GÉANT

REST architecture for perfSONAR Authentication model

Nina Jeliazkova (BREN)
Candido Montes (RedIris)

connect • communicate • collaborate

RESTful perfSONAR

GÉANT

Investigation of REST architecture for web services and its applicability to perfSONAR web services (Jan 2010)

- REST principles
- Mapping of existing perfSONAR schemas
- Authentication model
- Performance measurement

REpresentational State Transfer (REST)

- Architectural style for distributed information systems on the Web
- Simple interfaces, data transfer via **hypertext transfer protocol (HTTP)**, stateless client/server protocol
  - GET, POST, PUT, DELETE
- Each **resource** is **addressed** by its own **web address**
- **Lightweight** approach to **web services**
- **Simplifies/enables** development of **distributed and local** systems
- **SOAP vs REST**
  - SOAP is a protocol , REST is an architectural style, not a protocol.
  - SOAP: established WS-SOAP standards,
  - **REST: Currently NO standards for RESTful applications, but merely design guides.**

REST design guide

GÉANT

Resource oriented

- **Every object (resource) is named and addressable (e.g. HTTP URL)**

Example: http://perfsonarservice.net/MeasurementArchive/interface/interface_identifier

Transport protocol

- HTTP is the most popular choice of transport protocol, but there are examples of systems using other protocols as well.

Operations

- Resources (nouns) support limited number of operations (verbs). HTTP operations are the common choice, when the transport protocol is HTTP.

RESTful operations

- GET (retrieve the object under specified URL)
- PUT (update the content of an object at the specified URL)
- POST (create a new object and return the URL of the newly created resource)
- DELETE (delete the object)

All operations, except POST should be safe (no side effects) and idempotent (same effect if executed multiple times).

Non-RESTful operations

- Everything else , e.g. POST XML message to http://perfsonarservice.net/MAservice

Resource representation (Media types)

Hypermedia as the Engine of Application State (hyperlinks!)

perfSONAR REST resources

GÉANT

1)Service: URL: http://servicehostname:port/servicename/
Representation: text/xml
Operations:
- GET - return metadata of the service itself
- PUT - input: XML with service metadata ; output: adds service metadata
- POST - input: XML with service metadata ; output: replaces service metadata
- DELETE - remove service metadata

2)Topology elements URL:
http://servicehostname:port/servicename/topologyelements?paramName=value¶mName=value1¶mName1=value2
Representation: text/xml
Operations:
- GET Returns NMWG representation of topology elements (response of the NMWG MetadataKeyRequest)
- PUT
- POST Create new topology element(s) by sending representation in NMWGT XML
- DELETE Delete all topology elements

Parameters : parameters and values as in NMWG/NMWGT

perfSONAR REST resources

GÉANT

3) Topology elements of specific type (Metadata)
SOAP request: MetadataKeyRequest

URL (REST request):

http://servicehostname:port/servicename/topologyelements/{type-of-topologyelement}/

metadata?parameterName=value

Representation: NMWG XML , content type text/xml or to be defined

Operations:

- GET, PUT, POST, DELETE

4) Topology elements of specific type (Data)
SOAP request: SetupDataRequest

URL (REST request):

http://servicehostname:port/servicename/topologyelements/{type-of-topologyelement}/metadata?parameterName=value

Representation: NMWG XML , content type text/xml or to be defined

Operations:

- GET, PUT, POST, DELETE

perfSONAR REST resources

GÉANT

5) A single topology element

URL:

http://servicehostname:port/servicename/topologyelements/{type-of-topology-element}/{idof-the-topology-element}

Representation: text/xml

Operations:

- GET Retrieve metadata of the topology element
- PUT Add metadata of the topology element input: NMWGT XML
- POST Replace metadata of the topology element input: NMWGT XML
- DELETE Delete topology element

Examples:

- GET http://servicehostname:port/servicename/topologyelements/interface/1


```xml
<nmwgt:interface>
<nmwgt:hostname>myhostname</nmwgt:hostname>
<nmwgt:ifName>myifname</nmwgt:ifName>
<nmwgt:ifDescription>My Interface</nmwgt:ifDescription>
<nmwgt:ifAddress>10.0.0.1</nmwgt:ifAddress>
<nmwgt:ifIndex>eth0</nmwgt:ifIndex>
<nmwgt:direction>in</nmwgt:direction>
<nmwgt:capacity>100000000</nmwgt:capacity>
</nmwgt:interface>
```

Performance comparison

Data retrieval performance

Response time, ms

9000.00
8000.00
7000.00
6000.00
5000.00
4000.00
3000.00
2000.00
1000.00
0.00

8300.00
4200.00
2900.00 2900.00
2100.00
1600.00
1400.00
719.60
671.40
455.60
340.10
312.40
190.20
150.00
133.30
81.20
53.00
32.70
22.10
21.20

15min 90min 8h 24h 3 days 1 week 1 month 3 months 6months 1 year

Queried time range (1 interface,2 directions, 5 min resolution)

REST-StAX, ms
REST-NMWG
SOAP-NMWG

SOAP Authentication model

GÉANT

The authentication of the official perfSONAR is based on

- Web Services Security (SOAP 1.1)
- X.509 digital certificate profile
- SAML profile in order in to include the security tokens defined in its architecture.

REST Authentication model

GÉANT

There is no defined standard for protecting RESTful web services.

There is a draft, HTTP Authentication: Token Access Authentication (http://tools.ietf.org/html/draft-hammer-http-token-auth-00)which can be used in to include the security tokens in every request.

- Originally created to propose a better solution for OAuth;
- Quite generic and it can be used for other kind of tokens.

HTTP Request without token

```
GET /resource HTTP / 1.1
Host: example.com
```

returns the following authentication challenge:

```
HTTP / 1.1 401 Unauthorized
WWW-Authenticate: Token
  realm="http://example.com/",
 coverage="base base+body-sha-
  256",
timestamp="137131190",
class="x509v1 x509v3 saml20-
  base64"
```

realm:

coverage: the list of authentication coverage names supported by the server.

timestamp: this is used by the server to publish its current time, enabling clients to synchronize their close with the server.

class: the list of token types supported by the server.

method: the list of authentication method names supported by the server, provided as a space-delimited list.

HTTP Request with token

```
GET /resource HTTP / 1.1
  Host: example.com
Authorization: Token
token="h480djs93hd8...yZT4=",
coverage="base",
nonce="dj83hs9s",
timestamp="137134190",
auth="djosJKDKJSD8743243/jdk3
  3klY=",
class="x509v3",
method="rsassa-pkcs1-v1.5-
  sha-256"
```

token: the value used to represent the security token.

coverage: sets the name of the authentication coverage method used by the client to make the request. (See section 5.2 of the draft).

nonce: contains a random string as the draft specifies.

timestamp: contains the timestamp of the user's client.

auth: the output of the authentication method function after applying it to the selected coverage as described in draft Section 7).

class: sets the name of the token type used by the client to make the request.

method: the name of the authentication method used by the client to make the request. (See Section 7 of the draft).

Profile based on X.509 digital certificates

This profiles define how a token based on a X.509 digital certificate should be sent. The requirements are:

Token types: the following tokens are defined for this profile:

Class name	Token type
x509v3	An X.509 v3 certificate capable of signature-verification at a minimum.
x509v1	An X.509 v1 certificate capable of signature-verification at a minimum.

Token value: the token value is represented using the base 64 codification of the DER value of the certificate.

Authentication method: Use of *rsassa-pkcs1-v1.5-sha-256* for calculating the *auth* parameter using the private key of the certificate.

Profile based on X.509 digital certificates

```
GET /resource HTTP / 1.1
Host: example.com
Authorization: Token
token="h480djs93hd8...yZT4=",
coverage="base",
nonce="dj83hs9s",
timestamp="137134190",
auth="djosJKDKJSD8743243/jdk33klY=",
class="x509v3",
method="rsassa-pkcs1-v1.5-sha-256"
```

token: the DER value of the X.509 digital certificate in base64.

coverage: sets the name of the authentication coverage method used by the client to make the request.

nonce: contains a random string as the draft specifies.

timestamp: contains the timestamp of the user's client.

auth: the output of the *rsassa-pkcs1-v1.5-sha-256* method function.

class: the token type as described in the previous table.

method: the authentication method used by the client, which MUST be "rsassa-pkcs1-v1.5-sha-256".

Profile based on SAML assertions

This profiles define how a token based on a SAML assertion should be sent.

There are different confirmation methods:

- Bearer
- Holder-of-Key
- Sender-vouches

SAML
Bearer Confirmation Method

GÉANT

The requirements are:

Token types: it MUST be the value "saml20-base64" or "saml11-base64".

Token value: the token value is represented using the base 64 codification of the SAML assertion.

Authentication method: any authentication method can be used but in case the 'none' method is not used the selected key and its transmission is out of scope of this document.

```
GET /resource HTTP / 1.1 Host:
  example.com Authorization:
  Token
  token="h480djs93hd8...yZT4=",
  coverage="none",
  class="saml20-base64",
  method="none"
```

token: the SAML assertion in base64.

coverage: sets the name of the authentication coverage method used by the client to make the request.

class: the token type for the SAML Assertion base on the second version of that technology.

method: the authentication method used by the client.

SAML
Bearer Confirmation Method

GÉANT

The requirements are:

Token types: it MUST be the value "saml20-base64" or "saml11-base64".

Token value: the token value is represented using the base 64 codification of the SAML assertion.

Authentication method: any authentication method can be used but in case the 'none' method is not used the selected key and its transmission is out of scope of this document.

```
GET /resource HTTP / 1.1
Host: example.com
Authorization: Token
         token="h480djs93hd8...yZT4=",
        coverage="none",
        class="saml20-base64",
        method="none"
```

token: the SAML assertion in base64.

coverage: sets the name of the authentication coverage method used by the client to make the request.

class: the token type for the SAML Assertion base on the second version of that technology.

method: the authentication method used by the client.

SAML
Holder-of-Key Confirmation Method

GÉANT

The requirements are:

Token types: it MUST be the value "saml20-base64" or "saml11-base64".

Token value: the token value is represented using the base 64 codification of the SAML assertion.

Authentication method: Use of *rsassa-pkcs1-v1.5-sha-256* for calculating the *auth* parameter using the private key which has generated the public key included in the <SubjectConfirmation> element.

```
GET /resource HTTP / 1.1
Host: example.com
Authorization: Token
        token="h480djs93hd8...yZT4=",
        coverage="base", nonce="dj83hs9s",
        timestamp="137134190",
        auth="djosJKDKJSD8743243/jdk33klY=",
        class="saml20-base64",
        method="rsassa-pkcs1-v1.5-sha-256"
```

token: the SAML assertion in base64.

coverage: sets the name of the authentication coverage method used by the client to make the request.

nonce: contains a random string as the draft specifies.

timestamp: contains the timestamp of the user's client.

auth: the output of the *rsassa-pkcs1-v1.5-sha-256* method function.

class: the token type for the SAML Assertion base on the second version of that technology.

method: the authentication method used by the client, which MUST be "rsassa-pkcs1-v1.5-sha-256".

SAML
Sender-Vouches Confirmation Method

The requirements are:

Token types: it MUST be the value "saml20-base64" or "saml11-base64".

Token value: the token value is represented using the base 64 codification of the SAML assertion.

Authentication method: Use of *rsassa-pkcs1-v1.5-sha-256* for calculating the *auth* parameter using the private key which has signed the SAML assertion.

```
GET /resource HTTP / 1.1

Host: example.com

Authorization: Token

token="h480djs93hd8...yZT4=",
coverage="base", nonce="dj83hs9s",
timestamp="137134190",
auth="djosJKDKJSD8743243/jdk33klY=",
class="saml20-base64", method="rsassa-
pkcs1-v1.5-sha-256"
```

token: the SAML assertion in base64.

coverage: sets the name of the authentication coverage method used by the client to make the request.

nonce: contains a random string as the draft specifies.

timestamp: contains the timestamp of the user's client.

auth: the output of the *rsassa-pkcs1-v1.5-sha-256* method function.

class: the token type for the SAML Assertion base on the second version of that technology.

method: the authentication method used by the client, which MUST be "rsassa-pkcs1-v1.5-sha-256".

AA for REST services
RFC draft preparation

- Diego Lopez
- Elena Lozano
- Candido Rodriguez
- Klaas Wierenga
- Nina Jeliazkova

GÉANT

Thank you!