Security in VPH-Share

1 Introduction

Security is an important aspect of the VPH-Share platform and the consortium has adopted high standard technical solutions to make sure that the information provided into the VPH-Share system are stored and transferred among the different services into a secure and authorised way.

These security layers go in parallel with the control that each resource owner has on who can access his/her resources.

This document aims at providing an overview of the security layers in place in the VPH-Share infostructure from a users' perspective. More technical details on each of the aspects can be anyway found in the appropriate deliverables on the VPH-Share website[1].

The VPH-Share security is in summary based on a series of security layers and services which are then used by all the other components to maintain the privacy of the information. Here only the VPH-Share components which provides security services are described, while all the others not mentioned are user of the same services (i.e. the Data Publication Suite).

The document is divided in four sections:

- Authentication
- Policies and authorisation
- Application Level Security
- LOBCDER (FileStore)

2 Authentication

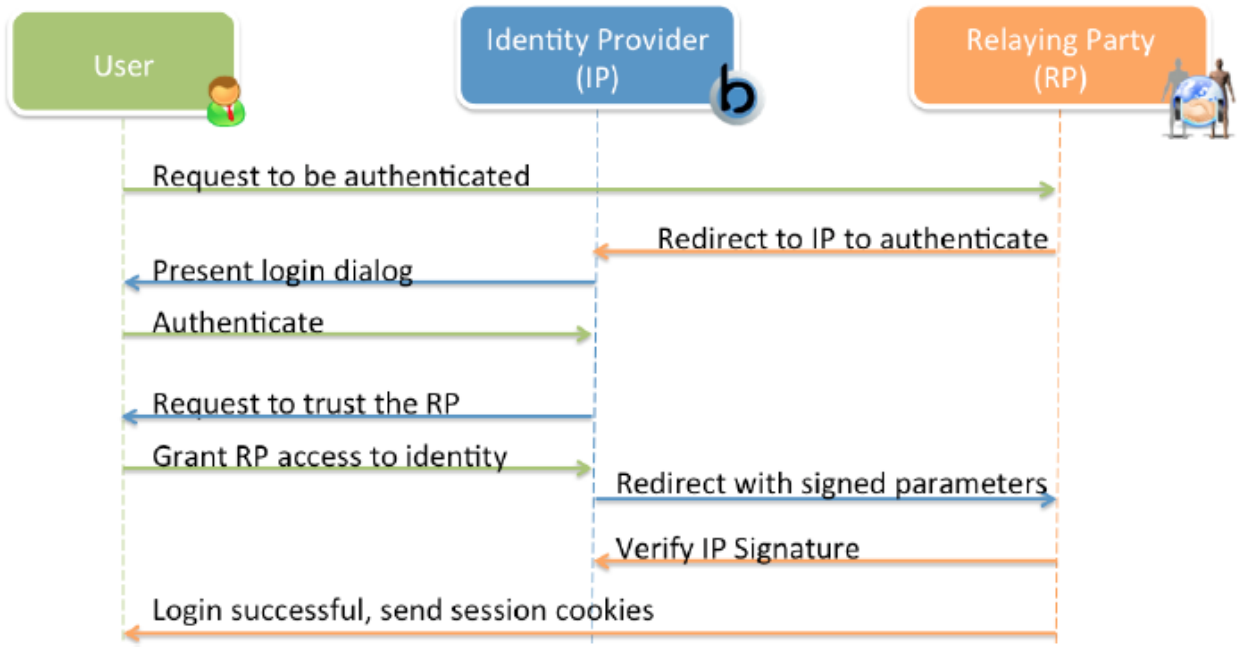
The VPH-Share portal (Master Interface or MI for short, https://portal.vph-share.eu) is the point of access for all users to the different resources and services. It is thus responsible for passing to the other services the information on the user identity and granted permissions.

Access to the Master Interface and communication are under HTTPS[2] protocol with a validated and trusted security certificate.

[1] http://www.vph-share.eu/content/deliverables

[2] http://en.wikipedia.org/wiki/HTTP_Secure

The first action provided by the VPH-Share portal is registration and log into the system. The authentication mechanism provided by the MI uses a decentralised method based on the OpenID protocol[3]. A schematic representation of the authentication method is shown below:



Schematic representation of the OpenID authentication mechanism. In this scheme the MI acts as the relying party. The authentication is demanded to an external Identity Provider in which the user's information is stored. The Master Interface relies on this Identity Provider to assign the correct privileges to the user.

The main actors of this process are:

- The User, who wants to access to the MI and needs to be authenticated;
- The Identity Provider, representing a trusted service where the User's identity is registered;
- The Relying Party, representing the service where the user wants to be authenticated (in this case the MI).

The authentication mechanism works as follows: when the user tries to login into the MI (Relying Party), his/her request is redirected to an external Identity Provider that shows the login dialog and handles the authentication process. According to this authentication, the Identity Provider assigns the appropriate privileges to the user. These privileges are sent back to the MI that will

[3] http://openid.net/

now be able to successfully terminate the login process and open a dedicated session for the user, according to his/her access rights.

Currently the only Identity Provider supported by the VPH-Share system is Biomed Town[4]. Other identity providers (e.g. Google) can be added in the future if appropriate.

If the user is recognised as a valid one, a session is opened into the Master Interface and a valid ticket is generated. The authentication ticket holds all the information about the user and is signed by the Master Interface to prove its authenticity. The ticket is structured as the follow:

uid=<username>;validuntil=<expire-time>;cip=<ip>;tokens=<roles>;***udata*=<username>,<fullname>,<email>,<language>,<country>,<postcode>**;sig=<ticket signature>

- uid: username of the user;
- validuntil: timestamp indicating when the ticket validity ends; at present, this is set to 12 hours from its creation; after this time it is refused as invalid; in case of longer processes, services are provided to automatically regenerate the ticket and not block the execution;
- cip: IP of the client which generates the ticket;
- tokens: the roles and all the permissions assigned to the user. The permissions are used in particular by the security proxy (see later sections for more details on this), and are in the form <nameresource>_<typeresource>_<permission(admin/read/edit>);
- udata: it contains information on the user;
- sign: this is the ticket signature that is generated from the MI with a private key DSA 2048 bit. This is used by all the other services to verify the ticket validity.

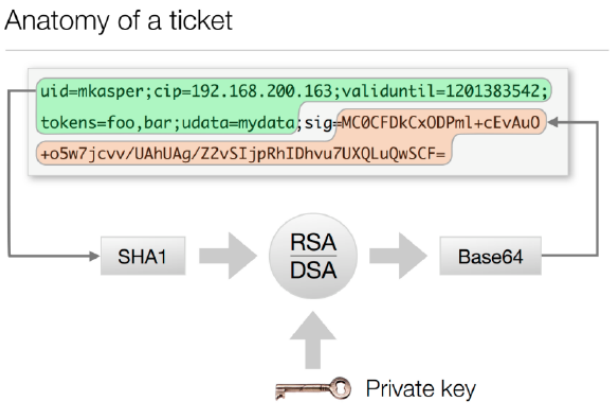


Figure 1: Ticket generation

[4] http::://www.biomedtown.org

The authentication ticket is based on the Apache mod_auth_pubtkt[5] technology. The mod_auth_library5[6], a Python package to support the ticket creation and validation, has been developed and integrated into the Master Interface. The ticket is signed with the DSA Algorithm5[7] with a 2048 bytes key. The ticket is then compressed in base64 to make easier the communication of different systems.

Anyone (or any service), who has the below public key and a ticket, can verify it and if valid it can open a session being sure that the information contained in it are safe and generated from the MI.

-----BEGIN PUBLIC KEY-----
MIIDOzCCAi4GByqGSM44BAEwggIhAoIBAQDYHe5/lDkW2i2bJTx4jcMERgL477G+
T13a2KhOw3ld17asktxZtvlJsxEpkdlTUTe2FFdssrcjQ+bVytBJgUf3KjfK7rhK
SuVLzNC09Z3vjhr983WrBG7or7r/hKjPlgaLsNNS1GVjHEHS+jurGaE+7LhnXmfu
Z4Ly5wA2NQdbp1AbcQ6CjqXdtKYw+7MQdj3cacM1PArWhnVDdPC02TwZJA7ae4O1
wbPcUQmxtIMw3FYR5emjbz2C98VUvUdI6FbS1QdV/ZLPDP3j0IRcdQ/RrnyMLtbN
32p+H4xdrHbDQOnoRWhGhxcNp/k/xgJtHx2z+/lpbtUyCqPVOwsq6cO9AhUA+ySB
y+jH9iSi/TITIISmarwuA0ECggEBALilFPDXQTT4UpxkLer46KV0cnoek5AbNMFh
fKdm/E2P9CzBgHLk/Q1UiZRou7rRKvtvond4d7CeTK2XVa7uQM0Rbg1O7ABbczvh
a04dWggGAHr5rEZk6OBPzgW++YwvLEisF9f2vTufzWhAgOMUYWSI+joc/lLjFOuf
TNLey2s7bjELP7nA6TmGrCtR/XOliLPjloA9O5TSjYDHLMCq9r+TGQEm51dh2Tfs
KQnkbu/0It6ECr0t/G+5chJin5R0x5qO/yqdywEzhhNwd+G0eMcY6B/aJwNn/wNe
zyuMenTyPwoFZHm4PtXjyuz0+41n39v0V5Rhc3VC9R27t0wa1hsDggEFAAKCAQBZ
aQZsWveEVgi73QL8qb+9b+EeG3GPEM1H5AOxOqq+rPRHS2+dJjiDvUeZD/cDNW+c
EYv949skpIrZkthDiRWxaf6ZmhAB66mg4dmgLJWtN61lrYzD8n1RacH0HmBe20R8
DL+UKBoyRllB27IVLbFGCTl77jYsDxP6Q7uL4koJOkN2FtYJCxqOMGAfhIqbtibg
WJm1CQAKYb0mON+rTonOwzoK8GHtzqXtkbeY5HbBSOdiOHJCjtfFZDEZS0FaZXT+
OfULRFdOouooIdiQyNxKsId/pkL6hBXL6QvVzfVaGiE5nhWskgtmOsKanWWLeGtR
sv5tHXp48zLsDtXeFncp
-----END PUBLIC KEY-----

At the Master Interface level the same ticket is saved in a cookie, named *vph-tkt,* which is validated at each request of a page. In fact, the MI provides a service that allows the ticket consumers to validate the ticket and to retrieve a JSON26 notation of the user attributes. In accordance to the current regulations, the user will have to explicitly accept the use of cookies for the correct functioning of the system.

The validation service receives the ticket to be validated as an URL parameter. If the ticket is valid, a JSON notation of the user information is returned and the following one:

[5] https://neon1.net/mod_auth_pubtkt/

[6] http://pypi.python.org/pypi/mod_auth_library/1.0

[7] http://en.wikipedia.org/wiki/Digital_Signature_Algorithm

{
"username": "mbalasso",
"language": "",
"country": "ITALY",
"role": ["developer", "friend"],
"postcode": "40033",
"fullname": "Matteo Balasso",
"email": "m.balasso@scsitaly.com"
}

If the given ticket is not valid, the HTTP 403 status code is returned.

If a user needs a copy of his/her own actual ticket (for example in the deployment of Applications/Atomic services) he/she can copy it into the clipboard by using the button available in the user profile page (https://portal.vph-share.eu/profile/).

3 Policies and Authorization

The underlying philosophy of the security layer is to extract the security measures from the service logic, ideally making the security transparent for the service itself.

The system provides the ability to define access criteria for services based on the attributes assigned to the user and the execution environment. A security policy is a structured document that defines these security criteria. These policies can be as simple as a role-based mechanism or as complex as a combination of policy rules depending on more complex attributes.

An example of a simple role-based policy could be: "**If** the user **has the role** *paediatrics* assigned, **then** they should be granted access to the Paediatrics database service".

An example of a complex policy rule could be: "**If** the user **has at least** a bachelor's degree in medicine **and has** signed a confidence agreement **then** grant access to all medical stores".

Security policies can be a very powerful mechanism for developers to express accessibility constraints via rules for services. However, there is no way with which the VPH-Share platform can know them beforehand, as they are specific for each service. As a result, it is responsibility of the application developers to infer these criteria from the application requirements, the user attributes provided by the Master Interface and the information contained in the request.

The administration of the security system is integrated with the Atmosphere Internal Registry via the Master Interface, allowing application developers to update the policy rules associated with their services (even already instantiated ones). It will therefore be possible to maintain and update the access criteria for services without the need to define a new service template in each case.

To express policy rules the project has chosen the OASIS standard language XACML (eXtensible Access Control Markup Language)[8].

3.1 Security for Atomic Service invocations: the Security Proxy

The Security Proxy is a service that allows enforcing the XACML policies defined in the Master Interface. For securing both an atomic service and non-LOBCDER data resource (LOBCDER comes with its own security mechanism, see later section for this), it is necessary to have the security proxy installed and properly set up on the template that will be used for instantiating your services. The following sections explain how to install and configure the security proxy on an empty template from the project repositories.

3.2 Setting up and configuring the security proxy

3.2.1 Installing the proxy distributable package in an ASI template

The current version of the security proxy comes as a Linux standard deployment package, let it be either a Debian or rpm package for Debian or RedHat derived Linux distributions, respectively. Both of them can be downloaded directly from the repositories of the project and are properly configured with their prerequisites (Java version 7 or higher).

Hence, the normal procedure for setting up a template with a service with the security proxy is to start a proper Atomic Service Invocation (ASI) following the procedures in the user manual for 'Creating a new Atomic Service' and then issuing the proper installation command for the chosen Linux distribution, let it be:

- 'apt-get install vph-secproxy' for Debian and derived distributions
- 'yum install vph-secproxy' for RedHat and derived distributions

And this will install both the security proxy and his dependencies (Java7) in the system.

3.2.2 Creating proper policy files/roles

The security policy secures services by executing certain policy files against the user attributes retrieved from the identity platform. This is true even for role based access mechanisms, because roles are stored in the system as security policy files, in XACML format. Hence, regardless the security of the service is going to be based on roles or more complex security files, their corresponding XACML files must be present in the system first. In the case of role based access control, the system comes with a tool for creating this policy file automatically for the service administrator without needing anything about XACML, but it is also possible to provide the XACML

[8] http://en.wikipedia.org/wiki/XACML

content for the service itself. The policy files of the system can be managed through https://portal.vph-share.eu/security/ in the Master Interface. Please note that users are allowed to manage only the policies they own, but they can currently see all policies in order to be able to reuse existing one.

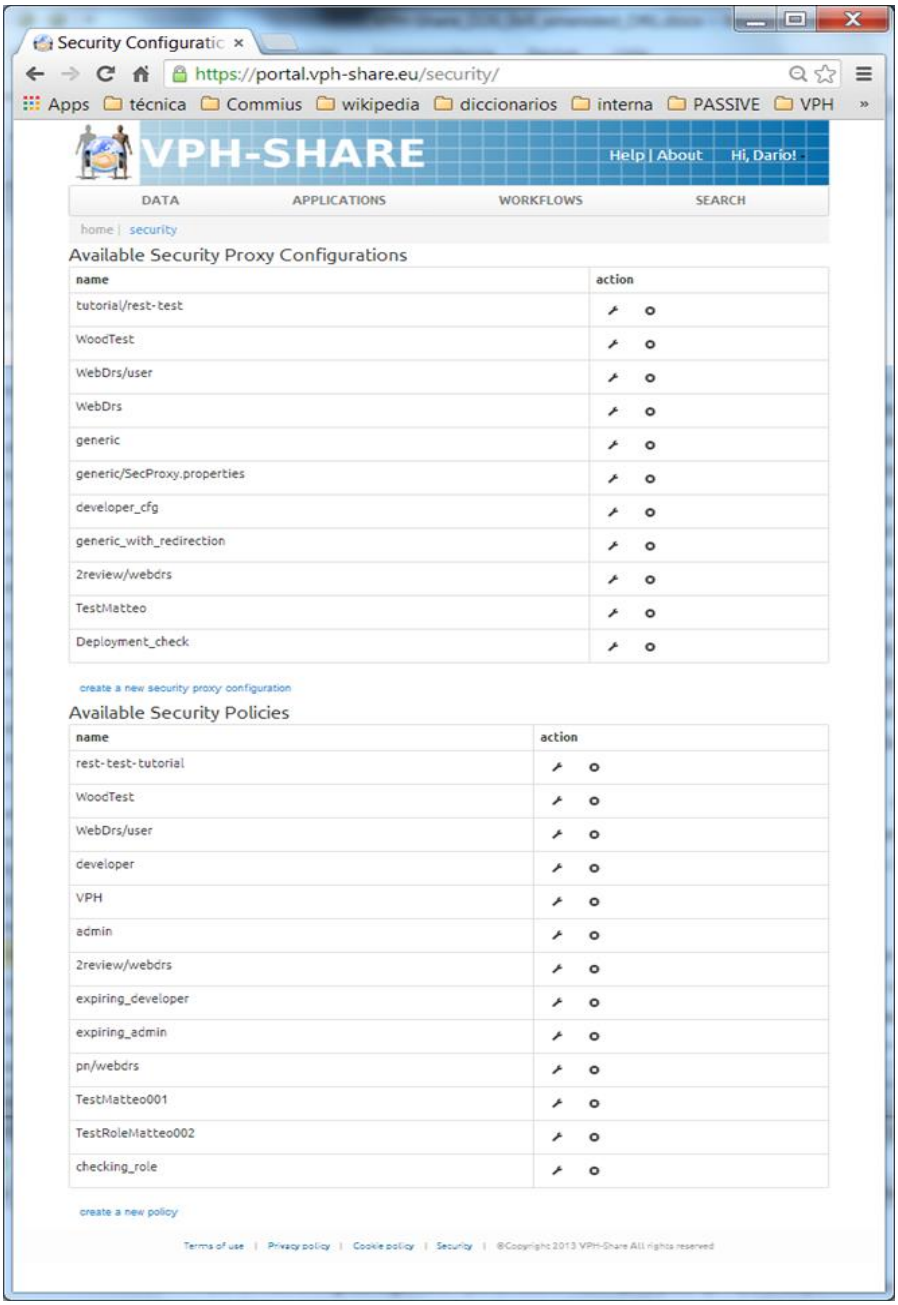


Figure 2: Security administration page

This page has two main sections: one for the management of the policies present in the system and another one for configuring the service. In the 'Available security policies' section, it is

possible to create new policies or to modify existing ones. There are basically two ways to create or modify a security policy: one for naïve users and another one for XACML expert users. The advantage of the mode for naïve users is that it is much easier to use, without any need for any knowledge of XACML at all, but it is far less powerful than the expert mode: the naïve mode allows to set only a predefined set of security conditions for the policy, while the expert mode allows to directly provide the content of an XACML file. In fact, it may be a good idea even for expert users to start by creating a new policy with the naïve mode and then completing it editing the generated XACML file directly. There is a link 'Create a new policy' for creating a new policy at the bottom of the 'Available security policies group':

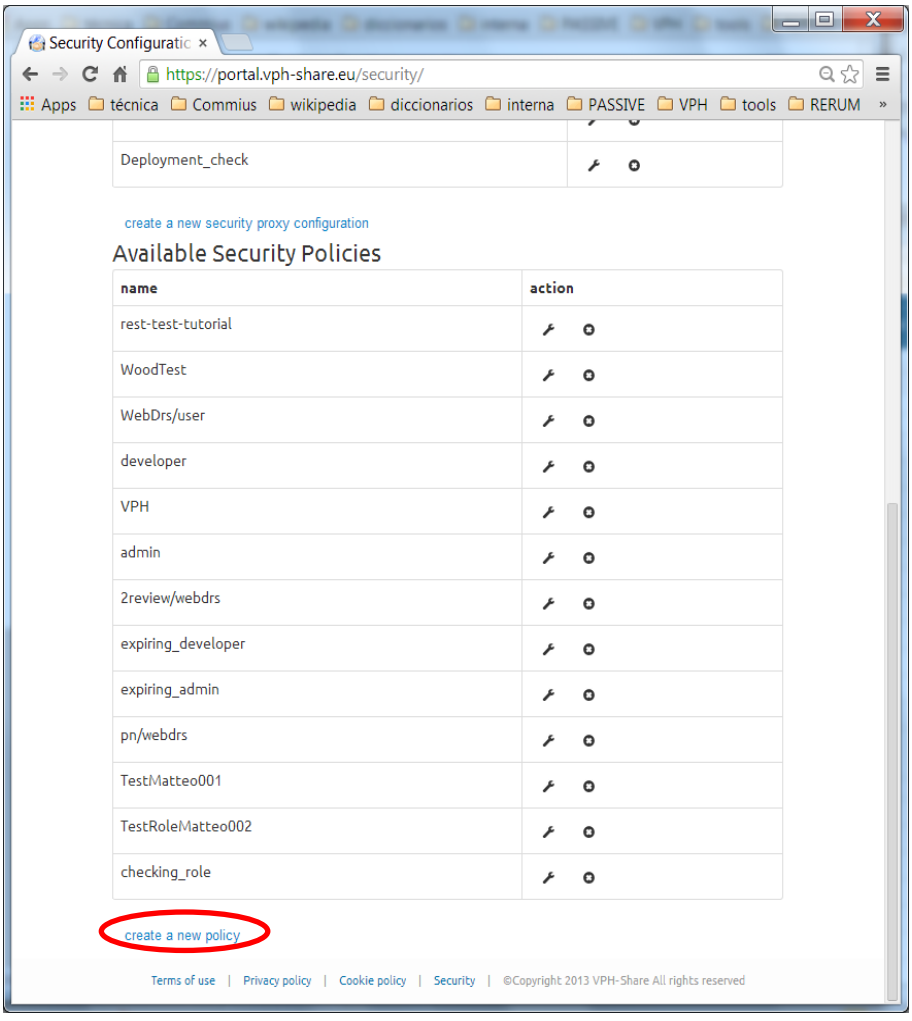


Figure 3: Create a new policy option

When this option is selected the 'Policy editing' page is displayed. The naïve mode currently deployed in the Master Interface only allows creating policies for Role Based Access Control (RBAC), that is, creating a policy file that supports RBAC for a given role, but a more advanced one is planned for implementation in the.

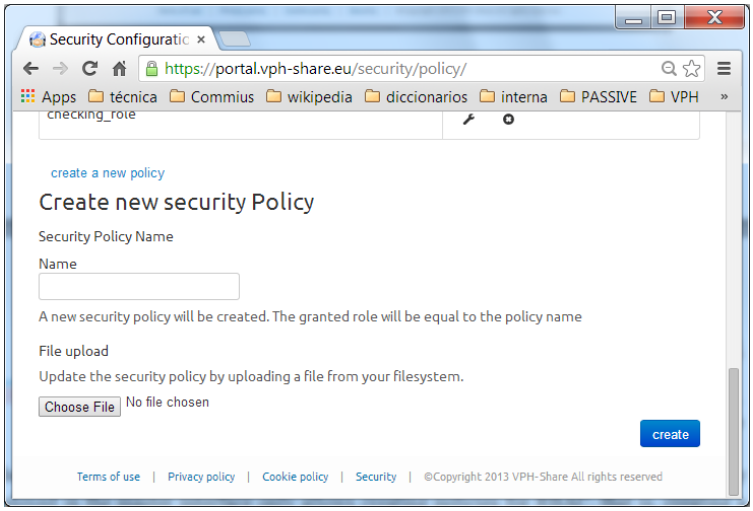


Figure 4: Creating a new security policy

As it may be seen in Figure 4, it is possible to create a complete RBAC policy file by simply providing the name of the role supported, which should be the name of one already declared in the identity platform. However, the upcoming new version of the MI will display a combo box showing only the roles already set up in the identity platform. Alternatively, it is possible to use the advanced mode by directly providing an XACML file clicking on the 'choose file' option. Once the XACML file has been chosen or the name of the new role has been provided, the new policy may be created by clicking on the 'create' button.

It is also very important to note that it is possible to reuse policy definitions from one service to another as long as they are already present in the system. This is mainly achieved by taking advantage of the cloud path of the policy. Each policy must be created specifying a complete cloud path name that behaves in a similar way as a directory path name, which, in this case, is located in the cloud facade of the project. As such, the name is supposed to include a folder path (possibly empty) of directories split by a '/' character and finished by the name of the resource itself. For instance: sheffield_hospital/cardiology/heart_attack/heart_attack_user might refer to the role heart_attack_user of the service heart_attack of the cardiology department of the Sheffield hospital. But it is also possible to use this mechanism to create global and intermediate roles. For instance, the role VPH refers to a global role for all VPH users, and the role /sheffield_hospital/sheffield_hospital_staff could refer to the role sheffield_hospital_staff of the Sheffield hospital. In fact, it is VERY advisable that each institution creates its own roles only under their own folder to avoid mixing all names.

In the given page you may either create a new policy file or modify an existing one, if it was already created. The first step is to specify the complete cloud path name for the policy. It is not mandatory, but extremely advisable that the roles and policy names follow the structure <cloud path to the resource to be protected>/<role name> as in the previous examples.

If the resource did not exist yet, you may create or modify an existing one by any of these ways:

- Entering the XACML content in the 'Security Policy file content' text box and clicking on its corresponding 'Update' button to its right
- Choosing a file to upload on the 'Upload file' button and clicking on its corresponding 'Update' button to its right
- Simply click on the 'create' button and the system will create and upload a new policy file for a role named as the box 'Name'

3.2.3 Creating a configuration for the security proxy

To configure the security proxy it is necessary to provide it with a configuration in the cloud facade. This configuration can be either an already existing one (possibly a global configuration) or another one created specifically for the service.

To create a new configuration for a service, access https://portal.vph-share.eu/security/ and click on the 'create new security proxy configuration' link:

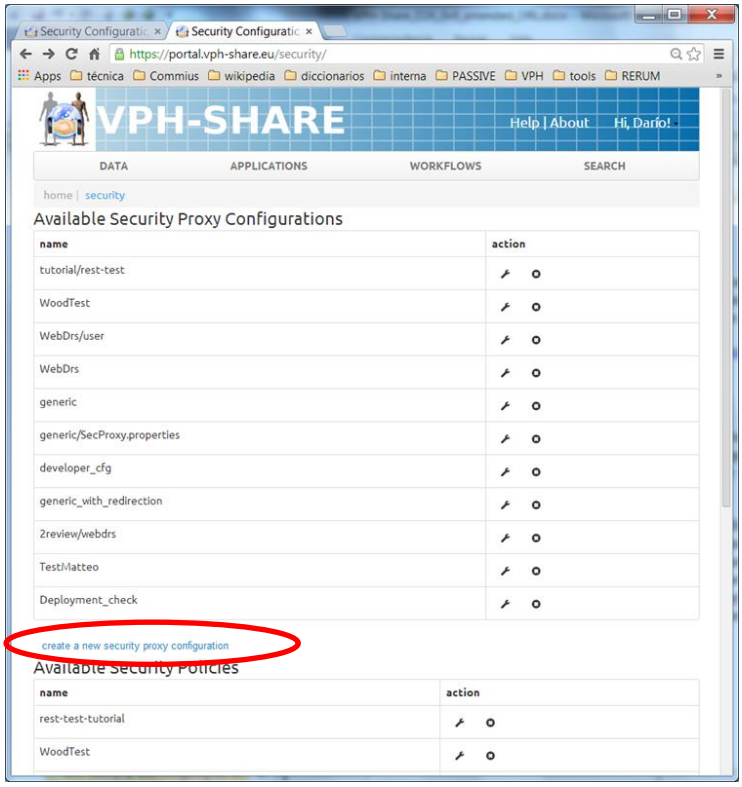


Figure 5: Choosing to create a new security proxy configuration

After clicking the link, the page will change to display something like:

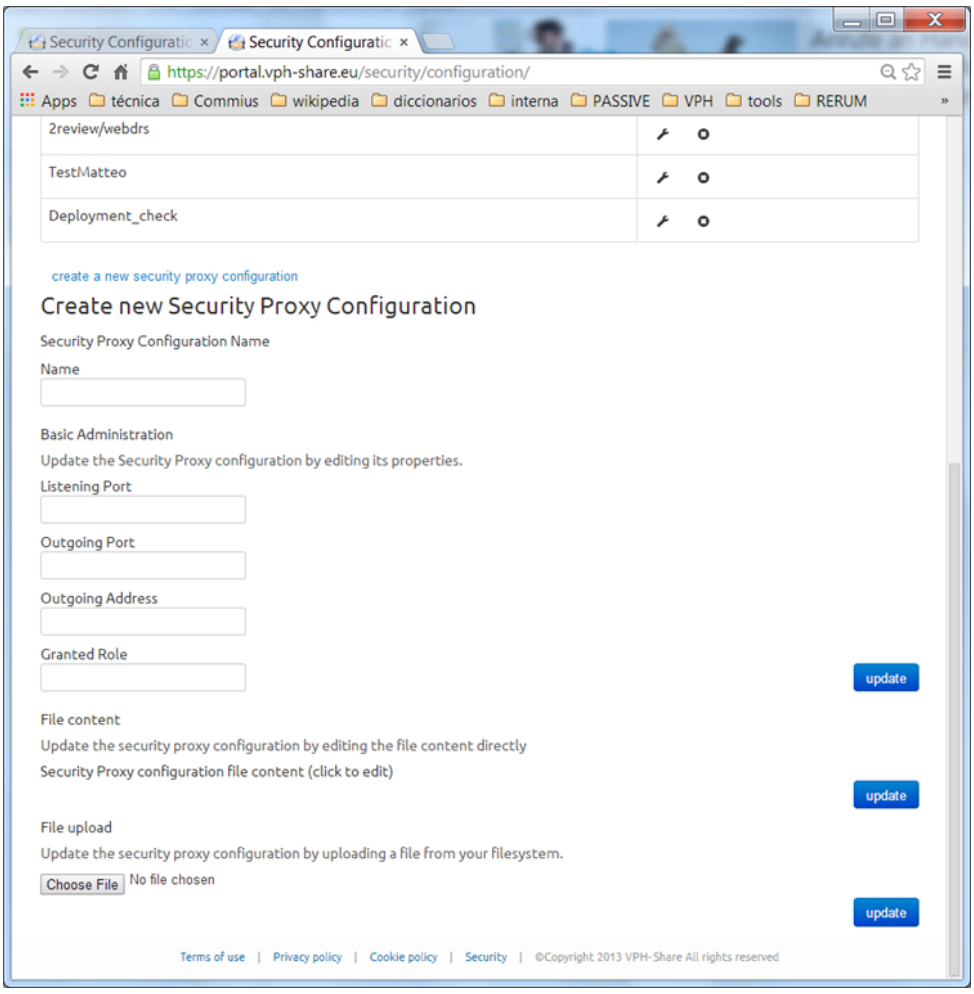


Figure 6: security policy configuration page

In the text box 'Name' a complete cloud path name to the configuration should be inserted and saved for reference. This cloud path name is the link between the security proxy installed on the Atomic Service and the cloud facade and will therefore be used later for configuring the atomic service security.

The path name allows defining a hierarchical tree structure for the configurations. This way, it is possible to define global and semi global configurations meant to be reused and local configurations meant to be specific for a given service. For instance, there is a predefined configuration named 'generic' that is assigned by default to any new atomic service created in the system. But for example, the University of Sheffield could create a uni_sheffield/general configuration for all their services or a uni_sheffield/cardiology/heart_break configuration for a heart_break service of the cardiology department.

Besides providing the complete path name, there are two ways for specifying the configuration parameters for a given service: a basic but limited one, and a more complex and complete one.

In the basic configuration, the user simply enters several values in some text boxes that will be used for creating later a configuration file. If the user does not enter any of these values, then the default ones will be used. The following fields are available:

- Listening port: The port where the service will be listening for requests, or, more exactly, the port where the security proxy will be listening for intercepting incoming requests that will be redirected to the services in case they are granted. A common valid could be 8081 for intercepting incoming https requests;
- Outgoing port: The port that the security proxy will redirect the requests that are granted access. A value frequently used for this is 8081;
- Outgoing address: The address of the machine that will be hosting the service to be secured. This will be normally the loopback address (127.0.0.1) but this is not mandatory;
- Granted role: a list of roles that will be granted access to any resource installed on this atomic service. This roles must have been already uploaded to the system as explained in the previous section.

In the complete mode, the user may provide a standard properties file or type its content that will be used later for configuring the system. A configuration file is basically a list of associations in the form property=value. This configuration file should include an entry granted_roles with a list of the form <regular_expression>:<comma_separated_role_list> separated by the ';' character. Valid examples of this could be:

- granted_roles=my_app1/resource1:role1
- granted_roles=my_app1/resource1:role1, role2
- granted_roles=my_app1/resource1:role1, role2;my_app1/resurce2:role1
- granted_roles=my_app1/resource1:role1, role2;my_app1/resurce2:role1;my_app2/resource1:global_role1
- granted_roles=^[1].)*$:2review/webdrs,developer;.*/admin/?:admin

There are 'update' buttons for each of the configuration methods (typing the values in the text boxes, typing the configuration file content directly or loading the file). Click on the one that corresponds to the way you have provided the configuration and the configuration will be created / updated.

3.2.4 Linking the security proxy to the cloud configuration

Once you have prepared a valid configuration in the cloud facade through the MI, you are ready to link the security proxy to it. Enter the Atomic Service and execute: /sbin/AgentConfigurator.sh. The program will ask for the name of the complete cloud path name of the configuration that was created in the step before. Type it in there and press intro. The program will ask for confirmation of the information entered.

Each service must have a 'properties' configuration file that is named SecProxy<service>.properties where <service> can be any value, including nothing, and must be

located in the directory /etc/vph-secproxy. In fact, the default configuration file is SecProxy.properties.

The security proxy comes shipped with a default configuration file that should be normally enough. In fact, the configuration file is upgraded with the contents retrieved from the cloud facade according to the cloud path specified when executing /sbin/AgentConfigurator.sh, but note that the configuration file includes additional properties that cannot be managed from the cloud facade because they are currently not meant to do so, as they are implementation specific ones. The only parameter that is foreseen to be of interest for developers is the 'socket_timeout' one, which specifies the time the security proxy will wait for a response of the service before closing the connection.

3.2.5 Setting the proxy for multiple end points

In case you want to have your security proxy protecting multiple end points, there is a different procedure to be followed. The security proxy is prepared to run multiple worker threads against different end points, but it needs a proper configuration for each of these. Hence, to configure the proxy to protect many end points, you need to specify the proper cloud paths to the AgentConfigurtor.sh script. This is done by executing:

/sbin/AgentConfigurator/sg <list of complete cloud path configuration names> where the list of cloud path configuration names must be split by the " " character

Once the proxy is installed, it is necessary to tell it the name of the service that it is meant to protect. Though it is possible to perform this task by directly touching the configuration files of the agent, there is a tool explicitly created for this. Once the security proxy is installed, it is enough to run

'/usr/sbin/AgentConfigurator.sh'

And it will start the tool. The tool will ask for the name of the service in the cloud façade.

4 Application level security

There are two ways to achieve security at application level: through the policy mechanism and defining it in your application. In this section we will explain a little further how to achieve this in both cases.

4.1 Defining security at application level with the policy mechanism

Current version of the security proxy allows two ways of working: a simple one for naïve users and an advanced one for engineers who are experts in XACML.

4.1.1 Simple mode

The simple way provides a RBAC (Role based access control mechanism): you need to create a role that will have access to a service and assign that role to the users that are meant to access that service. For instance, if you create an oncology simulator only for the oncologist doctors of the Sheffield hospital, you may create the role 'Sheffield/Oncologist' and assign that role to all the oncologist staff of the Sheffield hospital. Next, you configure the security proxy of the service to accept the role 'Sheffield/Oncologist'. Alternatively, you may take advantage of an already existing role at a higher level. For instance, if you have a checking service for all the staff in the Sheffield hospital, you may have the already existing Sheffield/staff role and reuse it for this service. In any case, It is not necessary to have any knowledge of XACML at all to use this naïve mode, because the system is able to create a proper XACML policy for you.

Future versions of the system will provide an advanced mode that will allow naïve user with no XACML expertise to create more powerful policies and the most common selection criteria as well.

4.1.2 Expert mode

Alternatively, XACML expert engineers may provide a complete XACML policy themselves that will be evaluated by the system as long as it is based in the attributes issuing the request and the system attributes, knowing that:

- The role attribute will be included in the XACML request under the name 'role'
- All the values of the user attributes defined in the identity platform will be included in the XACML request as attributes whose name will be uData<order> where order is a number indicating the order that the attribute is reported by the identity platform
- System attributes such as current date and time are accessible as environment attributes

For more details about how to create the policies with the MI and setting up the security proxy, refer to section 'Installing the proxy distributable package in an ASI template'.

4.2 Defining security in the application

The standard way to define the security in VPH-Share is through the policy system, whose intention is precisely to decouple the application from the security itself. However, there is still one situation in which a developer might be interesting in defining the security partially or completely in the application.

This situation is when the decision to grant access to a given resource depends on the logic of the service, and this service relies partially or completely on some additional information that is nor provided by the identity provided nor is included in the system attributes (date and time). The current version of the security proxy is not able to provide other information than the mentioned user attributes and system date and time, and in this situation it may be necessary that the service wants to access the information contained in the request to make the decision.

As explained, the security proxy is meant to work in a way transparent for the service. As such, the service may access the information of the request exactly in the same way as if the service were not protected by a security proxy. But besides that, the service may also access the security token described in the first section to complement the original request information with the one provided by the identity provider.

In any case, the future versions of the security proxy are foreseen to dump all the text information of the original request in the XACML request used to make the decision, thus allowing making decisions also based on the business logic, even with the naïve mode.

5 LOBCDER (FileStore)

To protect against eavesdropping LOBCDER is running over SSL.

To protect against unauthorized access LOBCDER uses the following authentication mechanisms:

- Local: Accounts can be stored on a local DB
- Local: An authentication ticket provided by Master Interface is passed to LOBCDER. LOBCDER can verify the ticket against a public key. Using this mechanism LOBCDER is independent from the ticket validation service and therefore has improved it is response time by removing the network's overhead and the service's response time. It is also became more secure against eavesdropping since there is no need to send authentication tokens to an external service.

To solve the problem many WebDAV clients have with long credentials (many WebDAV clients including davfs are unable to use credentials longer than 256 characters), LOBCDER provides a REST transaction service. This service is able to produce short aliases and map them to long security tickets. These aliases, are short-lived.

To handle authorization and permissions LOBCDERs resources have Access Control List (ACL). ACL consists of the list of the roles that are allowed to access the resource and permissions set by the resource owner for these roles. A request for a resource is considered authorized if the user belongs to a specified role and this role is allowed to perform the requested operation (read or write), or if the user is an owner of the requested resource. Here “user role” is a synonym of the user’s group.

This ACL is kept in an internal metadata table. A resource may have as many roles in its ACL as needed. To manipulate the content of ACL, LOBCDER provides a set of REST services:

- GET https://host.com/lobcder/rest/item/permissions/{uid} Returns resource permissions: owner, read, write
- PUT https://host.com/lobcder/rest/item/permissions/{uid} Sets resource permissions for owner, read, write
- PUT https://host.com/lobcder/rest/items/permissions?path={path} Sets permissions for folder and subtree in a bulk mode