Deliverable D3.3

Tools for the integration of confidential data

Grant Agreement	Health-F5-2008-200787
Acronym	OpenTox
Name	An Open Source Predictive Toxicology Framework
Coordinator	Douglas Connect

Contract No.	Health-F5-2008-200787	
Document Type:	Deliverable Report	
WP/Task:	WP3 / D3.3	
Name	Tools for the integration of confidential data	
Document ID:	OpenTox Deliverable Report WP3-D3.3	
Date:	June 30, 2010	
Status:	Final version	
Organisation Responsible:	IST	
Contributors	Andreas Maunz Luchesar Iliev Roman Affentranger Barry Hardy (review)	IST IDEA DC DC

Distribution:	Public

Purpose of Document:	To document results for this deliverable

Document History:	1 – Produced by Andreas Maunz (IST)
	2 – Updated 100319 by Andreas Maunz (IST)
	3 – Updated 100405 by Andreas Maunz (IST)
	4 – Updated 100515 by Andreas Maunz (IST)
	5 – Updated 100527 by Andreas Maunz (IST)
	6 – Updated 100611 by Luchesar Iliev (IDEA)
	7 – Updated 100617 by Luchesar Iliev (IDEA)
	8 – Updated 100618 by Andreas Maunz (IST)
	9 – Revised 100629 by Roman Affentranger (DC) 10 – Final version 100630 by Barry Hardy
	11 – Revised 110310 by Andreas Maunz (IST)

Table of Contents

List of abbreviations

AA	Authentication and Authorisation
AAA	Authentication, Authorisation and Accounting
AAI	Authentication and Authorisation Infrastructure
API	Application Programming Interface
FOAF	Friend of a Friend
LDAP	Lightweight Directory Access Protocol
PGP	Pretty Good Privacy (cf. OpenPGP)
PKI	Public Key Infrastructure
REST	REpresentational State Transfer
SSL	Secure Sockets Layer
SSO	Single Sign-On
TLS	Transport Layer Security
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
WAR	Web Application aRchive (file format)

Summary

OpenTox strives to produce an open framework that allows access to a wide audience of toxicology and chemical experts, model and algorithm developers. While many of the resources provided within the framework will not require any special restrictions, the possibility to integrate confidential in-house data is deemed an important functionality as well. Just like any public resource on the internet, this information needs to be well protected from unauthorised access, which requires the implementation of an authentication and authorisation infrastructure. This infrastructure deals with "who" has the right to do "what" with the confidential data. It involves confirming the identity of the user who requires access (authentication) and then putting this confirmed identity against a set of restrictions to determine whether the requested access should be granted or denied (authorisation). The restrictions themselves are defined through different access control policies. Additionally, the system might log all access requests (accounting) for different purposes: management, billing, security, etc.

We have considered different AAI solutions and have chosen OpenSSO – a single sign-on authentication and authorisation server by Sun Microsystems – for an initial implementation. A policy configuration service has also been developed in order to define and manage the access control policies. Finally, a common user and policy database has been established.

Because the OpenTox Application Programming Interface (API) is based on the Representational State Transfer (REST) architecture, implementing a complete AAI is not a trivial task. While REST is becoming increasingly popular over the Internet, the security technologies to support it are not well established yet or a subject of current research. For this reason, the solution presented in this deliverable is considered merely an initial step to investigate the problem and gain important experience. It is aimed to be a current working and usable solution, but there is certainly scope for future extensions, including the implementation of entirely new concepts.

1. Introduction

General overview

Much of the success of the internet is due to its inherently open, decentralized nature, which allows any resource, made available through it, to be reached conveniently on a global scale. When these resources are of sensitive nature, however, this inherent openness can become a serious problem.

Confidentiality has been defined by the International Organization for Standardization (ISO) in ISO-17799 as "ensuring that information is accessible only to those authorised to have access" and is one of the cornerstones of information security. The concept could be further broken apart into defining "who" has the right to do "what" with certain data. This leads to two tasks that need to be handled:

- **Authentication**: Confirming the identity of the user who is requiring access to the confidential information (that is, confirming that they are indeed who they claim to be.)
- **Authorisation**: Putting this confirmed identity against the set restrictions to determine whether the requested access should be granted or denied.

Authentication works by binding an identity to one or more so-called "factors". These factors are either something that the user *has* (a security token, ID card, debit card, etc.), the user *knows* (e.g. a password, PIN code) or the user *is* or *does* (such as fingerprint, retinal pattern, DNA sequence, signature, face, voice, etc.). When a user requests authentication they must provide these previously bound factors for verification, e.g. they must show their ID card, type in their password (arguably the most widespread factor over the internet), submit to a retinal scanner, etc. Once the claimed identity and the factor match, the user is considered authenticated and can proceed further with authorisation.

The authorisation process is based on policies or rules that specify the access rights to the different resources. Each policy may allow or deny access to a certain resource based on:

- User identity: access is granted only to specific users (or to specific groups, in which case the user must be member of the said group) or, conversely, specific users or groups are disallowed access. AND
- Type of access: usually, at least "read" and "write" access are differentiated (the latter effectively also being "delete"), but depending on the specific data being controlled other types could be defined. In the context of the REST services the types would match the four methods: GET, POST, PUT and DELETE.

Other conditions might be set as well (most often time restraints), and all these parameters are matched together. So, for instance, the policy for the resource "Bob's work files" might allow both read and write access to the user "Bob", but allow only read access to user "Bob's boss" and deny all access to the user "Bob's wife".

The whole process is divided into two phases: 1) policy definition phase, and 2) policy enforcement phase. The policy definition usually (but not necessarily) coincides with resource creation (or it becoming available online), while the policy enforcement happens when access to the resource is requested.

Because of the close interrelations between the authentication and authorisation processes, they are typically handled by a common infrastructure, referred to as "Authorisation and Authentication Infrastructure" or AAI. Closely related is also the process of "accounting", which provides means to keep log of all access requests, whether successful or not, either for billing or for security purposes.

OpenTox context

OpenTox strives to produce an open framework that allows access to a wide audience of toxicology and chemical experts, biologists, model and algorithm developers. While many of the resources provided within the framework will not require any special restrictions, the possibility to integrate certain in-house data of a confidential nature is deemed an important functionality as well. Just like any public resource on the internet, that information needs to be well protected from unauthorised access, which requires the implementation of an AAI solution.

The AAI should handle the following tasks in the OpenTox context:

- Maintain a database with information about the users such as: identity (user name), real name, organization, address, e-mail, phone number, etc., together with their authentication factors (most likely a password, but probably also PKI certificates or other cryptographic keys or tokens). User groups should also be supported, allowing the users to participate in one or more of them.
- Provide an interface for new users to register in the database and for the existing ones to maintain their records and authentication factors. Additionally, an administrative interface should be available that will also allow the maintenance of the user groups.
- Provide a mechanism for the users to authenticate themselves before the OpenTox services. This mechanism should be secure enough, so that no sensitive data (e.g. user's password) is leaked.
- Maintain a database of access control policies for all available resources: datasets, algorithms, models, etc. Because of the REST implementation, each resource is, in effect, a Universal Resource Identifier (URI), which also defines the types of access available: the HTTP methods GET, POST, PUT and DELETE.
- Provide an interface for the users to define policies for their newly created data:
    - select specific users and/or groups from the user database
      AND
    - set the appropriate permissions (allow or deny) for the four methods.
- Provide a mechanism to create the policies based on either users' preferences or preset default values.
- Provide an interface for the users and a mechanism to modify the existing policies.
- Provide a mechanism for the OpenTox services to match an access request of an authenticated user to the policy database for authorisation: that is, to receive an answer whether the user is to be allowed or denied the requested access.

Because the OpenTox API is based on the Representational State Transfer (REST) architecture, implementing such AAI is not a trivial task. While REST is becoming increasingly popular over the Internet, the security technologies to support it are not well established yet or are even still research topics.

We have considered several approaches to implement an AAI:

1. Oracle Corp.'s (formerly Sun Microsystem's) **OpenSSO** and its community-based fork **OpenAM.**
2. **RDF metadata** access control lists, attached to the resources, and based on the **FOAF+SSL** authentication mechanism.
3. a **PKI- or PGP-based** solution using asymmetric cryptography to both protect the resources from unauthorised use and enforce access control policies.

The 2nd and especially the 3rd approach looked rather promising in keeping up with the spirit of the REST architecture, particularly in a widely distributed environment. However, they are still in early stages of development and the required effort to implement a solution based on any of them was found to be beyond the available resources, especially when keeping in mind that OpenTox is not specifically oriented towards information security.

For this reason, OpenSSO and OpenAM emerged as the best current solution when all pros and cons were taken into consideration. They are easy to set up, and allow REST calls to be implemented for authentication and authorisation as well as for policy management, and may be easily integrated into the current API. Paired with an LDAP directory, they can also be easily connected with the already existing database of opentox.org users.

OpenSSO/OpenAM

OpenSSO is a single sign-on authentication and authorisation server which was developed by Sun Microsystems (now part of Oracle), which would, however, require a custom REST policy web service, due to currently missing support. REST interfaces are of increasing importance. Quote from the OpenSSO developers (1):

> *"The recent rapid advancements and adoption of Web services, service-oriented architecture (SOA), and Representational State Transfer (REST) architecture within enterprises have left the industry wanting more. Organizations and developers, such as those who focus on Web 2.0, are demanding interface support from identity and access management software. The Open Web SSO Project, called OpenSSO for short, answers those demands."*

OpenAM is a community-based clone of OpenSSO. OpenAM maintains 100% compatibility to existing OpenSSO versions.[1] It is maintained by Forgerock, a Norwegian software company with a focus on open source enterprise software.[2] There is also a wiki available.[3] The OpenSSO Users Mailinglist[4] is very active and provides contact to experts in the field of authentication and authorisation.

OpenLDAP

OpenLDAP is an LDAPv3 compatible directory server intended to hold identity information. OpenSSO can attach a variety of such databases in a very flexible manner, including Microsoft's Active Directory, and generic LDAP servers. We use OpenLDAP as a common backend to the PLONE Content Management System (CMS), on which the opentox.org site is currently based, and OpenSSO.

The article series *Securing Applications With Identity Services*[5] describes in detail the setup of an OpenSSO deployment, including client application. Installation of the server is very simple - it is only necessary to extract and start the web server, as well as to deploy OpenSSO as a web application within it (WAR file). Any J2EE compatible web server can be used (e.g. Tomcat, Glassfish).

---

[1] Sun Microsystems has been acquired by Oracle Inc., who, in turn, have decided to discontinue OpenSSO

[2] ForgeRock Website (Downloads). [Online] http://forgerock.com/downloads.html

[3] ForgeRock. *OpenAM WIKI.* [Online] https://wikis.forgerock.org/confluence/display/openam/Home

[4] OpenSSO user's mailing list. [Online] https://opensso.dev.java.net/servlets/SummarizeList?listName=users

[5] Ranganathan, Aravindan. Securing Applications With Identity Services. [Online] [Cited: 5 1, 2010.] http://developers.sun.com/identity/reference/techart/id-svcs.html

2. Managing Authentication and Authorisation with OpenSSO

General Overview

Securing Applications With Identity Services describes comprehensively the REST interface of OpenSSO. The full interface is available within OpenTox, while this documentation only describes a subset of it. In particular, you will also find services for checking validity of tokens ("Token validation"), for finding which user a specific token belongs to ("Attributes"), or for log out.

How does Authentication and Authorisation work with OpenSSO? In a nutshell, the user authenticates against the OpenSSO server at the URI `http://<hostname>:<port>/auth/authenticate` by POSTing "username" and "password" and in turn receives a token. The token may be used to get authorisation from the OpenSSO server to access (i.e. GET, POST, PUT, DELETE) a specific URI. The process of deciding authorisation is governed by policies stored on the server. The authorisation request itself consists of a POST to `http://<hostname>:<port>/auth/authorize`. The request should contain target URI, one of the four access methods, and a valid token. In case of a grant, a boolean value `true` is returned as content. In case of a deny, boolean `false` is returned.

A lot of backends (e.g. LDAP servers) can be attached to OpenSSO to validate subject identity data. Also, very flexible policies can be created using wildcards and overlapping policies. The flow diagram in Figure 1 describes a possible use case involving authentication and authorisation based on OpenSSO.

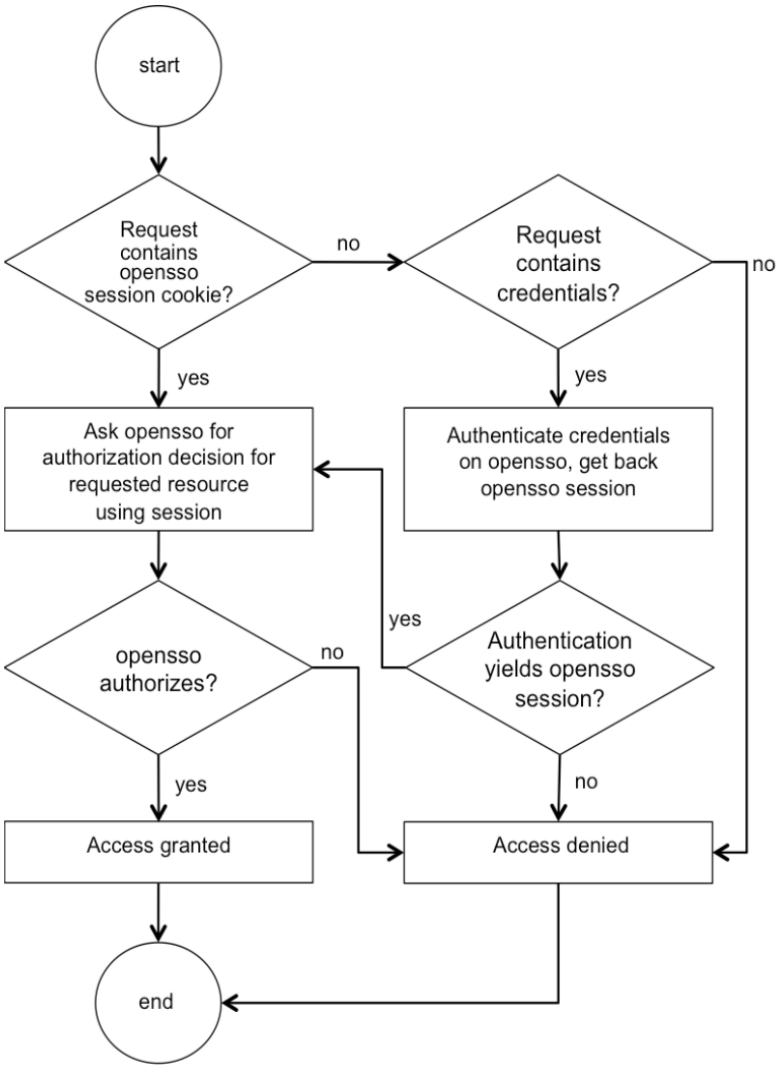

Figure 1: Workflow diagram

Security

All traffic circulating user ids, and passwords should be SSL-secured for privacy.

Since SSL is a standard technology for the web, server certificates should be easy to install and maintain. They are also available free of charge.[6]

All calls to our services, as described in this document, can be made secure using `https`. However, in this documentation we neglect this for better readability.

[6] StartSSL. *Welcome to StartSSL.* [Online] [Cited: 5 1, 2010.] http://www.startssl.com/

OpenTox Authentication and Authorisation Scenario

Authentication: the client authenticates against OpenSSO and obtains a token. The user data is drawn from the LDAP backend that also the OpenTox Plone CMS-based website uses.

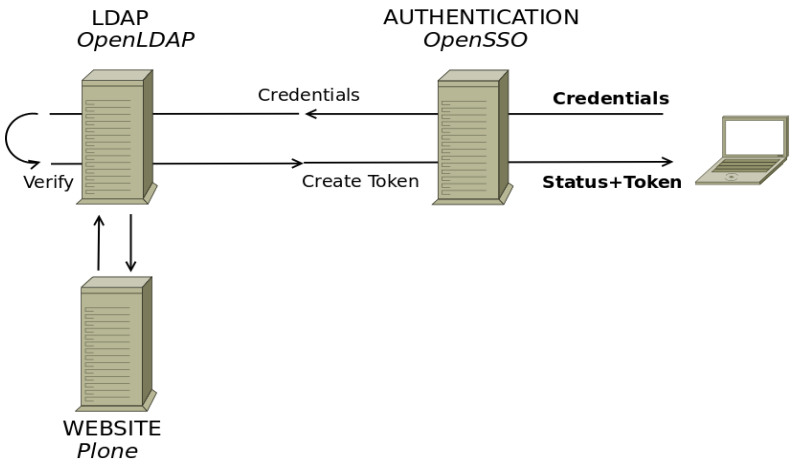

Figure 2: Authentication

- LDAP (here: *OpenLDAP*) is used by the OpenTox Plone CMS as a Data Store. Therefore, OpenTox Plone CMS registered users are instantly available as users in OpenTox web services.
- For unknown credentials, no token is created and an appropriate status code is generated.

Authorisation: The token is used to permit or deny a client a specific action. It encodes a conjunction of user and point of time, and has a certain lifetime[7]. If a token is authorised for the action according to the current server's policy, the web service performs the action.

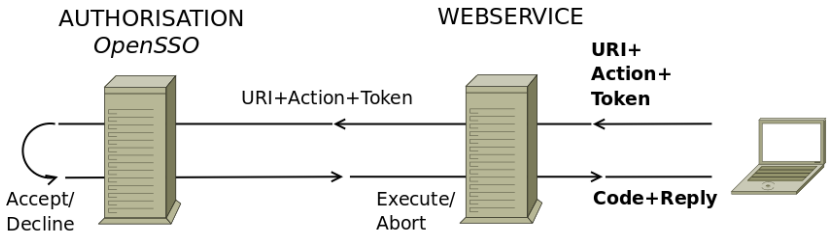

Figure 3: Authorisation

- Tokens encode user identity and time constraints.
- Tokens are valid for a certain time period only (customizable).
- The triplet URI+Action+Token makes up the call to be authorised (not only URI+Token).
- Actions are one of GET, PUT, POST, DELETE.

[7] The lifetime should be set to a high value to allow chained configurations of services (currently 24hrs).

Example Session (Authentication & Authorisation)

After configuring policies for target URI `http://opentox:8080/protected` (see chapter 3), the REST calls below may be used for authentication and authorisation.

Note: OpenSSO accesses the OpenTox Plone CMS-based user data repository for authentication. Thus, the fields `UID` and `SEC` should be replaced with values for a user configured in the OpenTox Plone CMS-based website.

```
# Authentication...
# =================
curl -i -d "username=<UID>" -d "password=<SEC>"
http://opensso.in-silico.ch/auth/authenticate?uri=service=openldap
Reply: HTTP/1.1 200 OK
Content-Type: text/plain
token.id=AQIC5wM2LY4SfcxrnpcZCmbfdsKTyxG9E66uu5FVhefps7I=@AAJTSQACMDE=#

# Authorisation...
# ================
curl -i -d "uri=http://opentox.org:8080/protected" -d "action=GET" -d "subjectid=
AQIC5wM2LY4SfcxrnpcZCmbfdsKTyxG9E66uu5FVhefps7I%3D%40AAJTSQACMDE%3D%23"
http://opensso.in-silico.ch/auth/authorize
Reply: HTTP/1.1 200 OK
Content-Type: text/plain
boolean=true
```

Here, authorisation is granted for the user (boolean=true). `<UID>` and `<SEC>` should be replaced with credentials of an OpenTox website user.

Note: All parts of the query were URL-encoded here (not necessary for POST). For multiple authentications of the same user the `ForceAuth` flag may be used by appending `&uri=ForceAuth=true` to the authentication query string.

REST interface (Authentication and Authorisation)

The OpenTox API needs modification in some places and the introduction of some new components to support Authentication and Authorisation. We have two cases:

(a) Authentication against OpenSSO: should be done by the client application

(b) Authorisation against OpenSSO for resource and action combinations

For (a), we need to transmit user credentials and obtain a token, while part (b) can be decomposed into:

(b1) Client authorisation request to the web service: should be done by the client application

(b2) Authorisation request confirmation from web service to OpenSSO: should be done by the web service.

The following is a proposal for OpenTox API extensions, according to (a), (b1), and (b2): (a)

Desired action	URL	Parameters	Return values (conditions)
Authentication	POST on /auth/authenticate	username[1] password[1] uri[1]	200 + token (Valid) 401 (Invalid)
Token validation	POST on /auth/isTokenValid	tokenid[1]	200 + Boolean
Logout	POST on /auth/logout	subjectid[1]	200 + void

[1] URI-encoded: **YES**
Note: all parameters are **form** parameters.

(b1)

Desired action	URL	Parameters	Return values (conditions)
All	All	as before + tokenid[1]	as before + 401

[1] URI-encoded: **YES**
Note: all parameters are **form** parameters.

(b2)

Desired action	URL	Parameters	Return values (conditions)
Authorisation	POST on /auth/authorize	uri[1] action[1] subjectid[1]	200 + Boolean (Grant) 401 + Boolean (Deny)

[1] URI-encoded: **YES**
Note: all parameters are **form** parameters.

Table 1. Authentication and Authorisation OpenTox API

3. Managing policies

This chapter describes how policies, on the basis of which access to a specific resource is granted (or not granted), are managed. Please review the SUN OpenSSO policy guide[8] to understand access policies in OpenSSO. As currently no support is available for managing policies via REST calls in OpenSSO, we created a specific service for this purpose. Note the following important aspects about policy evaluation:[9]

> *"When multiple policies are applicable to a particular resource, the order in which the policies are evaluated is not deterministic."*

This means that positive results – i.e. allows – add up, as long as no negative results are encountered. In the latter case, evaluation is stopped immediately and access is denied. In the former case, the effective outcome policy is the addition of all the (positive) results. Access is also denied, if no rule matches.[10]

There are wildcard operators for URIs (TARGET_URI). The following URI matches all possible URIs: `*://*:*/*`. The wildcard operators (`*`) mask protocol (`http` or `https` only), hostname, port, and resources, respectively. The resource may be masked more specifically, e.g. `http://*:*/path/to/*` matches everything only in or below `/path/to` in the web server's root directory, assuming `http` protocol.

The wildcard operator stretches by default across all levels, e.g. http://opentox.org/* will match http://opentox.org/1 as well as http://opentox.org/1/2. However, there is a one-level-wildcard operator: `-*-` that will only match the former. To prevent excessive wildcard use, initially only the one-level-wildcard operator was allowed. . Currently, the following wildcards are allowed:

Certain wildcards are now allowed in resource URIs in policies. These are:

```
/dataset/-*-
/feature/-*-
/compound/-*-
/conformer/-*-
/metadata/-*-
/model/-*-
/algorithm/CDKPhysChem/-*-"
/algorithm/JOELIB2/-*-"
```

Note: in case of an access denial, a matching rule's decision cannot be overridden by a more specific rule.

Upon a call of the authorisation REST interface, OpenSSO

1. finds all policies applicable to the combination of user (encoded in the token) and target URI.
2. creates an effective policy from the policies found in 1.

Generally, the service follows a conservative principle: the effective current policy is constrained by the most restrictive policy that applies to the current URL. If no policy applies, access is denied.

---

[8] Chapter 4 Managing Policies. *Sun OpenSSO Enterprise 8.0 Administration Guide.* [Online] [Cited: 5 1, 2010.] http://docs.sun.com/app/docs/doc/820-3885/gipxb?a=view

[9] Applying Policy Logic. *Sun OpenSSO Enterprise 8.0 Administration Guide.* [Online] http://docs.sun.com/app/docs/doc/820-3885/gjeby?a=view

[10] Need help defining policies. *OpenSSO: Mail reader.* [Online] https://opensso.dev.java.net/servlets/ReadMsg?listName=users&msgNo=3354.

Example Session (Policies)

```
# Listing all my policies...
# ========================
curl -i -X GET http://opensso.in-silico.ch/pol -H "subjectid: AQIC5wM2LY4SfcyH9ELyynby356aOvAkimDYeEz2wzWTSX4=@AAJTSQACMDE=#"
HTTP/1.1 200 OK
Content-Type: text/plain

# Creating a policy...
# ====================
curl -i -H "Content-Type: application/xml" -T /home/am/aa/Pol-REST/sample-pol.xml -X POST http://opensso.in-silico.ch/pol -H "subjectid: AQIC5wM2LY4SfcyH9ELyynby356aOvAkimDYeEz2wzWTSX4=@AAJTSQACMDE=#"
HTTP/1.1 200 OK
Content-Type: text/plain
Policies were created under realm, /.

# Listing all my policies (again)...
# ==============================
curl -i -X GET http://opensso.in-silico.ch/pol -H "subjectid: AQIC5wM2LY4SfcyH9ELyynby356aOvAkimDYeEz2wzWTSX4=@AAJTSQACMDE=#"
HTTP/1.1 200 OK
Content-Type: text/plain
s2_policy
```

```
# Listing my s2 policy...
# ============================
curl -i -X GET -H "subjectid: AQIC5wM2LY4SfcyH9ELyynby356aOvAkimDYeEz2wzWTSX4=@AAJTSQACMDE=#"
http://opensso.in-silico.ch/pol/s2_policy
HTTP/1.1 200 OK
Server: nginx/0.6.32
Content-Type: text/xml

<?xml version="1.0" encoding="UTF-8"?>
<Policies>
    <Policy name="s2_policy" createdby="id=amadmin,ou=user,dc=opensso,dc=java,dc=net"
lastmodifiedby="id=amadmin,ou=user,dc=opensso,dc=java,dc=net" creationdate="1275290803394"
lastmodifieddate="1275290803394" referralPolicy="false" active="true">
        <Rule name="s2 rule 2">
            <ServiceName name="iPlanetAMWebAgentService"/>
            <ResourceName name="http://opentox.org/s2"/>
            <AttributeValuePair>
                <Attribute name="POST"/>
                <Value>allow</Value>
            </AttributeValuePair>
            <AttributeValuePair>
                <Attribute name="GET"/>
                <Value>allow</Value>
            </AttributeValuePair>
        </Rule>
        <Subjects name="s2 subject 2" description="">
            <Subject name="amaunz" type="LDAPUsers" includeType="inclusive">
                <AttributeValuePair>
                    <Attribute name="Values"/>
                    <Value>uid=amaunz,ou=people,dc=opentox,dc=org</Value>
                </AttributeValuePair>
            </Subject>
        </Subjects>
    </Policy>
</Policies>

# Deleting my policy...
# =================
curl -i -X DELETE http://opensso.in-silico.ch/pol/s2_policy -H "subjectid:
AQIC5wM2LY4SfcyH9ELyynby356aOvAkimDYeEz2wzWTSX4=@AAJTSQACMDE=#"
HTTP/1.1 200 OK
Content-Type: text/plain
Policies were deleted under realm, /.
```

REST interface (Policies)

To create a policy, issue a POST to http://<pol-server>/Pol/opensso-pol with the XML file to transfer and header entry "Content-Type: application/xml". The XML file should match the following schema:

```
<!DOCTYPE Policies PUBLIC "-//Sun Java System Access Manager7.1 2006Q3
  Admin CLI DTD//EN" "jar://com/sun/identity/policy/policyAdmin.dtd">

<Policies>
<Policy name="POLICY_NAME" referralPolicy="false" active="true">
   <Rule name="RULE_NAME">
      <ServiceName name="iPlanetAMWebAgentService" />
      <ResourceName name="TARGET_URI"/>
      <AttributeValuePair>
         <Attribute name="ACTION_NAME" />
         <Value>ACTION_VAL</Value>
      </AttributeValuePair>
   </Rule>
   <Subjects name="SUBJECT_GROUP" description="">
        <Subject name="SUBJECT_ID" type="LDAP_TYPE" includeType="inclusive">
            <AttributeValuePair>
                <Attribute name="Values"/>
                <Value>LDAP_DN</Value>
            </AttributeValuePair>
        </Subject>
   </Subjects>
</Policy>
</Policies>
```

Figure 4: XML template for policies

All parts are mandatory. **Bold** parts may occur more than one time, allowing for compound objects such as complex datasets, which may consist of compounds and features stored on different servers, under different URIs. It allows also for multiple actions and subjects. The following table explains the fields that must be set:

POLICY_NAME	Arbitrary string, e.g. "my_policy". Must not contain spaces!
RULE_NAME	Arbitrary string, e.g. "my rule"
TARGET_URI	URI to protect, e.g. "http://opentox.mybox.org/res"
ACTION_NAME	One of "GET", "PUT", "POST", "DELETE"
ACTION_VAL	One of "allow", "deny"
SUBJECT_GROUP	Arbitrary string, e.g. "my people"
SUBJECT_ID	Arbitrary string, e.g. "John Doe"
LDAP_TYPE	One of "LDAPUsers", "LDAPGroups"
LDAP_DN	Distinguished name, e.g. "uid=jdoe, ou=people, dc=opentox,dc=org"[11]

[11] Individuals always use: uid=<uid>, ou=people, dc=opentox, dc=org,
Groups always use: cn=<gid>, ou=groups, dc=opentox, dc=org.
Note: <uid>/<gid> should be replaced with OpenTox Plone user/group ids, respectively.

The following table documents the OpenTox Policy REST interface:

Desired action	URL	Parameters	Return values (conditions)
1. Create a policy	POST XML file to http://<pol-server>/Pol/opensso-pol	subjectid[1]	200 (OK) 400 (XML contains errors) 500 (Other Errors)
2. List policies	GET on http://<pol-server>/Pol/opensso-pol	subjectid[1]	200 (OK) 500 (Other Errors)
3. List policy pol	GET on http://<pol-server>/Pol/opensso-pol/pol	subjected[1]	200 (OK) 401 (Unauthorised) 500 (Other Errors)
4. Delete policy pol	DELETE on http://<pol-server>/Pol/opensso-pol/pol	subjectid[1]	200 (OK) 400 (Policy non-existent) 401 (Unauthorised) 500 (General Error)

[1] Header-Parameter

- Explanation of parameters: subjectid attribute should have a valid token as value.
- *Note:* PUT is currently not supported. All operations apart from 3. return "Content-Type: text/plain" in their header, since they return the original output of the ssoadm command, which is used internally to fulfill the requests. Command 3. returns "Content-Type: text/xml".

Table 2. Policy API.

Obtaining User Attributes

OpenSSO provides user attributes associated with a specific token, such as the distinguished name (dn) needed in policies, as well as other information, through a dedicated service:

```
# User attributes...
# =====================
curl -i -d "attributes_names=uid" -d
"subjectid=AQIC5wM2LYaSfcyjBrAxpCWyKYD7SYNOyh%2Ba2TCDf2Fz97E%3D%40AAJTSQACMDE%3D%23"
http://opensso.in-silico.ch/auth/attributes

HTTP/1.1 200 OK
Content-Type: text/plain
userdetails.token.id=AQIC5wM2LYaSfcyjBrAxpCWyKYD7SYNOyh+a2TCDf2Fz97E=@AAJTSQACMDE=#
userdetails.attribute.name=uid
userdetails.attribute.value=amaunz
userdetails.attribute.name=mail
userdetails.attribute.value=andreas@maunz.de
userdetails.attribute.name=sn
userdetails.attribute.value=Maunz Andreas
userdetails.attribute.name=dn
userdetails.attribute.value=uid=amaunz,ou=people,dc=opentox,dc=org
```

Obtaining Group Membership Data

Since policies allow for authorisation based on groups, services for finding groups, and the groups a specific user is in, should be available. OpenSSO provides such facilities with the "search" and "read" services.[12]

```
# Listing all groups...
# =====================
curl -i -d "attributes_names=objecttype" -d "attributes_values_objecttype=group" -d
"admin=AQIC5wM2LY4SfcwSwYFi4MY3Z%2Ff52VpgCovcl%2FItde2OC0I%3D%40AAJTSQACMDE%3D%23" http://opensso.in-
silico.ch/auth/search
HTTP/1.1 200 OK
Content-Type: text/plain
string=development
string=partner
```

[12] Chapter 10 Using the REST Identity Interfaces. *Sun OpenSSO Enterprise 8.0 Developer's Guide.* [Online] [Cited: 1 5 2010.] http://docs.sun.com/app/docs/doc/820-3748/gjdsc?a=view

```
# Membership of a specific user...
# =====================
curl -i -d "name=amaunz" -d attributes_names="group" -d
"admin=AQIC5wM2LY4SfcwSwYFi4MY3Z%2Ff52VpgCovcl%2FItde2OC0I%3D%40AAJTSQACMDE%3D%23" http://opensso.in-silico.ch/auth/read
HTTP/1.1 200 OK
Content-Type: text/plain
identitydetails.name=amaunz
identitydetails.group=development
identitydetails.group=partner
```

There exist two groups, “development” and “partner”. User “amaunz” is member of both. Note that the parameter for the token is now “admin”. Any registered user is entitled to search for groups and memberships. It is also possible to search for all existing usernames by setting `attributes_values_objecttype=user` in the first query above.

4. Managing Privileges

A privilege is a “policy for policies”, i.e. a privilege regulates access to the URI of the policy. Our system implements such a service (see section 3) by allowing only the owner to change the policy.

Creating and Deleting Privileges

For every newly created policy, a privilege is simultaneously created. On write access (currently only DELETE) to the policy, authorisation is requested from the privilege unless the policy has not been existent before (creation of a policy). In the latter case, the user creating the policy is associated with it as the owner of the policy.

There are two basic use cases for policy *p* at URI *u*, protected by privilege *q*:

1) Create *p* at URI *u*, where there is no existing policy for *u*. This requires no authorisation and creates *q*. As a side effect, the system remembers the user and associates him/her with *q* as the owner of *q*.
2) Delete *p*. This requires write access to *u*, granted or denied by *q*. Only the owner has write access.

The described procedure gives the owner full control over the policies created by him. In this way, he/she can control who has the right to access a particular algorithm, model, dataset, etc.

Deleting a policy releases the protected URIs.

Multiple Policies

The owner can register multiple policies referencing the same resource “on top" of the first policy. Upon an authorisation request, OpenSSO calculates then the effective policy from all policies, as described in section 3.

5. Specific issues and directions for future development

As already mentioned in the introduction, REST security is still work-in-progress, and many problems have yet to find their solutions. Further, the OpenTox infrastructure itself presents unique security challenges. Below follows a short discussion on each of the specific issues that have been identified.

Users and groups

Users of the OpenTox infrastructure might come from rather different organizational domains. Some of these domains might be happy with a common, centralised user database, while others might insist on keeping their own databases and even doing their own authentication.

Because a user database already exists for the OpenTox website, it was decided to use it as a common, centrally provided database for the whole OpenTox infrastructure as well. Thus, anyone who wishes to access the OpenTox infrastructure will have to first register through the OpenTox website. However, two exceptions exist.

First, as already mentioned, some organizations might prefer to keep their users off such a central database. For this reason, it should be possible to build a *federated* system, where several AA systems talk to each other, exchanging authentication and authorisation data. All usernames then also include a *domain* or *realm* which identifies the organization they belong to or which handles their registration data. For instance users Bob and Mary might be registered through OpenTox's site (they need not be project partners, but merely use the website to handle their registration data – which, however, would still have to be confirmed), and there might be another user named Bob from a different organization, which uses its own AA service. The three users will then be, respectively: bob@opentox.org, mary@opentox.org and bob@somethingelse.com. This mechanism thus provides means to preserve uniqueness of the usernames when multiple organizations are involved – not much unlike the e-mail system where there might be coinciding users (**bob**@...), but still being unique when the complete address (e.g. bob@gmail.com vs. bob@yahoo.com) is taken into account.

The second exception concerns the *guest* or *anonymous* access. In fact, most of the OpenTox data might not have any specific restrictions imposed on it, being entirely public. While it is still possible to require everyone desiring access to such data to register on opentox.org, this could have detrimental effect on the number of potential users, for different reasons: some people might consider this a breach (even if minor) of their privacy, others might not be willing to spend the time filling in the registration form, etc. For this reason, it was decided to provide such users with an anonymous or guest account, which effectively bypasses the authentication process, similar to the once popular anonymous access to FTP sites. Of course, besides being able to only see data marked as public, such users will also likely be imposed with more stringent restrictions in terms of computational resources that they could use at any given time.

The user groups present a specific challenge, because they need an efficient system to manage them. It should be powerful enough, yet easy to use and understand, because while the groups provide a convenient way to manage access restrictions, quite often they also present a serious security risk. A good, if somewhat distant, example are the ill-fated Facebook privacy policies, with which the social network's users have struggled for quite a long time with variable success. At least part of the problem has been the somewhat vague definition of different "friend groups": friends, friends of friends, networks, custom lists, etc.

Access control policies

There are several issues related to how the access control policies work:

- While for the newly created resources the access control policy is defined by the creator, just who can create those new resources is an open question. Obviously, the answer is also rather dependent on the type of resource in question. The anonymous/guest user presents an even further challenge.
- When a restricted resource is used to produce a new resource, the problem with access control *inheritance* might arise: should the newly produced resource get automatically the same restrictions as the originating one, or should the user instead be free to choose different access control or even make the newly created resource public? It is to be expected that different approaches might make most sense in different scenarios.
- While OpenSSO's policies have strict syntax, it is useful to have a more efficient way of communicating them between the clients and services and between the services themselves.
- Because the creation of a new resource and the registration of its access control policies is not an atomic process, special attention needs to be paid to the scenarios where only one of them fails.

6. Extended use case examples

Authentication

1. The **client** establishes encrypted SSL/TLS connection to the **OpenSSO server**.
2. The **client** sends username and password through the encrypted channel to the **OpenSSO server**.
3. The **OpenSSO server** verifies the presented user credentials against **opentox.org's user database**.
4. Depending on the result of this verification:
    a. On success, the **OpenSSO server** returns a cryptographic *token* to the **client**.
    b. On failure, the **OpenSSO server** returns appropriate error message.

Uploading dataset

1. The **client** establishes encrypted SSL/TLS connection to the **dataset service**.
2. The **client** fetches the user and group lists from the **OpenSSO server**.
3. The **client** presents the **user** with an interface to select the required permissions for the dataset to be uploaded: different users and/or groups could be selected and for each one any of the four HTTP methods GET, POST, PUT and DELETE could be either allowed or disallowed.
4. The **client** POSTs the dataset to the **dataset service**, providing along also:
    a. The token acquired from the authentication phase.
    b. The policies to be created based on the user preferences from step 3.
5. The **dataset service** sends the provided token for validation to the **OpenSSO server**.
6. If the token is not valid, the **dataset service** returns an appropriate error message. Otherwise, it checks with the **OpenSSO server** whether the user is allowed to create new data.
7. If the user is allowed to create new data, the **dataset service** registers the uploaded dataset, but does not publish it yet.
8. The **dataset service** requests the **OpenSSO policy service** to create the requested policy by the user in step 3 for the resources registered in the previous step.
9. If the creation of access control policy fails, the **dataset service** returns an appropriate error message to the **client**. Otherwise, if the policy is successfully created, the dataset is published.
10. The **dataset service** returns the newly created dataset ID to the **client**.

Creating a Model

1. The **client** establishes encrypted SSL/TLS connection to the **model service**.
2. The **client** POSTs a dataset URI to the **model service** , providing also:
    a. a feature URI describing an endpoint
    b. model parameters
    c. a valid *token*.
3. The **model service** POSTs a request for the dataset to the **dataset service** providing the *token*.
4. The **dataset service** requests permission from the **OpenSSO server** to access the dataset URI via GET using the *token*.
    a. On success, the **dataset service** returns the dataset to the **model service**. The **model service** learns a model for the endpoint from the dataset, based on the model parameters and returns the model URI to the **client.**
    b. On failure, the **dataset service** returns an appropriate error code to the **model service**. The **model service** returns an appropriate error code to the **client**.

Logout

1. The **client** establishes encrypted SSL/TLS connection to the **OpenSSO server**.
2. The **client** invalidates the *token* using the **OpenSSO server**.

The upload of a dataset illustrates the creation of a policy, whereas during model creation, this policy is used to allow the model service to access the dataset. An interesting aspect in the model building step is to employ the user's token in a chained configuration, i.e. not only between client software and service, but also in service-to-service communication.

A crucial step is to invalidate the token after it has been used in a workflow. This principle should guide all use cases to prevent fraudulent use of tokens. Also, SSL/TLS encrypted connections should be used.

7. Conclusions

The proposed approach makes general protection available for resources accessible by URIs. It provides a single sign-on technique involving all previously registered users and groups from the OpenTox website. OpenSSO has a long history of successful deployments in mission-critical scenarios, however, its application in REST-based environments is rather new.

Because of the novelty of the area and the general lack of REST-centered security solutions, a lot of issues that need further research have been identified. In fact, while the presented solution should be sufficient for the basic integration of confidential data, it might not prove flexible enough once the OpenTox infrastructure gets more "real-life" usage.

Therefore, along with the gradual improvement of the current approach, it is deemed rather important to simultaneously investigate entirely different concepts – mostly those that target specifically the RESTful services, such as the solutions based on FOAF+SSL and symmetric or asymmetric (PGP-like) encryption of data.