Docker Leash Server Documentation

Release 0.0.1.dev0

Mathieu Alorent

Mar 15, 2018

Contents:

GitHub repository	
Builds and tests coverage	

CHAPTER 1

Installation

1.1 Server Installation (leash)

The server can be launched from a *virtualenv* or a *docker container*.

Even if it is not mandatory, we recommend to run the service over TLS, *gunicorn* can do it since version 17.0. Alternativelly, you can use any reverse proxy you are used to.

Also, *Flask* ship an internal webserver, it should be avoided on production, just use a dedicated WSGI HTTP Server like *gunicorn* or *wsgi*.

Table of Contents

The server is the central point of authorization and configuration. Run it on a server reachable by your clients (docker daemon). For a better availability, don't hesitate to scale your deployment.

1.1.1 Using virtualenv

Listing 1.1: Install Docker Leash in a *virtualenv*

```
$ virtualenv venv
$ source ./venv/bin/activate
$ pip install docker-leash
$ pip install -r requirements.txt
```

Running the service

Using flask internal webserver

Listing 1.2: Launch Docker Leash using Flask internal server.

```
$ export FLASK_APP=docker_leash.leash_server.py
$ python -m flask run --host 0.0.0.0 --port 80
 * Running on http://[::]:80/
```

Using gunicorn

Listing 1.3: Launch Docker Leash with gunicorn.

```
$ gunicorn --workers=5 --bind=[::]:80 --reload \
  docker_leash.leash_server:app
```

If you choose to support TLS directly with *gunicorn*, just add options *certfile*, *keyfile*, *ciphers* and change port listen port to *443*:

Listing 1.4: Launch gunicorn with TLS support.

```
$ gunicorn --workers=5 --bind=[::]:443 --reload \
  --certfile=/certs/server.crt --keyfile=/certs/server.key \
  --ciphers=TLSv1.2 \
  docker_leash.leash_server:app
```

1.1.2 Using Docker container

Warning: Of course, the *docker-leash* server could be deployed as a Docker container, but be careful to don't brick yourself by running the container on the same host as the one you want to secure.

We publish ready to use container images on Docker Hub, please have a look at our docker repository.

Building the image

You may want to build the image yourself.

Listing 1.5: Build docker image from sources.

```
$ git clone https://github.com/docker-leash/leash-server.git
$ cd leash-server
$ docker build -t leash-server .
```

Running the service

You can simply launch the service using *docker cli* or *docker-compose*. Don't forget to mount the configuration over the respective files.

Listing 1.6: Launch *docker-leash* using *docker*.

```
$ docker run \
  -d \
  -p 443:443 \
  -v /path/to/your/certs/:/certs:ro \
  -v /path/to/your/conf/groups.yml:/srv/docker-leash/groups.yml:ro \
  -v /path/to/your/conf/policies.yml:/srv/docker-leash/policies.yml:ro \
  --certfile=/certs/server.crt --keyfile=/certs/server.key \
  --ciphers=TLSv1.2 \
  dockerleash/leash-server:latest \
  gunicorn --workers=5 --bind=[::]:443 app.leash_server:app
```

Listing 1.7: docker-compose.yml

```
version: '2'

services:
  leashserver:
    image: dockerleash/leash-server:latest
    command: gunicorn --workers=5 --bind=[::]:443 --chdir=/srv/docker-leash \
      --certfile=/certs/server.crt --keyfile=/certs/server.key \
      --ciphers=TLSv1.2 \
      docker_leash.leash_server:app
    volumes:
      - /path/to/your/certs/:/certs:ro
      - /path/to/your/conf/groups.yml:/srv/docker-leash/groups.yml:ro
      - /path/to/your/conf/policies.yml:/srv/docker-leash/policies.yml:ro
    ports:
      - "443:443"
    restart: always
```

Alternatively, you can build a child image including your configuration.

Listing 1.8: Your personnal *Dockerfile*

```
FROM dockerleash/leash-server:latest
COPY configuration/*.yml /srv/docker-leash/
COPY certs/* /certs/
```

Note: The current *command* launched from the image doesn't include TLS options, and listen by default on port *80*. Indeed, the bind mount of */certs*, is optionnal.

Next, read *Client Installation (json)* if your planned rules don't rely on *clients hostname* or *images/containers names* else *Collar Installation (client)*.

1.2 Collar Installation (client)

On *docker daemon* side, deploy the *docker collar plugin*. The *collar-client* will enrich requests from the *docker daemon* with *clients hostname*. It will also try to translate *images* or *containers ids* to *human readable names*. Such conversion are necessary if your planned rules rely on *images/containers names*. If that's not the case, then refer to the *Client Installation (json)*.

1.2.1 Build the plugin yourself

The plugin construction is managed using a *makefile*.

Listing 1.9: building the client image

```
$ make rootfs
$ make create
```

1.2.2 Installing from registry

We provide prebuilt *docker plugin image* directly available on the docker hub.

Listing 1.10: building the client image

```
$ docker plugin install dockerleash/leash-client
```

1.2.3 Configure the plugin

The *collar* need to be configured according to your local environment.

Note: Plugin need to be disabled to be configured.

Variables:

- LEASH_URL mandatory
- LEASH_CA_CERT /certs/ca.pem
- LEASH_CONNECT_TIMEOUT 10
- DOCKER_CA_CERT /certs/ca.pem
- DOCKER_CERT_FILE /certs/cert.pem
- DOCKER_KEY_FILE /certs/key.pem
- DOCKER_URL https://127.0.0.1:2376
- DOCKER_CONNECT_TIMEOUT 10
- DEBUG default false
- ALLOW_PING default true

- ALLOW_READONLY default false
- SENTRY_DSN default None

Mounts:

- certs /certs/ /etc/docker/plugins/collar.d/

Listing 1.11: Define leash server url

```
docker plugin dockerleash/collar:0.1 LEASH_URL=https://your-leash-server
```

1.2.4 Load the plugin on docker daemon start

The *docker daemon* need to start the plugin on boot.

Listing 1.12: Enable the collar

```
docker plugin enable dockerleash/collar:0.1
```

1.2.5 Activate the collar

Edit the *docker daemon* configuration.

Listing 1.13: /etc/docker/daemon.json

```
{
        "authorization-plugins": ["dockerleash/collar:0.1"]
}
```

Next, read *Users authentication*.

1.3 Client Installation (json)

On *docker daemon*, you can connect to the *leash-server* using a simple *json*, but this imply that your planned rules don't rely on *clients hostname* nor *images/containers names*. If that's not the case, then refer to the *Collar Installation (client)*.

1.3.1 Deploy the json file

The connection to the *leash-server* rely on a simple json file placed in usually in the */etc/docker/plugins/* directory.

Listing 1.14: /etc/docker/plugins/leash.json

```
{
  "Name": "leash",
  "Addr": "https://docker-leash.kumy.org",
  "TLSConfig": {
    "InsecureSkipVerify": false,
    "CAFile": "/etc/docker/plugins/cacert-root.pem"
  }
}
```

Fill the *Addr* field with your *leash-server* url. Point the *CAFile* field to the path of the CA used to sign *leash-server* webservice.

Even if it's not recommended, you can disable SSL certificate verification by setting the *InsecureSkipverify* field to *true*.

1.3.2 Activate the plugin

Edit the *docker daemon* configuration and add *authorization-plugins* field to enable the plugin.

Listing 1.15: /etc/docker/daemon.json

```
{
        "authorization-plugins": ["leash"]
}
```

Now restart *dockerd*.

Listing 1.16: restart the *docker* daemon

```
$ systemctl restart docker
```

Next, read *Users authentication*.

1.4 Users authentication

By default, access to the docker daemon is restricted by the permissions set on the unix socket (generally *unix:///var/run/docker.sock*). If your planned policies don't need to know users identity (only anonymous rules), then you can skip jump to *Need Authentication* section.

1.4.1 Anonymous is sufficient

Listing 1.17: /etc/docker/daemon.json

```
{
  "authorization-plugins": ["leash"],
  "hosts": ["unix:///var/run/docker.sock"]
}
```

1.4.2 Need Authentication

If your planned rules rely on user authentication, then the *docker daemon* need to have TLS authentication enabled.

Note: It is only possible to authenticate using SSL certificate only when using TCP socket. When connecting to unix socket, users will be threated as anonymous.

You have many possibilities depending on how you launch the docker daemon (ex: *systemd*, *upstart*...), but the simplest way seems to configure it directly in the */etc/docker/daemon.json*.

Add the *tcp socket* (*"0.0.0.0:2376"*) to the *daemon.json* file.

Use your favorite *SSL Certificate* provider to secure the traffic over the tcp socket. Set fields *tlscert* and *tlskey* to the path of your files.

The *tlscacert*, is responsible for authenticating your clients certificates. We recommend to manage your own CA (Certificate Authority), see *Users authentication via TLS*.

Listing 1.18: /etc/docker/daemon.json

```
{
        "authorization-plugins": ["dockerleash/collar:0.1"],
        "hosts": ["unix:///var/run/docker.sock", "0.0.0.0:2376"],
        "tls": true,
        "tlsverify": true,
        "tlscacert": "/etc/pki/CA/certs/our.company.ca.to.authenticate.users.crt",
        "tlscert": "/etc/pki/tls/certs/full.name.of.your.host.crt",
        "tlskey": "/etc/pki/tls/private/full.name.of.your.host.key"
}
```

Note: On Ubuntu: You may encounter error "*unable to configure the Docker daemon with file /etc/docker/daemon.json: the following directives are specified both as a flag and in the configuration file: hosts: (from flag: [fd:///], from file: [unix:///var/run/docker.sock 0.0.0.0:2376])*." In such case, override systemd script:

$ systemctl edit docker

[Service]
ExecStart=
ExecStart=/usr/bin/dockerd

1.4.3 Users authentication via TLS

For advanced features, users need to authenticate with the *docker daemon*. The current way is use *clients certificates*.

The official docker documentation has a nice tutorial on how to manage CA, *Server* and *Client* certificates.

We also provide informations to *Manage your CA using EasyRSA*.

Next, read *Configuration*.

CHAPTER 2

Configuration

2.1 Configuration

The users authorizations are defined on *leash-server* side.

Configurations are stored as a YAML file. We have one file to define *groups of users*, and one for the *policies*.

The *policies* permit to attach *checks* on *docker action*.

2.1.1 Policies configuration file format

Example configuration file:

Listing 2.1: policies.yml

```
---

- description: Servers are fully accessible to Admins.
               Monitoring group can only list containers.
               Default policy is Deny.
  hosts:
    - +^srv\d\d.*
  default: Deny
  policies:
    - members:
        - admins
      rules:
        any:
          Allow:

    - members:
        - monitoring
      rules:
        containersList:
```

```
          Allow:


- description: Users have access to containers and images starting
               by their name.
               Admin have full access to the host.
               Default policy is ReadOnly.
  hosts:
    - +^wks\d\d.*
  default: ReadOnly
  policies:
    - members:
        - admins
      rules:
        any:
          Allow:

    - members:
        - users
      rules:
        containersLogs:
          ContainerName:
            - ^bar-
            - ^foo-
            - ^$USER-
        containers:
          ContainerName:
            - ^foo-
            - ^$USER-
          BindMounts:
            - -/
            - +/home/$USER
            - +/0


- description: For all other hosts,
               Admin have full access to the host.
               Deny access to Anonymous users.
               Default policy is ReadOnly.
  hosts:
    - +.*
  default: ReadOnly
  policies:
    - members:
        - admins
      rules:
        any:
          Allow:
    - members:
        - Anonymous
      rules:
        any:
          Deny:

...
```

We can break down the sections as follow.

Listing 2.2: General file format

```
- description: <Optionnal: Human description of the ruleset>
  hosts:
    - <server name regexp>
    - ...
  default: <Default action if no rule match> (Deny, Allow, ReadOnly)
  policies:
    - members:
        - <group name>
        - ...
      rules:
        <action 1>:
          <check>:
        <action 2>:
          <check>:
            - <arg1>
            - <arg2>
            - ...
        <action 3>:
          <check>:
            <arg1>: value
            <arg1>: value
            ...: ...
    - ...:
        - <group name>
        - ...
      rules:
        ...:
```

"Docker actions" list

As the list is quite long, please refer to the *Docker actions list* page.

"Checks" list

The *checks* are some sort of plugin to *leash-server*. They permit to verify/filter the access to one or more resources.

check name	Description
Allow	Authorize the request inconditionnally
Deny	Deny the request inconditionnally
ReadOnly	Allow only read-only actions
BindMount	Restrict bind mounts
ContainerName	Restrict by container name
ImageName	Restrict image name
VolumeName	Restrict volume name
Privileged	Check the privileged flag
User	Restrict user

Note: More checks to come. See the related issues in our repository.

2.1.2 Groups configuration file format

Here is a groups policies configuration sample:

Listing 2.3: groups.yml

```
---

admins:
  - rda
  - mal

users:
  - jre
  - lgh
  - dga
  - ore
  - pyr

monitoring:
  - xymon_1
  - xymon_2

anonymous:
  - Anonymous

all:
  - "*"

...
```

We can break down the sections as follow.

```
<group name>:
  - <username 1>
  - <username 2>
```

Note: The *Anonymous* username is a reserved word. It permit to define rules explicitly for non connected users.

You're done! If you need, you can now discover some *Configurations examples*.

2.2 Docker actions list

Docker action names are a mapping to the internal actions from the Docker API.

2.2.1 ContainersCreate

Docker engine API documentation

2.2.2 Sample request payload

Listing 2.4: POST /containers/create


```
{
  "RequestBody": {
    "Tty": true,
    "Cmd": null,
    "Volumes": {},
    "Domainname": "",
    "WorkingDir": "",
    "Image": "linuxserver/smokeping",
    "Hostname": "",
    "StdinOnce": true,
    "HostConfig": {
      "CpuPeriod": 0,
      "MemorySwappiness": -1,
      "ContainerIDFile": "",
      "KernelMemory": 0,
      "Memory": 0,
      "CpuQuota": 0,
      "UsernsMode": "",
      "AutoRemove": true,
      "BlkioDeviceReadIOps": null,
      "Dns": [],
      "ExtraHosts": null,
      "PidsLimit": 0,
      "DnsSearch": [],
      "Privileged": false,
      "IOMaximumIOps": 0,
      "CpuPercent": 0,
      "NanoCpus": 0,
      "Ulimits": null,
      "CpusetCpus": "",
      "DiskQuota": 0,
      "CgroupParent": "",
      "BlkioWeight": 0,
      "MemorySwap": 0,
      "RestartPolicy": {
        "MaximumRetryCount": 0,
        "Name": "no"
      },
      "OomScoreAdj": 0,
      "BlkioDeviceReadBps": null,
      "VolumeDriver": "",
      "ReadonlyRootfs": false,
      "CpuShares": 0,
      "PublishAllPorts": false,
      "MemoryReservation": 0,
      "BlkioWeightDevice": [],
      "ConsoleSize": [0, 0],
      "NetworkMode": "default",
      "BlkioDeviceWriteBps": null,
      "Isolation": "",
      "GroupAdd": null,
      "CpuRealtimeRuntime": 0,
      "Devices": [],
      "BlkioDeviceWriteIOps": null,
      "Binds": null,
      "CpusetMems": "",
```

```
      "Cgroup": "",
      "UTSMode": "",
      "PidMode": "",
      "VolumesFrom": null,
      "CapDrop": null,
      "DnsOptions": [],
      "ShmSize": 0,
      "Links": null,
      "CpuRealtimePeriod": 0,
      "IpcMode": "",
      "PortBindings": {},
      "SecurityOpt": null,
      "CapAdd": null,
      "CpuCount": 0,
      "DeviceCgroupRules": null,
      "OomKillDisable": false,
      "LogConfig": {
        "Config": {},
        "Type": ""
      },
      "IOMaximumBandwidth": 0
    },
    "Labels": {},
    "AttachStdin": true,
    "User": "",
    "Env": [],
    "Entrypoint": null,
    "OnBuild": null,
    "AttachStderr": true,
    "NetworkingConfig": {
      "EndpointsConfig": {}
    },
    "AttachStdout": true,
    "OpenStdin": true
  },
  "RequestMethod": "POST",
  "RequestPeerCertificates": [
↪"LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSURUVENDQWpXZ0F3SUJBZ0lRU25qUmVZbUFBRGFtZjdhbklpeUhLekFOQ
↪"],
  "User": "kumy",
  "RequestUri": "/v1.35/containers/create",
  "RequestHeaders": {
    "Content-Length": "1435",
    "Content-Type": "application/json",
    "Accept-Encoding": "gzip",
    "User-Agent": "Docker-Client/17.12.0-ce (linux)"
  },
  "UserAuthNMethod": "TLS"
}
```


2.2.3 ContainersList

Docker engine API documentation

2.2.4 Sample request payload

Listing 2.5: GET /containers/json


```
{
  "RequestMethod": "GET",
  "RequestPeerCertificates": [
↪"LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSURUVENDQWpXZ0F3SUJBZ0lRU25qUmVZbUFBRGFtZjdhbklpeUhLekFOQ
↪"],
  "User": "kumy",
  "RequestUri": "/v1.35/containers/json",
  "RequestHeaders": {
    "Accept-Encoding": "gzip",
    "User-Agent": "Docker-Client/17.12.0-ce (linux)"
  },
  "UserAuthNMethod": "TLS"
}
```


2.2.5 ContainersUpdate

Docker engine API documentation

2.2.6 Sample request payload

Listing 2.6: POST /containers/{id}/update


```
{
  "RequestBody": {
    "CpuPeriod": 0,
    "MemorySwappiness": null,
    "MemorySwap": 0,
    "BlkioDeviceReadIOps": null,
    "CpuQuota": 0,
    "CpuCount": 0,
    "Memory": 0,
    "PidsLimit": 0,
    "CgroupParent": "",
    "IOMaximumIOps": 0,
    "NanoCpus": 0,
    "CpusetCpus": "",
    "DiskQuota": 0,
    "BlkioWeight": 0,
    "RestartPolicy": {
      "MaximumRetryCount": 0,
      "Name": "always"
    },
    "BlkioDeviceWriteBps": null,
    "CpuShares": 0,
    "OomKillDisable": null,
    "MemoryReservation": 0,
    "BlkioWeightDevice": null,
    "CpuPercent": 0,
    "BlkioDeviceReadBps": null,
    "CpuRealtimeRuntime": 0,
    "Devices": null,
```

```
    "BlkioDeviceWriteIOps": null,
    "CpusetMems": "",
    "KernelMemory": 0,
    "Ulimits": null,
    "CpuRealtimePeriod": 0,
    "DeviceCgroupRules": null,
    "IOMaximumBandwidth": 0
  },
  "RequestMethod": "POST",
  "RequestPeerCertificates": [
↪"LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSURUVENDQWpXZ0F3SUJBZ0lRU25qUmVZbUFBRGFtZjdhbklpeUhLekFOQ
↪"],
  "User": "kumy",
  "RequestUri": "/v1.35/containers/mariadb/update",
  "RequestHeaders": {
    "Content-Length": "619",
    "Content-Type": "application/json",
    "Accept-Encoding": "gzip",
    "User-Agent": "Docker-Client/17.12.0-ce (linux)"
  },
  "UserAuthNMethod": "TLS"
}
```


2.2.7 ImagesBuild

Docker engine API documentation

2.2.8 Sample request payload

Listing 2.7: POST /build


```
{
  "RequestPeerCertificates": [
↪"LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSURUVENDQWpXZ0F3SUJBZ0lRU25qUmVZbUFBRGFtZjdhbklpeUhLekFOQ
↪"],
  "RequestHeaders": {
    "Content-Type": "application/x-tar",
    "Accept-Encoding": "gzip",
    "User-Agent": "Docker-Client/17.12.0-ce (linux)"
  },
  "RequestMethod": "POST",
  "Host": "kumy-nuc",
  "User": "kumy",
  "RequestUri": "/v1.35/build?buildargs=%7B%7D&cachefrom=%5B%5D&cgroupparent=&
↪cpuperiod=0&cpuquota=0&cpusetcpus=&cpusetmems=&cpushares=0&dockerfile=Dockerfile&
↪labels=%7B%7D&memory=0&memswap=0&networkmode=default&rm=1&shmsize=0&t=test&target=&
↪ulimits=null",
  "UserAuthNMethod": "TLS"
}
```


2.2.9 ImagesCommit

Docker engine API documentation

2.2.10 Sample request payload

Listing 2.8: POST /commit


```
{
  "RequestBody": null,
  "RequestMethod": "POST",
  "RequestPeerCertificates": [
↪"LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSURUVENDQWpXZ0F3SUJBZ0lRU25qUmVZbUFBRGFtZjdhbklpeUhLekFOQn
↪"],
  "User": "kumy",
  "RequestUri": "/v1.35/commit?author=&comment=&container=memcached&repo=test&tag=test
↪",
  "RequestHeaders": {
    "Content-Length": "5",
    "Content-Type": "application/json",
    "Accept-Encoding": "gzip",
    "User-Agent": "Docker-Client/17.12.0-ce (linux)"
  },
  "UserAuthNMethod": "TLS"
}
```


2.2.11 ImagesCreate

Docker engine API documentation

2.2.12 Sample request payload

Listing 2.9: POST /images/create


```
{
  "RequestMethod": "POST",
  "RequestPeerCertificates": [
↪"LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSURUVENDQWpXZ0F3SUJBZ0lRU25qUmVZbUFBRGFtZjdhbklpeUhLekFOQn
↪"],
  "User": "kumy",
  "RequestUri": "/v1.35/images/create?fromSrc=-&message=&repo=traefik%3Aalpine2&tag=",
  "RequestHeaders": {
    "Content-Type": "text/plain",
    "Accept-Encoding": "gzip",
    "User-Agent": "Docker-Client/17.12.0-ce (linux)"
  },
  "UserAuthNMethod": "TLS"
}
```


Listing 2.10: POST /images/create


```
{
  "RequestMethod": "POST",
  "RequestPeerCertificates": [
↪"LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSURUVENDQWpXZ0F3SUJBZ0lRU25qUmVZbUFBRGFtZjdhbklpeUhLekFOQn
↪"],
  "User": "kumy",
  "RequestUri": "/v1.35/images/create?fromImage=traefik&tag=alpine",
  "RequestHeaders": {
```

```
    "Content-Length": "0",
    "Content-Type": "text/plain",
    "Accept-Encoding": "gzip",
    "User-Agent": "Docker-Client/17.12.0-ce (linux)"
  },
  "UserAuthNMethod": "TLS"
}
```


2.2.13 ImagesInspect

Docker engine API documentation

2.2.14 Sample request payload

Listing 2.11: GET /images/{name}/json


```
{
  "RequestMethod": "GET",
  "RequestPeerCertificates": [
↪"LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSURUVENDQWpXZ0F3SUJBZ0lRU25qUmVZbUFBRGFtZjdhbklpeUhLekFOQn
↪"],
  "User": "kumy",
  "RequestUri": "/v1.35/images/traefik:alpine/json",
  "RequestHeaders": {
    "Accept-Encoding": "gzip",
    "User-Agent": "Docker-Client/17.12.0-ce (linux)"
  },
  "UserAuthNMethod": "TLS"
}
```

2.2.15 Containers

action name	API description
ContainersList	List containers
ContainersCreate	Create containers
containersInspect	Inspect container
containersListProcess	List processes running inside a container
containersLogs	Get container logs
containersChanges	Get changes on a container's filesystem
containersExport	Export a container
containersStats	Get container stats based on resource usage
containersResizeTTY	Resize a container TTY
containersStart	Start a container
containersStop	Stop a container
containersRestart	Restart a container
containersKill	Kill a container
ContainersUpdate	Update a container
containersRename	Rename a container
containersPause	Pause a container
containersAttach	Unpause a container
containersAttach	Attach to a container
containersAttachWebsocket	Attach to a container via a websocket
containersAttachWebsocket	Wait for a container
containersRemove	Remove a container
containersGetInfoAboutFiles	Get information about files in a container
containersGetFilesystemArchive	Get an archive of a filesystem resource in a container
containersExtractArchiveToDirectory	Extract an archive of files or folders to a directory in a container
containersPrune	Delete stopped containers

2.2.16 Images

action name	API description
imagesList	Returns a list of images on the server.
ImagesBuild	Build an image from a tar archive with a Dockerfile in it.
ImagesCreate	Create an image by either pulling it from a registry or importing it.
images-delete	Remove an image, along with any untagged parent images that were. referenced by that image.
ImagesInspect	Return low-level information about an image.
ImagesCommit	Create a new image from a container.

2.3 "Checks" list

The *checks* are some sort of plugin to *leash-server*. They permit to verify/filter the access to one or more resources.

check name	Description
Allow	Authorize the request inconditionnally
Deny	Deny the request inconditionnally
ReadOnly	Allow only read-only actions
BindMount	Restrict bind mounts
ContainerName	Restrict by container name
ImageName	Restrict image name
VolumeName	Restrict volume name
Privileged	Check the privileged flag
User	Restrict user

Note: More checks to come. See the related issues in our repository.

2.4 Allow

Authorize the request inconditionnally.

2.5 Example usage

Listing 2.12: policies.yml

```
---

- description: Everything is allowed.
  hosts:
    - +.*
  default: Deny
  policies:
    - members:
        - administrators
      rules:
        any:
          Allow:

...
```

2.6 BindMount

Authorize the request only if the bind mount option doesn't run over the defined rules.

Checks are recursive. Every paths not explicitly defined are disallowed. A path not terminated by a / will be considered as starting blob (*/foo**).

Path starting with + represent an allowed path, hence, path starting with - will be disallowed. The special key *$USER* will be replaced by the current connected user. You can escape the *$* sign by preceding it by a *‘ (“$USER”) to use ‘$USER* as a litteral.

BindMount are on evaluated only if the action include binded paths. That's the case for:

- *ContainersCreate*

2.7 Example usage

Listing 2.13: policies.yml

```
---

- description: Allow everything if bind mounts are validated.
  hosts:
    - +.*
  default: Allow
  policies:
    - members:
        - users
      rules:
        any:
          BindMount:
            - "-/"
            - "+/home/$USER/"
            - "+/0/"
            - "-/mnt/something-"

...
```

2.8 ContainerName

Allow the request if the container name respect the rules.

If you wish to use this module then we recommend to *install the collar* it will enrich *docker API* requests by replacing *numeric ids* by *human readable* names used in the *ContainerName* rules.

Containers name check is a list or regular expressions.

2.9 Example usage

Listing 2.14: policies.yml

```
---

- description: Allow action only if container names respect the rules.
  hosts:
    - +.*
  default: Allow
  policies:
    - members:
        - all
      rules:
        containers:
          ContainerName:
            - ^foo-
            - ^bar-
```

```
            - ^$USER-

...
```

2.10 Deny

Deny the request inconditionnally.

2.11 Example usage

Listing 2.15: policies.yml

```
---

- description: Everything is disallowed.
  hosts:
    - +.*
  default: Deny
  policies:
    - members:
        - all
      rules:
        any:
          Deny:

...
```

Note: This example is a bit silly, as the default rule is *Deny*. It could have been written as follow.

Listing 2.16: policies.yml

```
---

- description: Everything is disallowed.
  hosts:
    - +.*
  default: Deny

...
```

2.12 ImageName

Allow the request if the image name respect the rules.

Images name check is a list or regular expressions.

Note: The *docker prune* commands are not restricted by this module.

2.13 Example usage

Listing 2.17: policies.yml

```
---

- description: Allow action only if images names respect the rules.
  hosts:
    - +.*
  default: Allow
  policies:
    - members:
        - all
      rules:
        containers:
          ImageName:
            - ^foo-
            - ^bar-
            - ^$USER-

...
```

2.14 Privileged

Allow the request if the container don't require *privileged* flag.

Volumes name check is a list or regular expressions.

2.15 Example usage

Listing 2.18: policies.yml

```
---

- description: Allow action only if the container has `privileged` flag off.
  hosts:
    - +.*
  default: Allow
  policies:
    - members:
        - all
      rules:
        containers:
          Privileged:

...
```

2.16 ReadOnly

Allow the request if is *read-only*. ie: method of the *docker API* is of type *GET* or *HEAD*.

2.17 Example usage

Listing 2.19: policies.yml

```
---

- description: Allow only read-only actions.
  hosts:
    - +.*
  default: Deny
  policies:
    - members:
        - all
      rules:
        any:
          ReadOnly:

...
```

2.18 User

Allow the request if the container will *run as* specified user.

Volumes name check is a list of regular expressions. If one rule is valid, then the request is validated.

2.19 Related documentation

- docker run documentation

2.20 Examples

Example 6: Restrict by run as user name

Listing 2.20: policies.yml

```
---

- description: Run as user override.
  hosts:
    - +.*
  default: Allow
  policies:
    - members:
        - all
      rules:
        any:
          User:
            - ^nobody$
            - ^$USER$

...
```

2.21 VolumeName

Allow the request if the volumes name respect the rules.

Volumes name check is a list or regular expressions.

2.22 Example usage

Listing 2.21: policies.yml

```
---

- description: Allow action only if volumes names respect the rules.
  hosts:
    - +.*
  default: Allow
  policies:
    - members:
        - all
      rules:
        containers:
          VolumesName:
            - ^foo-
            - ^bar-
            - ^$USER-

...
```

CHAPTER 3

Manage your CA using EasyRSA

We choose to use EasyRSA CLI utility to build and manage our PKI (Public Key Infrastructure) CA. Their README.quickstart.md has good informations on basic usage. Please refer to their documentation for general usage of EasyRSA.

Note: We are used to encrypt the socket traffic using CAcert.org and authenticate our users using our own CA (PKI (Public Key Infrastructure)). But you can also use your own managed CA to accomplish both.

3.1 EasyRSA PKI initialization

Listing 3.1: Initialize an EasyRSA PKI.

```
# Obtain EasyRSA
#
$ wget https://github.com/OpenVPN/easy-rsa/releases/download/v3.0.3/EasyRSA-3.0.3.tgz
$ tar xf EasyRSA-3.0.3.tgz
$ cd EasyRSA-3.0.3


# Configure EasyRSA
#
$ cp vars.example vars
# Edit `vars` file and adapt to your need. You probably need to
# uncomment EASYRSA_REQ_* directives ;)
$ vim vars


# Initialize the pki
#
$ ./easyrsa init-pki
$ ./easyrsa build-ca
```

```
# Copy the CA public key to a central path
$ cp pki/ca.crt /etc/pki/CA/certs/our.company.ca.to.authenticate.users.crt


# If you choose to use your managed CA to encrypt the socket,
# then let's generate a certificate for it. Please replace
# `full.name.of.your.host` with the value you plan to use
# for accessing the docker socket (aka DOCKER_HOST envvar).
#
$ ./easyrsa build-server-full full.name.of.your.host nopass

# Place the generated certificates in a know directory and
# use them in your `/etc/docker/daemon.json` file
# (`tlskey` / `tlscert`)
#
$ cp pki/issued/full.name.of.your.host.crt /etc/pki/tls/certs/full.name.of.your.host.
↪crt
$ cp pki/private/full.name.of.your.host.key /etc/pki/tls/certs/full.name.of.your.host.
↪key
```

3.2 Generate client certificat for user

Listing 3.2: Generate a client certificate for *username1*.

```
# Generate a client certificate
#
$ ./easyrsa build-client-full username1 nopass

# Place generated files in user's home directory
#
$ mkdir /home/username1/.docker/
$ install -o username1 -g username1 -m 0444 pki/issued/username1.crt /home/username1/.
↪docker/cert.pem
$ install -o username1 -g username1 -m 0400 pki/private/username1.key /home/username1/
↪.docker/key.pem
$ install -o username1 -g username1 -m 0444 pki/ca.crt /home/username1/.docker/ca.pem

# Check everything is working
$ docker --tlsverify -H=full.name.of.your.host:2376 version
# or
$ DOCKER_HOST=tcp://full.name.of.your.host:2376 DOCKER_TLS_VERIFY=1 docker version
```

CHAPTER 4

Configurations examples

Examples are accessible in the *examples/configs/example_xx/* directories.

4.1 Example 1: Everything is allowed (as without the plugin)

You can have all the permissions using this configuration. This is the same as running docker daemon without this plugin. Between us, this is not very useful...

Listing 4.1: policies.yml

```
---

- description: Everything is allowed.
  hosts:
    - +.*
  default: Allow

...
```

Listing 4.2: groups.yml

```
---

...
```

Note: In this case, the *groups.yml* could be empty.

4.2 Example 2: Read only / write

Here we want all users to have only the ability to execute read only commands whereas the administrators will have access the write commands too. Anonymous users cannot do anything.

Listing 4.3: policies.yml

```
---

- description: Admins can do everything.
               Authenticated users are restricted to read-only actions.
               Anonymous users cannot do anything.
  hosts:
    - +.*
  default: ReadOnly
  policies:
    - members:
        - administrators
      rules:
        any:
          Allow:

    - members:
        - anonymous
      rules:
        any:
          Deny:

...
```

Listing 4.4: groups.yml

```
---
anonymous:
  - Anonymous

administrators:
  - rda
  - mal
...
```

4.3 Example 3: Restrict by container name

Here we want all users from groups *group1* and *group2* to manage only containers having their name starting with *foo-* or *bar-* or *$USER-*. Otherwize read-only actions are permitted. All administrators are allowed to manage all containers. Anonymous and all other users cannot do anything.

Listing 4.5: policies.yml

```
---

- description: Admins can do everything.
               Users from groups are restricted by container name. Or read-only.
               Anonymous users cannot do anything.
  hosts:
    - +.*
  default: Deny
  policies:
    - members:
        - administrators
      rules:
        any:
          Allow:

    - members:
        - user_group1
        - user_group2
      rules:
        containers:
          ContainerName:
            - ^foo-
            - ^bar-
            - ^$USER-
        any:
          ReadOnly:

...
```

Listing 4.6: groups.yml

```
---
user_group1:
  - jre
  - sve
```

```
user_group2:
  - cjo
  - mgr

administrators:
  - rda
  - mal
...
```

4.4 Example 4: Server are manageable only by admins

Here we want to give access to the servers only by admins. All other hosts are fully accessible by connected users from group but not from admins. Anonymous or other users cannot do anything.

Listing 4.7: policies.yml

```
---

- description: Servers are restricted to admin only.
  hosts:
    - +^srv\d\d.*
  default: Deny
  policies:
    - members:
        - administrators
      rules:
        any:
          Allow:

- description: All other hosts are open to group, else deny.
  hosts:
    - +.*
  default: Deny
  policies:
    - members:
        - users
      rules:
        any:
          Allow:

...
```

Listing 4.8: groups.yml

```
---
users:
  - jre
  - sve
  - cjo
  - mgr

administrators:
  - rda
  - mal
...
```

4.5 Example 5: Server are manageable by admins, workstations by users

Here we want to give full access to the admins. Workstations are are restricted to authenticated users, bind mounts are limited to */home/$USER/* and can only manage *containers* and *images*. All other actions are read-only. Unauthenticated users cannot do anything on workstations. All other hosts are read-only even for admins.

Listing 4.9: policies.yml

```
---

- description: Servers are restricted to admin only.
  hosts:
    - +^srv\d\d.*
  default: Deny
  policies:
    - members:
        - administrators
      rules:
        any:
          Allow:

- description: Workstations are restricted to connected users or admins.
               Users from `users` group can only manage containers or images,
               bind mounts are restricted to `/home/$USER/`. All other actions
               are read-only.
               All other or unauthenticated users cannot do anything
               on workstations.
  hosts:
    - +^wks\d\d.*
  default: Deny
  policies:
    - members:
        - administrators
      rules:
        any:
          Allow:

    - members:
        - users
      rules:
        containers:
          BindMounts:
            - -/
            - +/home/$USER
        images:
          Allow:
        all:
          ReadOnly:

- description: All other hosts are read-only even for admins.
  hosts:
    - +.*
  default: ReadOnly

...
```

Listing 4.10: groups.yml

```
---
users:
  - jre
  - sve
  - cjo
  - mgr

administrators:
  - rda
  - mal
...
```

4.6 Example 6: Restrict by run as user name

Here we want the containers to be running as default image *run as* user. If the user want to override the *run as* user, it is only allowed to *run as his* username or *nobody*.

Listing 4.11: policies.yml

```
---

- description: Run as user override.
  hosts:
    - +.*
  default: Allow
  policies:
    - members:
        - all
      rules:
        any:
          User:
            - ^nobody$
            - ^$USER$

...
```

Listing 4.12: groups.yml

```
---
all:
  - "*"
...
```

CHAPTER 5

docker_leash

5.1 docker_leash package

5.1.1 Subpackages

docker_leash.checks package

Submodules

docker_leash.checks.allow module

Allow

class Allow

Bases: *docker_leash.checks.base.BaseCheck*

A simple module that say *yes*.

run(*args*, *payload*)

Run the module checks.

Saying *yes* is easy: just do not raise any exception ;^)

Parameters

- **args** (*list or dict or str or None*) – The module arguments from the config
- **payload** (*docker_leash.payload.Payload*) – The payload of the current request.

docker_leash.checks.base module

Base

class BaseCheck

Bases: object

The *BaseCheck* class is the base class for all the checks

static replace_user(*value*, *payload*)

A helper function to replace $USER in string

If exact match is found, the replace is done by the value of the current connected user. If no user defined, then it return *None*. If '$' is preceded by a \ (backslash), then no substitution is done.

```
>>> payload = payload({"User": "mal"})
>>> BaseCheck.replace_user("$USER-loves-me", payload)
mal-loves-me
```

```
>>> payload = payload({"User": "mal"})
>>> BaseCheck.replace_user("\$USER-loves-me", payload)
$USER-loves-me
```

```
>>> payload = payload({"User": None})
>>> BaseCheck.replace_user("$USER-loves-me", payload)
# Return None

>>> payload = payload({})
>>> BaseCheck.replace_user("$USER-loves-me", payload)
# Return None
```

It works with *list* too:

```
>>> a = ["who-loves-you", "$USER-loves-me", "\$USER-loves-me"]
>>> payload = payload({"User": "mal"})
>>> BaseCheck.replace_user("$USER-loves-me", payload)
["who-loves-you", "mal-loves-me", "\$USER-loves-me"]
```

Parameters

- **value** (*string or list or None*) – The input to replace
- **payload** (*docker_leash.payload.Payload*) – The payload of the current request.

Returns The replaced value(s)

Return type string or list or None

run(*args*, *payload*)

Run the module checks.

The implemented check module receive the global configuration and the current payload. It is autonomous in how the checks are implemented.

If the requested action should be forbidden then the module must raise an *docker_leash.exceptions.UnauthorizedException*.

> **Warning:** This function *must* be overridden in each check modules.

Parameters

- **args** (*list or dict or str or None*) – The module arguments, from the configuration
- **payload** (docker_leash.payload.Payload) – payload of the current request

docker_leash.checks.deny module

Deny

class Deny
Bases: *docker_leash.checks.base.BaseCheck*

A simple module that say *no*

run(*args*, *payload*)
Run the module checks.

Saying no is easy, just raise an *docker_leash.exceptions.UnauthorizedException* ;^)

Parameters

- **args** (*list or dict or str or None*) – The module arguments from the config
- **payload** (docker_leash.payload.Payload) – payload of the current request

Module contents

__init__

5.1.2 Submodules

5.1.3 docker_leash.action_mdocker_leasher module

Action

class Action(*method*, *query*)
Bases: object

Variables

- ***method*** – HTTP method used
- ***query*** – unparsed query
- ***name*** – parsed name (i.e.: containersList, imagesBuild, . . .)
- ***version*** – API version call, if present
- ***querystring*** – parsed querystring
- ***namespace_name*** – (i.e.: containers, images, . . .)

_Action__namespace = None

_namespace_map = {'images': <class 'docker_leash.action_mapper.Image'>}

_parse()

Raise InvalidRequestException

_r_parse = <_sre.SRE_Pattern object>

is_readonly()
Can the action affect the state of a resource.

Return type bool

method = None

name = None

namespace
Initialize and return a Namespace object when required

Return type Namespace instance

namespace_name = None

classmethod namespace_register(*arg*)
A decorator to register Namespace for a given namespace name

If no namespace name is provided, it will be derivated from the class name.

Parameters

- **cls** (Action) –
- **arg** (*str or Namespace instance*) –

query = None

querystring = None

version = None

class Image(*action*)
Bases: *docker_leash.action_mapper.Namespace*

Proof of concept for Namespace subclasses

names()

Return type list or None

class Namespace(*action*)
Bases: object

Generic namespace

Variables ***action*** – link to (parent) action

action = None

5.1.4 docker_leash.checks_list module

Checks

class Checks
Bases: object

The *Checks* class is responsible for storing and deduplicating the checks to be launched.

Variables ***docker_leash.checks_list.Checks.checks*** (*list*) – The check list.

static _structure_convert(*data*)
An internal helper that will convert structure from the configuration to a better internal format.

Parameters **data** (*dict*) – The check to append. It is a dictionary with a single key / value.

Returns The reformatted test and arguments.

Return type dict

add(*data*)
Add a check to the list.

If the same check is already in the list, it will silently be discarded.

Parameters **data** (*dict*) – The check to append.

checks = None
list: Internal storage of the checks to be applied.

5.1.5 docker_leash.config module

Config

class Config(*groups=None*, *policies=None*)
Bases: object

The *Config* class is responsible for storing application groups and policies read from the datastore.

It has some handy functions to extract values from the configuration. It can respond to questions such as:

- "Which are the groups for a user?"
- "Which policies user belong to?"
- "Which tests are enabled for a user?"

Variables

- ***groups*** (*dict or None*) – The groups.
- ***policies*** (*dict or None*) – The policies.

Parameters

- **groups** (*dict or None*) – The groups.
- **policies** (*dict or None*) – The policies.

static _default_rule(*rule*)
Construct a default rule

Parameters **rule** (*dict*) – The current parsed rule

Returns A Checks containing only the default rule

Return type Checks

_get_policy_by_member(*username*, *policies*)
Extract the policies for a user name.

Return the concerned policies:

- If the user match in a group
- If the user is None, and "members" contains "Anonymous"
- Else return None

Parameters

- **username** (*str*) – The username
- **policies** (*dict*) – The policies to filter

Returns The policies for username

Return type None or dict

static _match_host(*hostname*, *host_rules*)
Validate if a hostname match hosts regex list

Parameters

- **hostname** (*str*) – The hostname
- **host_rules** (*list*) – List of hosts regex

Returns True if hostname match host rules

Return type bool

Raises ***ConfigurationException*** – if the host rules are invalid.

static _match_rules(*action*, *actions*)
Extract the checks for an action.

First match for exact comparison, then for the "any" keyword, and finally for "parents" action name.

Parameters

- **action** (docker_leash.action_mapper.Action) – The current action
- **actions** (*dict*) – The actions from the policies

Returns The filtered actions list

Return type *~docker_leash.checks_list.Checks*

get_rules(*payload*)
Return the rules for a payload.

Parameters **payload** (*str*) – The current payload.

Returns The rules concerned by the payload.

Return type list

groups = None
The loaded groups

policies = None
The loaded policies

update(*groups=None*, *policies=None*)
Update the stored configuration with the provided values.

Parameters

- **groups** (*dict or None*) – The groups.

- **policies** (*dict or None*) – The policies.

5.1.6 docker_leash.exceptions module

Exceptions used in Docker Leash Server

exception ConfigurationException(*value=None*)
Bases: *docker_leash.exceptions.DockerLeashException*

Exception for configuration errors.

Used when configuration files are invalid.

exception DockerLeashException(*value=None*)
Bases: exceptions.BaseException

Base for all Leash Server Errors.

json()
Format the exception as dict object.

Returns the exception as dict

Return type dict

exception InvalidRequestException(*value=None*)
Bases: *docker_leash.exceptions.DockerLeashException*

Exception for invalid payload.

Used when payload is invalid or incomplete.

exception NoSuchCheckModuleException(*value=None*)
Bases: *docker_leash.exceptions.DockerLeashException*

Exception for non existent check module.

Used when a check is configured by not existent or loadable.

exception UnauthorizedException(*value=None*)
Bases: *docker_leash.exceptions.DockerLeashException*

Exception for unauthorized action.

All *docker_leash.checks* modules must return this exception in order to deny the action to the user.

5.1.7 docker_leash.leash_server module

Leash server plugin for docker.

This module is responsible for dispatching HTTP requests.

activate()
Return implemented event system.

It is used internally by the Docker daemon to indicate which event system is concerned by the plugin. In the case of this plugin, it return *authz*.

See the official docker documentation.

Request:

```
GET /Plugin.Activate HTTP/1.1
Host: example.com
Accept: application/json
```

Response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: application/json

{
  "Implements": ["authz"]
}
```

Resheader Content-Type application/json

Status 200 valid response

Return type `flask.Response`

authz_request()

Process a request for authorization.

This is one of the main feature of this plugin. Depending on the configuration, the system, will allow or deny a request.

For a specific user, if no configuration match the *RequestMethod* and the *RequestUri*, then the default action is to deny the request.

See also:

Function *authz_response()* for response authentication.

See also:

See official docker documentation.

Request:

```
GET /AuthZPlugin.AuthZReq HTTP/1.1
Host: example.com
Accept: application/json

{
  "User": "mal",
  "AuthenticationMethod": "TLS",
  "RequestMethod": "POST",
  "RequestUri": "/v1.32/containers/json",
  "RequestHeaders": "<base64 encoded string>",
  "RequestBody": "<base64 encoded string>"
}
```

Response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: application/json

{
  "Allow": "true",
```

```
  "Msg": "Authorization granted",
  "Err": "Authorization granted"
}
```

Reqheader Accept application/json

<json string User The user identification

<json string AuthenticationMethod The authentication method used

<json enum RequestMethod The HTTP method (GET/DELETE/POST)

<json string RequestUri The HTTP request URI including API version (e.g., /v1.32/containers/json)

<json map[string]string RequestHeaders Request headers as key value pairs (without the authorization header)

<json []byte RequestBody Raw request body

>json bool Allow Boolean value indicating whether the request is allowed or denied

>json string Msg Authorization message (will be returned to the client in case the access is denied)

>json string Err Error message. Will be returned to the client in case the plugin encounter an error. The string value supplied may appear in logs, so should not include confidential information.

Resheader Content-Type application/json

Status 200 valid response

Status 400 malformed request

Status 422 invalid parameters

Return type `flask.Response`

authz_response()

Process a response for authorization.

This is one of the main feature of this plugin. Depending on the configuration, the system, will allow or deny a request.

> **Warning:** In the current version, we don't check any parameter, and always accept the request.

In contrast to *authz_response()*, this endpoint is called after the action has been processed by the docker daemon. The request payload contains additional fields representing the response from the daemon.

See also:

Function *authz_request()* for request authentication.

See also:

Check the official docker documentation.

Request:

```
GET /AuthZPlugin.AuthZReq HTTP/1.1
Host: example.com
Accept: application/json
```

```
{
  "User": "mal",
  "AuthenticationMethod": "TLS",
  "RequestMethod": "POST",
  "RequestUri": "/v1.32/containers/json",
  "RequestHeaders": "<base64 encoded string>",
  "RequestBody": "<base64 encoded string>",
  "ResponseStatusCode": "200",
  "ResponseHeaders": "<base64 encoded string>",
  "ResponseBody": "<base64 encoded string>"
}
```

Response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: application/json

{
  "Allow": "true",
  "Msg": "Authorization granted",
  "Err": "Authorization granted"
}
```

Reqheader Accept application/json

<json string User The user identification

<json string AuthenticationMethod The authentication method used

<json enum RequestMethod The HTTP method (GET/DELETE/POST)

<json string RequestUri The HTTP request URI including API version (e.g., /v1.32/containers/json)

<json map[string]string RequestHeaders Request headers as key value pairs (without the authorization header)

<json []byte RequestBody Raw request body

<json int ResponseStatusCode Status code from the docker daemon

<json map[string]string ResponseHeaders Response headers as key value pairs

<json []byte ResponseBody Raw docker daemon response body

>json bool Allow Boolean value indicating whether the request is allowed or denied

>json string Msg Authorization message (will be returned to the client in case the access is denied)

>json string Err Error message. Will be returned to the client in case the plugin encounter an error. The string value supplied may appear in logs, so should not include confidential information.

Resheader Content-Type application/json

Status 200 valid response

Status 400 malformed request

Status 422 invalid parameters

Return type `flask.Response`

index()
Main entry point. it respond to the *GET* method for the / uri.

setup_app(*application*)
Initialize the application

5.1.8 docker_leash.payload module

Payload

class Payload(*payload=None*)
Bases: `object`

The `Payload` class is responsible for decoding and storing the current analyzed request.

Variables

- ***data*** (*dict or None*) – The full request payload.
- ***user*** (*str or None*) – The connected user.
- ***method*** (*str or None*) – The request HTTP method.
- ***uri*** (*str or None*) – The request URI.
- ***headers*** (*dict or None*) – The request Headers.

Parameters **payload** (*dict or None*) – The payload to analyze and store.

static _decode_base64(*payload*)
Decode some parts of the payload from base64 to dict.

Parameters **payload** (*dict*) – The payload to fully base64 decode.

Returns The fully decoded payload.

Return type dict

static _get_headers(*payload*)
Extract the *Headers* from the payload.

Parameters **payload** (*dict*) – The payload to extract URI.

Returns The Headers.

Return type dict

static _get_host(*payload*)
Get the hostname

static _get_method(*payload*)
Extract the *Method* from the payload.

Parameters **payload** (*dict*) – The payload to extract method.

Returns The method name.

Return type str or None

Raises ***InvalidRequestException*** – if the payload is missing RequestMethod.

static _get_uri(*payload*)
Extract the *URI* from the payload.

Parameters **payload** (*dict*) – The payload to extract URI.

Returns The URI.

Return type str or None

static _get_username(*payload*)
Extract the *User* from the payload.

If the user is not connected (i.e.: *anonymous*), the value is an empty string.

Parameters **payload** (*dict*) – The payload to extract username.

Returns The username.

Return type str or None

data = None
The full request payload

get_headers()
Get the headers

get_host()
Get the hostname

headers = None
The request Headers

host = ''
The request Headers

method = None
The request HTTP method

uri = None
The request URI

user = None
The connected user

5.1.9 docker_leash.processor module

Processor

class Processor
Bases: object

The *Processor* class is responsible for launching all the *docker_leash.checks* defined in the configuration for the triplet *User*, *RequestMethod* and *RequestUri*.

static _process(*payload*, *check*)
Instanciate the requested action and launch *docker_leash.checks.base.BaseCheck.run()*

Parameters

- **payload** (Payload) – The request payload object.
- **check** (*str*) – The check name to run.

Raises

- ***UnauthorizedException*** – if the check denied the request.
- ***NoSuchCheckModuleException*** – if the check doesn't exists.

config = None
The currently loaded rules.

load_config()
Load rules from defined files in the global configuration.

Look for path in that order: path in config file, */etc/docker-leash/*, */config/* and docker-leash module directory.

run(*body=None*)
Check if the request is *accepted* or *denied*.

The request will be passed to all configured *docker_leash.checks* for the triplet *docker_leash.payload.Payload.user* + *docker_leash.payload.Payload.method* + *docker_leash.payload.Payload.uri*. If one *docker_leash.checks* sub-modules deny the action, then the whole request is declared as *denied*.

Parameters **body** (*str or dict or None*) – The HTTP request body

Raises

- ***UnauthorizedException*** – if the check denied the request.
- ***NoSuchCheckModuleException*** – if the check doesn't exists.

5.1.10 Module contents

A remote AuthZ plugin to enforce granular rules for a Docker multiuser environment

CHAPTER 6

API Documentation

6.1 Summary

6.2 API Details

POST /AuthZPlugin.AuthZReq

Process a request for authorization.

This is one of the main feature of this plugin. Depending on the configuration, the system, will allow or deny a request.

For a specific user, if no configuration match the *RequestMethod* and the *RequestUri*, then the default action is to deny the request.

See also:

Function `authz_response()` for response authentication.

See also:

See official docker documentation.

Request:

```
GET /AuthZPlugin.AuthZReq HTTP/1.1
Host: example.com
Accept: application/json

{
  "User": "mal",
  "AuthenticationMethod": "TLS",
  "RequestMethod": "POST",
  "RequestUri": "/v1.32/containers/json",
  "RequestHeaders": "<base64 encoded string>",
  "RequestBody": "<base64 encoded string>"
}
```

Response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: application/json

{
  "Allow": "true",
  "Msg": "Authorization granted",
  "Err": "Authorization granted"
}
```

Request Headers

- Accept – application/json

Request JSON Object

- **User** (*string*) – The user identification
- **AuthenticationMethod** (*string*) – The authentication method used
- **RequestMethod** (*enum*) – The HTTP method (GET/DELETE/POST)
- **RequestUri** (*string*) – The HTTP request URI including API version (e.g., /v1.32/containers/json)
- **RequestHeaders** (*map[string]string*) – Request headers as key value pairs (without the authorization header)
- **RequestBody** (*[]byte*) – Raw request body

Response JSON Object

- **Allow** (*bool*) – Boolean value indicating whether the request is allowed or denied
- **Msg** (*string*) – Authorization message (will be returned to the client in case the access is denied)
- **Err** (*string*) – Error message. Will be returned to the client in case the plugin encounter an error. The string value supplied may appear in logs, so should not include confidential information.

Response Headers

- Content-Type – application/json

Status Codes

- 200 OK – valid response
- 400 Bad Request – malformed request
- 422 Unprocessable Entity – invalid parameters

Rtype `flask.Response`

POST /AuthZPlugin.AuthZRes

Process a response for authorization.

This is one of the main feature of this plugin. Depending on the configuration, the system, will allow or deny a request.

Warning: In the current version, we don't check any parameter, and always accept the request.

In contrast to `authz_response()`, this endpoint is called after the action has been processed by the docker daemon. The request payload contains additional fields representing the response from the daemon.

See also:

Function `authz_request()` for request authentication.

See also:

Check the official docker documentation.

Request:

```
GET /AuthZPlugin.AuthZReq HTTP/1.1
Host: example.com
Accept: application/json

{
  "User": "mal",
  "AuthenticationMethod": "TLS",
  "RequestMethod": "POST",
  "RequestUri": "/v1.32/containers/json",
  "RequestHeaders": "<base64 encoded string>",
  "RequestBody": "<base64 encoded string>",
  "ResponseStatusCode": "200",
  "ResponseHeaders": "<base64 encoded string>",
  "ResponseBody": "<base64 encoded string>"
}
```

Response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: application/json

{
  "Allow": "true",
  "Msg": "Authorization granted",
  "Err": "Authorization granted"
}
```

Request Headers

- Accept – application/json

Request JSON Object

- **`User`** (*string*) – The user identification
- **`AuthenticationMethod`** (*string*) – The authentication method used
- **`RequestMethod`** (*enum*) – The HTTP method (GET/DELETE/POST)
- **`RequestUri`** (*string*) – The HTTP request URI including API version (e.g., /v1.32/containers/json)
- **`RequestHeaders`** (*map[string]string*) – Request headers as key value pairs (without the authorization header)

- **RequestBody** (*[]byte*) – Raw request body
- **ResponseStatusCode** (*int*) – Status code from the docker daemon
- **ResponseHeaders** (*map[string]string*) – Response headers as key value pairs
- **ResponseBody** (*[]byte*) – Raw docker daemon response body

Response JSON Object

- **Allow** (*bool*) – Boolean value indicating whether the request is allowed or denied
- **Msg** (*string*) – Authorization message (will be returned to the client in case the access is denied)
- **Err** (*string*) – Error message. Will be returned to the client in case the plugin encounter an error. The string value supplied may appear in logs, so should not include confidential information.

Response Headers

- Content-Type – application/json

Status Codes

- 200 OK – valid response
- 400 Bad Request – malformed request
- 422 Unprocessable Entity – invalid parameters

Rtype `flask.Response`

POST /Plugin.Activate

Return implemented event system.

It is used internally by the Docker daemon to indicate which event system is concerned by the plugin. In the case of this plugin, it return *authz*.

See the official docker documentation.

Request:

```
GET /Plugin.Activate HTTP/1.1
Host: example.com
Accept: application/json
```

Response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: application/json

{
  "Implements": ["authz"]
}
```

Response Headers

- Content-Type – application/json

Status Codes

- 200 OK – valid response

Rtype flask.Response

GET /
Main entry point. it respond to the *GET* method for the */* uri.

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

HTTP Routing Table

Python Module Index

d

Index

M

N

P

Q

R

S

U

V