_access_key
      secret_key: your_secret_key
```

Appendix C: LDAP configuration 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-----
```

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