B2ACCESS Service Integration

About

Document that describes how a service provider can integrate their service with B2ACCESS.

Service: B2ACCESS

Modified: 09 January 2018

Synopsis

This document describes how an EUDAT service provider, hosting an application which requires authentication, can use B2ACCESS to authenticate and authorise users. The provider has different options, depending on the type of service (web/non-web) and the type of authorisation it needs (fine grained, coarse grained, none).

It is assumed that integrators making use of this documentation have some understanding of the technologies they are integrating, including the commonly used abbreviations and terminologies. This document is **not** intended as a tutorial in the services it describes, although it may include material of an introductory nature.

Occasionally this document makes use of the terminology of RFC2119 (MUST, MUST NOT, SHOULD, etc.) and familiarity with these will also be helpful.

Introduction

In identity federations, there are *identity providers* (IdPs) and *service providers* (SPs), depending on whether they *produce* identity tokens or *consume* them. For the EUDAT CDI, IdPs are usually external to EUDAT (users can bring their own identity) and the SPs form the EUDAT CDI. (Note that, technically, a SP external to EUDAT could make use of B2ACCESS to authenticate users, but in doing so B2ACCESS would send attributes to the external SP, so this situation may raise issues with both data protection and the eduGAIN Code of Conduct.)

B2ACCESS is the EUDAT federated cross-infrastructure authorisation and authentication framework for user identification and access control enforcement. All services provided by EUDAT need to support B2ACCESS.

B2ACCESS uses the UNITY IDM technology. Currently three technologies to integrate a SP with B2ACCESS are supported: SAML, OAuth2 and X.509. Integration is only possible if the application supports one of these three technologies. We discuss these briefly here and provide more details below:

SAML

SAML is the Security Assertion Markup Language, an OASIS standard. The profile used by B2ACCESS is SAML Web SSO. It is the foundation of Shibboleth.

- Advantages:
    - More lightweight than the certificate login path
    - In its B2ACCESS implementation, it carries the SAML attribute assertion inside it (as an extension) for use for authorisation
    - It is an open standard (from OASIS)
- Disadvantages:

EUDAT receives funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 654065.

    - No delegation allowed (see OAuth2 below)
    - Generally only Web Single Sign-On (Web SSO) is supported, not Enhanced Client or Proxy (ECP) - this means that the user will need a browser to authenticate.

OAuth2

OAuth2 is a protocol originally designed (by a committee!) to delegate rights to services on the web: the classic example is a user who authorises a printer to fetch images from an image repository in order to make a printout of the images, but without the user handing over to the printer their repository credentials. The protocol has been extended to provide authentication - this, in turn, has turned into OpenID Connect (which is not covered in this documentation). OAuth2 is an IETF standard defined in RFC6749.

- Advantages:
    - Relatively simple, web-based access - see documentation below
    - Token can be used several times until it expires
    - Standard (IETF, RFC6749)
- Disadvantages:
    - Coarse-grained authorisation; however, user attributes can be queried from the server.
    - Slightly complicated flows
    - Not all the standard is implemented in B2ACCESS (see the OAuth section below for details)

X.509

The typical workflow is that the user authenticates to a portal and the portal obtains a certificate (using OAuth to authorise the issuance of the certificate) with which it can run services on behalf of the user. For example, the portal can then do transfers using GridFTP.

In the B2ACCESS workflows, X.509 certificates are handled by services which act on behalf of the users (just like the OAuth2 and SAML technologies) - however, unlike the others, X.509 certificates *can* be downloaded and managed outside of the browser in order to support command-line access or other non-browser clients.

- Advantages:
    - Works with an enormous range of services; exceptionally well established and highly interoperable technology.
    - In particular, works also for non-web access, either by having users download a credential which they then use themselves, or by doing the non-web access via a portal.
    - Enables fine grained authorisation since it carries the full SAML assertion
    - Can be used as much as you like until it expires
    - Allows delegation (in the sense of GSI (RFC 3820))
    - Standard (ITU/ISO/IEC)
- Disadvantages:
    - Typically needs to be loaded into the service at startup; this is no problem for commands running now and then but long-running services need to handle expiring certificates gracefully.
    - Expires at some (configurable) time. Thus there is no need for revocation but they have to be renewed and reloaded into whichever client is making use of them. Currently (as of B2ACESS 1.0) users must renew them themselves; there is no automatic renewal.
    - End users find them hard to manage directly: the default (OpenSSL) commands to handle them are somewhat esoteric, and browsers are fairly idiosyncratic in their handling of certificates. Moreover, as the certificates have short lifetimes, users need to renew them frequently.
    - Some user communities are avert to using certificates, as certificates have a reputation for being dificult to handle, and managing certificates with browsers and OpenSSL command-line commands is

EUDAT receives funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 654065.

indeed non-trivial
- No user-friendly interface for downloading certificate in B2ACCESS 1.0. This is only a problem in the case of advanced users wanting to use B2ACCESS in command-line mode.

Service Integration

Any service relying on B2ACCESS may choose any of the methods above, according to its security requirements. However, services must be aware that they are constrained by EUDAT data management policies and that attributes must be processed accordingly - see the General Service requirements "checklist" section below.

The different technologies to integrate services and their interaction with the B2ACCESS system are shown in Figure 1 below. On the left hand side it shows external identity providers - a few are greyed out since they are not currently supported in the production service - connecting to the *B2ACCESS AAI functions* which act as a credential translation service (or proxy). Regardless of what technology is used for the "external" identity and which attributes were presented, the authentication is *harmonised* into a single B2ACCESS credential which is meaningful to all EUDAT services that have integrated B2ACCESS. The user thus holds the same identity across all EUDAT services.

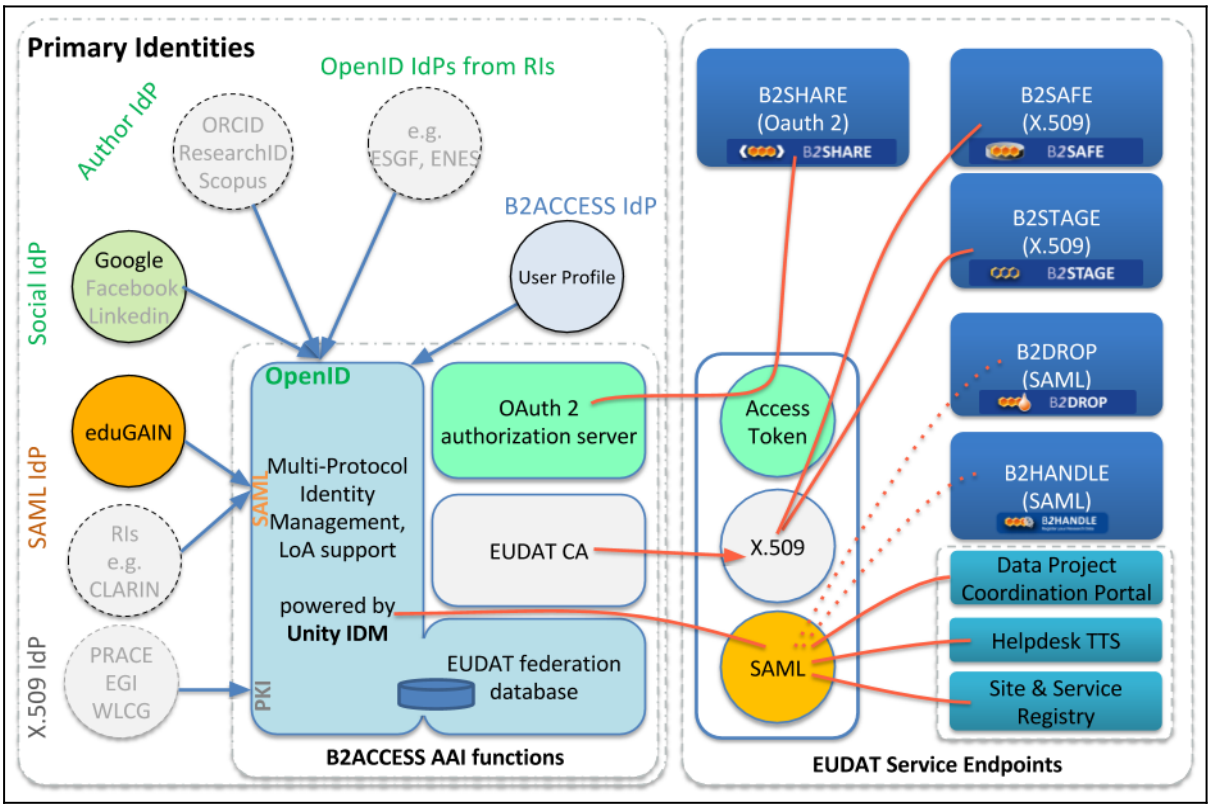

Figure 1. B2ACCESS Components, supported identity providers and compatible EUDAT services

General Service requirements - common to all service providers

EUDAT service providers relying on B2ACCESS for authentication must conform to certain data policies and

EUDAT receives funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 654065.

Table 1. Preproduction and production endpoints for the three different B2ACCESS-supported technologies

#	Protocol	Function	integration URL	Production URL	Comment
1	SAML web	entityid	https://b2access-integration.fz-juelich.de/saml-idp/metadata	https://b2access.eudat.eu:/saml-idp/metadata	
2	SAML web	metadata	https://b2access-integration.fz-juelich.de/saml-idp/metadata	https://b2access.eudat.eu:/saml-idp/metadata	
3	SAML web	SLO (single logout)	https://b2access-integration.fz-juelich.de/saml-idp/SLO-WEB	https://b2access.eudat.eu:/saml-idp/SLO-WEB	
4	OAuth2	Authorization Grant	https://b2access-integration.fz-juelich.de/oauth2-as/oauth2-authz	https://b2access.eudat.eu:/oauth2-as/oauth2-authz	Does not support POST - use GET
5	OAuth2/OIDC	Access Token	https://b2access-integration.fz-juelich.de/oauth2/token	https://b2access.eudat.eu:/oauth2/token	Supports the authorization code, implicit (only if permitted by an administrator) and client credentials flows; the resource owner flow is not supported. See the OAuth2 Background Information section below.
6a	OAuth2/OIDC	Token Information/Validation	https://b2access-integration.fz-juelich.de/oauth2-/tokeninfo	https://b2access.eudat.eu:/oauth2/tokeninfo	
6b	OIDC	User information	https://b2access-integration.fz-juelich.de/oauth2/userinfo	https://b2access.eudat.eu:/oauth2/userinfo	
6c	OIDC	Configuration Request	https://b2access-integration.fz-juelich.de/oauth2/.well-known/openid-configuration	https://b2access.eudat.eu:/oauth2/.well-known/openid-configuration	
6d	OIDC-JWK	JSON Web key	https://b2access-integration.fz-juelich.de/oauth2/jwk	https://b2access.eudat.eu:/oauth2/jwk	
7	Web	CA web services endpoint		https://b2access.eudat.eu:8445/ca/o/delegateduser	**NOTE PORT 8445, not 443.** OAuth protected.
8	Web	User portal	https://b2access-integration.fz-juelich.de/home/home	https://b2access.eudat.eu/home/home	Note port is standard port.
9	Web	Cert demo - user certificate "portal"		https://b2access.eudat.de:8445/oauth-demo/get_token.jsp	Useful as a demo but needs to be prettier if used by end-users.

Note that endpoints 7 and 10 are on port 8445 (inbound) which may be tricky to get opened in corporately managed firewalls as one will sometimes need to persuade IT staff of the merits of opening the port or enabling it in proxies.

User Name(s) and Attributes Released to Services

If the service is using SAML to authenticate, a SAML attribute statement is released about the subject. If using X.509, the same SAML attribute is embedded *as an extension inside the certificate* and can be extracted to use for authorisation decisions and account mappings.

The OID (object identifier) used for the extension is **iso(1) identified-organization(3) dod(6) internet(1) private(4) enterprise(1) globus(3536).1.1.1.12**

Services MUST NOT hardcode the value of the extension but should make the location configurable.

There are two user names used: one is the unique identifier assigned to the user within the EUDAT database, and the other, and more meaningful, is the Distinguished Name (DN) assigned to the user (this is the same DN as used in X.509 and LDAP). The former is a hex string (such as **01234567-89ab-cdef-0123-4567890abdef**, i.e. 32 hex digits in all). The latter is a full DN containing the unique identifier as one of the commonName (CN) parts, such as **/C=EU/O=EUDAT/OU=B2ACCES/CN=01234567-89ab-cdef-0123-4567890abdef/CN=user@example.edu**. There are two CNs in this DN; while the first is the unique identifier, the second is the principal obtained from the (first) external IdP the user used to create their account. (For the purists, yes, it would have been better to use domainComponent instead of C, O, OU.)

Technical details

The DN user name used by B2ACCESS is a string representation of the actual DN; the actual DN is encoded using either printableString or UTF8. The principal would have to use UTF8 encoding, as '@' is not a valid character in printableString. Services SHOULD NOT rely on particular encoding types used to encode the DN. Services MAY rely on the string representation.

The subject of the SAML assertion is currently the persistent identifier (e.g. **01234567-89ab-cdef-0123-4567890abdef**).

EUDAT

operational requirements. This section forms a quick checklist.

- The services MUST comply with EUDAT data processing policies: especially make sure they only use the attributes they need, and do not pass the attributes on (at least not without permission from the user) to parties outside EUDAT.
- The system MUST be run using best practices and MUST be secured with an IGTF or commercial X.509 certificate - under no circumstances are homemade, self-signed, or snake-oil certificates acceptable.
  - Best practices include (but are not limited to) practices for running secure systems (patching, monitoring, not running superfluous services) and for running systems complying with the IT requirements of the hosting organisations
  - Best practices also means best practices for system administration such as respecting user privacy and data protection rules
- Browser-facing hosts SHOULD be using browser-friendly certificates, i.e. issued by a certificate authority (CA) which is available in browsers' key stores.
  - These certificates are typically available from commercial CAs like DigiCert, Comodo, QuoVadis, etc, or via NRENs (who in turn get them from commercial CAs)
- The system MUST have have proper time settings (e.g. using NTP).
- Note that there will be requirements on production services, including monitoring, security and availability.

B2ACCESS Services Endpoints

As mentioned above, B2ACCESS can act as an identity provider for EUDAT services using any one of three different technologies. The table below shows the endpoints for the different technologies for both the staging (integration) and production service. The rest of this page explains how to use them; some familiarity with the relevant standards may be helpful.

EUDAT receives funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 654065.

The table below shows the attributes being published.

Table 2. SAML attributes published by B2ACCESS

Name	Mandatory	Multi-valued	Description
urn:oid:2.5.4.49 distinguishedName	YES	2	The DN as described above (it occurs twice, once with the OID as attribute name and once with distinguishedName)
unity:persistent	YES	1	The persistent identifier as described above
urn:oid:2.5.4.3 cn	YES	0..2	Common name. Occurs twice.
urn:oid:1.2.840.113549.1.9.1 userName	YES	0..2	Principal. A single value of the formuser@domain, where domain is (typically) a DNS-like subdomain representing the security domain of the user (e.g., "osu.edu") and user is generally a username, NetID, UserID, etc. of the sort typically assigned for authentication to network services within the security domain. Occurs twice.
urn:oid:2.5.4.10 o	YES	0..2	organisational affiliation. Occurs twice
email	YES	1	email address
memberOf	NO	0..*	A list of strings denoting group memberships and roles within those groups. As of release 1 of B2ACCESS, only service-related roles are used (such as /eudat:b2safe and /eudat:b2share) The service will perform the authorisation decision based on these roles. Note that a service role SHOULD NOT be needed for basic use of a service because users by default do not get any service roles and they should have some default use of the CDI without asking for additional permissions. The extra roles are meant for privileged use.

SAML for authentication - setting up a SAML service

In this section two sets of instructions for SAML implementations are provided: (1) SimpleSAMLphp and (2) Shibboleth. There is also a "common" section which applies regardless of which implementation you use.

Common

Your SP needs to be *registered* with B2ACCESS. If it is not, B2ACCESS will refuse to release attributes to it. In this case, you will get a message:

```
ERROR
SAML service got an invalid request.
If you are a user then you can be sure that the web application you was using previously is
either misconfigured or buggy.
If you are an administrator or developer, here the details of the error follows:
eu.unicore.samly2.exceptions.SAMLRequesterException: Issuer is not among trusted:
https://ganesha.esc.rl.ac.uk/eudat/shibboleth-sp
Caused by: eu.unicore.samly2.exceptions.SAMLRequesterException: Issuer is not among trusted:
https://ganesha.esc.rl.ac.uk/eudat/shibboleth-sp
```

EUDAT receives funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 654065.

The best way to register a service is to submit a ticket to the helpdesk, as it needs to be done by an administrator.

SimpleSAMLphp

SimpleSAMLphp is, as the name suggests, a simple way of setting up a SAML SP in Apache.

Use the PHP configuration available from the Annex, or alternatively follow the manual steps described below (with thanks to help/support from GOCDB supremo, David Meredith from STFC):

1. Log in as an administrator to SimpleSAMLPhp and under "Federation" locate the XML-to-PHP converter.
2. Download the EUDAT identity provider (IdP) metadata from EUDAT B2ACCESS metadata
    a. Paste the metadata into the box and convert.
    b. Then add the following entries after *entityID* and before *metadata-set*:
        i. Description,
        ii. OrganizationName,
        iii. Name,
        iv. OrganizationDisplayName,
        v. OrganizationURL,
        vi. Contacts.
    c. Note that a SimpleSAMLphp IdP selector actually displays the name entry; if it is not in the metadata then it will display the entity ID. So be sure to add a name entry which would be meaningful to users.
3. Add the metadata to your SimpleSAMLphp **saml20-idp-remote.php** file (e.g. **/var/simplesaml/metadata/saml20-idp-remote.php**)
4. In the **authsources.php** (e.g. **/var/lib/simplesaml/config/authsources.php**), ensure that the entry for "default-sp" has "'idp' => null" - this will enable the IdP selector. Alternatively, if EUDAT is your only source of identities, put "'idp' => 'https://unity.eudat-aai.fz-juelich.de:443/saml-idp/metadata'"

Shibboleth Service

Shibboleth is a SAML-based technology and protocol for authenticating to web services, very widely used by NRENs for national identity federations. While it started out differently, it currently uses the standard Web SSO profile of SAML from OASIS. Shibboleth is the name of a specific implementation of the technology but we use the term here in the generic sense that any interoperable implementation of SAML Web SSO will be supported by B2ACCESS. Since we focus on setting up a service provider, we focus on setting up the Apache module **mod_shib** for interacting with B2ACCESS. The **mod_shib** code is provided and documented for Apache but should be usable also with Tomcat.

Install

Install Apache 2.2 with **mod_shib**. Note that Shibboleth is configured slightly differently in different Apache versions; in our notes here we assume version 2.2. The **mod_shib** module is available in many Linux distributions, and built to match the Apache instance - check first if your distribution has made it available.

- Note that in addition to the **mod_shib** module which is linked into the Apache daemon, there is also a Shibboleth daemon, shibd. We configure it in the next section.
- Note that Shibboleth recommend that Apache is run with mpm_worker.
- Note that you must have host certificates for the host, issued to the name of the host the client is connecting to. (So if a client connects to https://b2access.eudat.eu/ but this name is hosted on a machine whose canonical name is (say) b2access-service01.fz-juelich.de, then the certificate must be issued to

EUDAT receives funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 654065.

b2access.eudat.eu). These are **not** the self signed certificates you generate for Shibboleth (below); they **MUST** be issued by a CA that issues browser-friendly certificates.

Configure the Shibboleth module

Edit Shibboleth2 configuration

Shibboleth configuration, by default, lives in **/etc/shibboleth**. To enable basic authentication, you need to configure only a single file, called **/etc/shibboleth/shibboleth2.xml** (but see the note below about making attributes available). A customisable version of **shibboleth2.xml** should already be available with your **mod_shib** package.

- Under *ApplicationDefaults*, set the *entityID* attribute to be the ID of the system you are configuring, as follows:
  - The entity ID should be an HTTPS URL.
    - It should use the hostname of the system you are setting up
    - It should have some sort of meaningful path (which doesn't need to point to anything actually on the system)
    - E.g. "https://ganesha.esc.rl.ac.uk/eudat/shibboleth-sp"
  - Add a REMOTE_USER="persistentid" attribute (see below)
- Locate the <Sessions> entry and change its attributes as follows:
  - Change (or add) checkAddress="true"
  - Change (or add) handlerSSL="true"
  - Change (or add) cookieProps="https"
- In EUDAT, you will normally only need one IdP, namely, the B2ACCESS service:
  - Within the section *<Sessions...>*, there should be an "SSO" entry. Customise this by setting the entityID attribute to "https://b2access.eudat.eu:443/saml-idp/metadata"
  - Comment out (or remove) any other attributes such as *discoveryProtocol* or *discoveryURL*.
  - Within the *<SSO>* entity, it probably says "SAML2 SAML1" - remove "SAML1", leaving only "SAML2".
- Following the *<Sessions...>* section there is an *<Errors...>* section. Set the *supportContact* attribute to point to a real email address. This address will be displayed to users when they encounter an error with the Shibboleth subsystem.
  - Set it your own email address while you are still setting up and testing the new system.
  - Once it is in (pre-)production, don't forget to change it to a helpdesk email address.
- In the following section *<MetadataProvider...>*, ensure that all are commented out, except for one:
  - <MetadataProvider type="XML" uri="https://b2access.eudat.eu:443/saml-idp/metadata"/>
    - Don't forget the slash at the end, closing the tag.

Validate the configuration

Next, run xmllint to check **shibboleth2.xml** for syntax errors:

```
xmllint shibboleth2.xml >/dev/null
```

If this returns without complaining, then the file is OK, and can be placed into /etc/shibboleth.

Certificate generation

Generate a self-signed certificate to sign and encrypt your SAML assertions; the SWITCH guidelines are a good starting point.

EUDAT receives funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 654065.

With keygen.sh (CentOS, RedHat, SuSE, OpenSUSE):

```
sudo /etc/shibboleth/keygen.sh -f -u shibd -h yourhost.example.org -y 3 -e
https://yourhost.example.org/shibboleth -o /etc/shibboleth/
```

Or if the keygen.sh script does not exist, use shib-keygen (Debian, Ubuntu):

```
sudo shib-keygen -f -u _shibd -h yourhost.example.org -y 3 -e
https://yourhost.example.org/shibboleth -o /etc/shibboleth/
```

This will place the certificate and private key in the **/etc/shibboleth** directory.

Note: If you are loath to run shib-keygen with root privileges, then run it as a different user, set a different output directory using the `-o` flag and install the key and certificate as root.

Notes on attributes

In order to make attributes available to your hosted portal/application, Shibboleth needs to be told. The file **attribute-map.xml**, also located in **/etc/shibboleth** by default, will have a few eduPerson attributes enabled by default.

```
<Attributes xmlns="urn:mace:shibboleth:2.0:attribute-map"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <!-- These mappings are specific to EUDAT. -->
    <Attribute name="unity:persistent" id="persistentid"/>
    <Attribute name="urn:oid:2.5.4.49" id="dn"/>
    <Attribute name="urn:oid:2.5.4.3" id="cn"/>
    <Attribute name="urn:oid:2.5.4.10" id="o"/>
    <Attribute name="email" id="email"/>
    <Attribute name="memberOf" id="memberOf"/>
</Attributes>
```

This set of instructions will tell Shibboleth to make the parameters sent by B2ACCESS available in the environment of your application. The name of the variable will be that of the 'id' attribute, so you can customise those names locally as needed (but note the 'persistentid' is referenced in Shibboleth2.xml described above, so if you change here you need to change it in the other file as well). The value will always be the value of the persistent id of the user who connects, but you *can* customise the name of the variable that holds the value.

Finalize shibd configuration

Start (or restart) shibd, e.g. **/etc/init.d/shibd start.**

Configure Apache

A detailed guide on all available options is provided on the Shibboleth wiki. This section provides some basic examples to get you started.

Make sure the **mod_shib** module is loaded:

```
LoadModule mod_shib /opt/shibboleth-sp/lib/shibboleth/mod_shib_22.so
```

EUDAT receives funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 654065.

Protect areas in your application:

```
<Location /some/path/in/your/webserver>
AuthType shibboleth
ShibRequestSetting requireSession 1
Require shib-session
</Location>
```

Protecting SCRIPTS with Shibboleth

Note that if you are trying to execute a script - which is normally the purpose - Apache will get confused because it will have two distinct handlers, one for Shibboleth and one for ExecCGI, and you might get one working and not the other. The recommended approach to solving this problem is as follows:

Your CGI-BIN directory (assuming here /usr/lib/cgi-bin, but in practice you would probably have a different path) should be protected as follows:

```
<Directory /usr/lib/cgi-bin>
        Options +ExecCGI -MultiViews -SymLinksIfOwnerMatch
        AllowOverride AuthConfig
        Order allow,deny
        Allow from all
</Directory>

ScriptAlias /eudat /usr/lib/cgi-bin/
```

Then, in your script directory, protect the scripts with .htaccess:

```
AuthType shibboleth
ShibRequestSetting requireSession 1
require shib-session
```

Now if you run printenv in your CGI-BIN directory (don't forget to give printenv execute permissions, say 0755) you will get the following:

```
#!/bin/sh
echo Content-Type: text/plain
echo
env
```

If you use a browser to go to (say) https://ganesha.esc.rl.ac.uk/eudat/printenv, you should first be directed to B2ACCESS to authenticate, and then you should see all the attributes, both those exported from B2ACCESS as well as the remaining connection parameters maintained by Shibboleth.

OAuth2 for authentication and authorisation

RFC6749 and RFC6750 provide additional background information on OAuth2.

An EUDAT service making use of OAuth for authentication or authorisation (or both) becomes in OAuth terminology a *client*. In this section we provide instructions how to build a client (optional), how to register one, and how then to use OAuth2.

EUDAT receives funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 654065.

OAuth2 Background Information

This section is intended to help the OAuth Client integrators use the B2ACCESS OAuth JAVA client API with Demo application. The processes documented here are for OAuth2; you can thus use any OAuth2 client. If you already have OAuth client code, skip to the section on how to register an OAuth client.

Background

OAuth2 is a framework for authorising a web client to access a resource, defined in RFC6749. In OAuth terminology there are four entities, as follows:

- The Resource (or Resource Server) which is being protected: e.g. the B2ACCESS CA for issuing certificates, or B2SHARE
- The Authorisation Server which issues access tokens and manages access to resources; this is provided as a part of B2ACCESS.
- The Client is the entity which seeks access to the resource; in our case it is typically a portal or other (web-based) entry point accessed by the user via a browser.
- The Resource Owner is the person who authenticates to the Authorisation Server to permit the access.

The (slightly simplified) workflow in B2ACCESS OAuth2 is as follows:

- In B2ACCESS, the "resource owner" is actually the end user.
- This does not mean that the end user "owns" the resource; it means that the end user authenticates (using a browser) to the Authorisation Server in the OAuth2 workflow.
- The Authorisation Server accepts any valid login by the end user as authorisation that the Client (portal) can obtain access.
- The Authorisation Server can then issue grants/tokens (see below) to the Client which the Client can use to access the Resource.
- The Resource validates the grants/tokens with the Authorisation Server to check that they are valid.
- If the Resource now needs to perform authorisation beyond accepting that the Authorisation Server is happy, then it asks the Authorisation Server for additional information about the end user, for whom the token was issued. It then uses this additional information to make an access control decision.

OAuth2 defines four different flows and allows for extensions (i.e. further flows to be defined). The four defined flows are as follows:

1. "Authorization Code" - an initial authorisation code is issued to the client and it then uses it to get the access token. This is the method used by B2SHARE.
2. "Implicit" - a simplified version of the first flow which skips the authorisation code issuance.
3. "Resource Owner Password Credentials" - another simplified flow where the Client is a privileged application with access to the Resource Owner credentials. *This is not used by B2ACCESS because it is currently not supported by the UNITY OAUTH2 server.*
4. "Client Authentication" - here an access token is issued directly to the client upon authentication (bypassing the authorisation code) upon the client presenting its credentials - typically BasicAuth (RFC2617) over a secure connection. *This is not supported by B2ACCESS because it is currently not supported by the UNITY OAUTH2 server*.
5. The Unity OAuth2 Authorisation Server supports a third (non-RFC) flow, called "OpenID hybrid"

If in doubt, use Authorization Code.

Some caveats:

EUDAT receives funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 654065.

- Clients need to be registered; see below.
- Note that UNITY's OAuth2 Authorisation Server supports only "Authorization Code" and "Implicit" (and the OpenID hybrid)
    - Implicit must be authorised by an administrator via the admin interface
    - If you attempt to use the Client Authentication - as one might, given that the example code uses it - the server will return an incorrect error message (it will say "code is required" rather than "unsupported"), and moreover the client code (in Java, see below) will fail to pick up the error message and report that the server unexpectedly closed the connection.

How to set up an OAuth Resource

Setting up an OAuth resource in B2ACCESS is extremely simple, as B2ACCESS uses RFC6750 bearer tokens only. If your service is set up already to use OAuth, you should be able to configure it using the endpoints in the endpoints table above; if not, you can easily build it yourself using the information in this section.

The basic idea of the bearer token (see RFC6749 section 7.1) is that the client sends an HTTP header including the 'Authorization' header with the bearer token. Extract this token, validate it using the process described below, and if it is valid, optionally query about user information if you need to authorise on additional user attributes. That's it!

Using the Token - Token Validation

The tokens used by B2ACCESS are RFC6750 Bearer tokens. The Resource should validate the tokens using the token validation endpoint (endpoint 6a in the endpoints table); its basic use is by adding 'Authorization: Bearer ' + tokenString to the HTTP header; cf. RFC6749 section 7.1. Note that anyone can validate a token as it is a bearer token: **there is no client id nor is client authentication required**.

The validation, if successful, returns a simple JSON structure like the following:

```
{"exp":1446552626,"sub":"18b81ab8-
b803-4a9a-80a4-8bbdbf5258b0","scope":["GENERATE_USER_CERTIFICATE","USER_PROFILE"],"client_id":n
ull}
```

If unsuccessful, the server returns a 401 Unauthorized HTTP return code.

Using the Token - User Info

Using the exact same method as with the token validation, but replacing "tokeninfo" with "userinfo", one can obtain the user information. See the attributes table above for the list of attributes returned. Again a 401 error is returned if the token is invalid, so this call could be used instead of the tokeninfo to validate the token.

How to register an OAuth client

In the Background section we have mentioned that an OAuth2 client must be registered. This is because otherwise a security risk is introduced, as, if any client can request access to resources, phishing is possible.

On the front page of the UNITY part of B2ACCESS (reference 8 in the B2ACCESS endpoints table above) there is a link in the top right corner called "Register a new account". **DO NOT LOG IN.** If you log in, the link will disappear. If you are logged in, log out. Click the link, then select the "OAuth client registration form."

Fill in the necessary details in the form and click *Submit and Accept:*

EUDAT receives funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 654065.

1. Required fields:

    a. Username and password, these are used to authenticate certain OAuth2 calls where basic authentication is needed.
    b. Email and contact details of the administrator of this OAuth2 client.
    c. Return URL, the OAuth2 client return URL. It is possible to have more than one return URL by having one on each line.
    d. Acceptance of the EUDAT policy (tickbox)
2. Optional:

    a. Client name, human readable display name for this OAuth2 client.
    b. Client logo, a logo for this OAuth2 client.
3. Make sure you remember the data you submitted, as you will need it to configure your OAuth client. If you forget it, ask your federation administrator to look it up for you (they can't see your password, though, nor can they change it).

    a. If you do not know who your federation administrator is, submit a ticket to the EUDAT helpdesk.

If you get a message saying "Form error: Please check the form for the invalid and missing mandatory values" then you've done something wrong. If the message doesn't tell you what is wrong, the missing or invalid entry should be highlighted when you click the message box away.

Somewhat confusingly, if the registration was successful, you are just returned to the Register Client box - there is no error message but also no explicit confirmation that the registration was successful.

X.509 for Authentication

OAuth is necessary for obtaining a certificate. The basic principle is that the Client obtains a certificate with which it can represent (or in fact impersonate) the end-user. Another principle is that the *private key* never crosses the wire, so it is generated (along with the CSR, the Certificate Signing Request). Please follow the steps below to generate a suitable X.509 certificate:

1. Obtain an Access Token using one of the OAuth flows above.
2. Generate a 1024-bit private key and a CSR. It doesn't matter which name (Distinguished Name) is asked for; the CA will ignore it and issue the certificate to the name to whom the access token was allocated. The test code uses '/CN=TestUser' which works fine.
3. POST the CSR to the endpoint number 7 in the table above, the CA web interface, with the following parameters:
    - Header should include the authorisation token, e.g. 'Authorization: Bearer abcdef0123456789'
    - Header should include content type, 'Accept-Encoding: identity'
    - The CSR must be included in the POST data with the name 'certificate_request' and the PEM encoded CSR as data, and the data MUST be URL encoded; otherwise the service will fail silently (return 200 OK but no data).
4. All going well, the certificate is returned immediately (it's an online CA and no additional approval is necessary) in the response using a text/plain mime type.
5. The certificate should then be paired up with the private key and they can now be used.

Additional things to note:

- How the certificate/key pair is used - and where it needs to be stored - is up to the Client; it is typically application-dependent.

EUDAT receives funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 654065.

- The certificate lifetime is 12 hours. The client is responsible for replacing the certificate when it is about to expire. However, at this time there is no renewal token, so a new certificate needs to be acquired.
- The token is a bearer token, so it can be used elsewhere from the client that obtained it. Use with caution.
- Curiously, the access token can be used several times, as long as it is valid.
- The CA performs no check of the key-size whatsoever. It works with 384 bits, which is the lowest OpenSSL will generate on the command line (which is too short to comply with most policies even for short-lived credentials), and it works with 16384 bits (which is much more than one would use even for a CA certificate)!
    - Use 1024 bits unless you have requirements to do otherwise.
- All the EUDAT services and clients that intend to use certificate-based authentication should trust the EUDAT online CA. This implies that you should include the appropriate (staging for test services; production for production services) CA certificate in your service and client trust-stores. The links to these B2ACCESS certificates and their fingerprints are shown on Table 3 below.

Certificate	SHA256 Fingerprint
Staging	DD:3E:1B:89:4B:CF:D6:58:76:D7:81:33:EA:AE:86:BE:75:D6:EB:1C:37:8A:1F:55:0C:1F:9C:61:C6:72:56:69
Production	C6:EA:7C:69:E1:39:B7:6E:B4:56:FD:4D:04:82:F8:65:CE:57:01:F7:3F:5A:1C:F8:C6:F8:DD:E5:FD:31:2E:58

Implementing a Portal Which Uses X.509 to Access Other Services

With the technology described in the previous sections, implementing a portal which relies on B2ACCESS is now all but straightforward. Just follow the instructions below:

1. Register the portal as an OAuth client
2. Implement B2ACCESS access control on the portal
3. When the user logs in, obtain an access token and use the access token to obtain a certificate.
4. Store the certificate and private key in a location which is unique to the user or session.
    - Example: /tmp/x509up_u9416dae7-80fc-4b91-9b11-3f5d4ce813af (where the number following "_u" is the Unity persistent id of the user that authenticated)
5. Now the user can access the portal and the portal can use the certificate and private key.

Implementing Non-Portal Access: Automated and Command Line Clients

Most of the text above is based on the premise that the *user* will access a *portal* and the portal will gateway all the user's actions via the user's browser. In other words, the user interacts via their browser with the portal and the portal in turn interacts accordingly with the actual service.

In this section we shall look at two other use cases; one where the user uses a command-line "thick client" to interact with a remote service (so not via a browser), and another where an automated service - an agent, or "robot" in IGTF-speak - accesses the remote service.

Implementing Command Line Access for Users

This type of access is seen as reserved for "expert" users: "normal" users would be using their browsers to access

EUDAT receives funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 654065.

services. An example could be access to iRODS via iCommands, or using a GridFTP command line client.

Currently, our best method for doing this is the following:

1. The user accesses the certificate portal (endpoint 10 in the table above)
2. The user authenticates using the usual web authentication workflow and obtains a certificate.
3. The user downloads/saves the certificate and private key into a file in a well-defined location (such as ~/.globus/usercert.pem and ~/.globus/userkey.pem or /tmp/x509up_u`id -u`) - the location will depend on the tool.
4. The user then proceeds to use the command line tool accordingly

Note the following:

- The private key by default is unprotected.
- The certificate contains the SAML extension with authorisation attributes
- There is no direct provisioning for renewal of certificates. Users will have to reauthenticate to the portal to obtain a new one.
- The lifetime of the certificate is fixed: it is set as a compromise between a credential which will not be revoked (so must be short-lived) and useful (so must be long-lived). The B2ACCESS team have considered other options but at the time of writing (March 2016) this is the only one offered.

Authorisation

(Note also the section "Future Work" below; while authorisation decisions are taken here at the service level, coming releases of B2ACCESS will enable also federation level access control decisions, in addition to making decisions at the service level.)

As of B2ACCESS 1.0, *all user attributes are published to all instances of all services.* Services must process only the attributes they strictly need and ignore the rest, in order to comply with eduGAIN requirements. The user attributes are published as follows:

- In SAML, they are embedded in the SAML assertion.
- In OAuth2, they are available by querying the userinfo endpoint (6b in the endpoints table above)
- In X.509, by extracting the SAML assertion from the certificate extension identified by the OID 1.3.6.1.4.1.3536.1.1.1.12.

Resources are expected to perform the following actions:

1. Authenticate users using B2ACCESS
2. Validate the user's authentication using best practices (using channels secured with host certificates, checking SAML signatures when available, validating tokens, checking certificate validity)
3. Obtain the required user attributes (in the interest of compliance with the eduGAIN Code of Conduct, servers must ignore attributes that they do not need, as mentioned above).
4. Use user attributes for authorisation, presentation, account mapping, logging, accounting, and/or statistical purposes as required.
5. Enforce access control correctly.

One option for simple authorisation is to just use the supported OAuth flows. The protected resource is set up as an OAuth Resource (see OAuth section above).

In future extensions, B2ACCESS will include an XACML framework available for SAML-based authorisation (i.e. SAML or X.509). Services are then expected to call out to a Policy Decision Point and to implement the Policy

EUDAT receives funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 654065.

Enforcement Points within the services.

Future Work

The B2ACCESS team aims to work on the following aspects in the future:

- Renewal tokens
  - A part of OAuth2, these too need supporting. They were supported in the Contrail OAuth server but have been lost in the migration to UNITY Authorisation Server.
- Implementing Automated Access (access for automated agents)
  - This will allow web based resources with dual protection: they can be accessed not only with federated access with B2ACCESS, but also with client certificates.
- Implementing a Reverse Proxy/Gateway with B2ACCESS.
  - This will enable a service provider to run (typically) an Apache web server as a front end to other services; it authenticates the users but proxies every HTTP request from the user's browser through to other services which may use different security methods.
- Deploying an XACML infrastructure to enable federation-level authorisation policies to be defined and implemented.
  - Services can also make their own decisions locally; of course the *enforcement* of the decision necessarily rests with the services.

Support

You can see an example of B2ACCESS integration with EUDAT's B2SHARE service on our hands-on training site.

You can also access our online training material for general information on B2ACCESS.

Support for B2ACCESS is available via the EUDAT ticketing system through the webform.

If you have comments on this page, please submit them though the EUDAT ticketing system.

Document Data

Version: 1.4.1

Authors:

Jens Jensen (STFC)

Willem Elbers (CLARIN)

Shiraz Memon (FZ Jülich)

Editors:

Hans van Piggelen (SURF)

Kostas Kavoussanakis (EPCC)

Sander Apweiler (FZ Jülich)

EUDAT receives funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 654065.

Annex: Sample PHP configuration to set up SAML Service Provider

Use the PHP configuration below to easily set up a SAML Service Provider in Apache.

```
$metadata['https://unity.eudat-aai.fz-juelich.de:443/saml-idp/metadata'] = array (
  'entityid' => 'https://unity.eudat-aai.fz-juelich.de:443/saml-idp/metadata',
  'description' =>
  array (
    'en' => 'eudat.eu',
    'cs' => 'eudat.eu',
  ),
  'OrganizationName' =>
  array (
    'en' => 'eudat.eu',
    'cs' => 'eudat.eu',
  ),
  'name' =>
  array (
    'en' => 'EUDAT B2ACCESS',
    'cs' => 'EUDAT B2ACCESS',
  ),
  'OrganizationDisplayName' =>
  array (
    'en' => 'eudat.eu',
    'cs' => 'eudat.eu',
  ),
  'url' =>
  array (
    'en' => 'http://eudat.eu/',
    'cs' => 'http://eudat.eu/',
  ),
  'OrganizationURL' =>
  array (
    'en' => 'http://eudat.eu/',
    'cs' => 'http://eudat.eu/',
  ),
  'contacts' =>
  array (
    0 =>
    array (
      'contactType' => 'technical',
      'givenName' => 'Shiraz',
      'surName' => 'MemonKuba',
      'emailAddress' =>
      array (
        0 => 'a.memon@fz-juelich.de',
      ),
    ),
  ),
  'metadata-set' => 'saml20-idp-remote',
  'SingleSignOnService' =>
  array (
    0 =>
```

EUDAT receives funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No. 654065.

```
    array (
      'Binding' => 'urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST',
      'Location' => 'https://unity.eudat-aai.fz-juelich.de:443/saml-idp/saml2idp-web',
    ),
    1 =>
    array (
      'Binding' => 'urn:oasis:names:tc:SAML:2.0:bindings:HTTP-Redirect',
      'Location' => 'https://unity.eudat-aai.fz-juelich.de:443/saml-idp/saml2idp-web',
    ),
  ),
  'SingleLogoutService' =>
  array (
    0 =>
    array (
      'Binding' => 'urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST',
      'Location' => 'https://unity.eudat-aai.fz-juelich.de:443/saml-idp/SLO-WEB',
      'ResponseLocation' =>
'https://unity.eudat-aai.fz-juelich.de:443/unitygw/SAMLSLOResponseConsumer',
    ),
    1 =>
    array (
      'Binding' => 'urn:oasis:names:tc:SAML:2.0:bindings:HTTP-Redirect',
      'Location' => 'https://unity.eudat-aai.fz-juelich.de:443/saml-idp/SLO-WEB',
      'ResponseLocation' =>
'https://unity.eudat-aai.fz-juelich.de:443/unitygw/SAMLSLOResponseConsumer',
    ),
    2 =>
    array (
      'Binding' => 'urn:oasis:names:tc:SAML:2.0:bindings:SOAP',
      'Location' => 'https://unity.eudat-aai.fz-juelich.de:443/saml-idp/SLO-SOAP',
    ),
  ),
  'ArtifactResolutionService' =>
  array (
  ),
  'keys' =>
  array (
    0 =>
    array (
      'encryption' => true,
      'signing' => true,
      'type' => 'X509Certificate',
      'X509Certificate' =>
'MIIGgjCCBWqgAwIBAgIHGJVrXCqgmTANBgkqhkiG9w0BAQsFADBYMQswCQYDVQQGEwJERTETMBEGA1UEChMKREZOLVZlcm
VpbjEQMA4GA1UECxMHREZOLVBLSTEiMCAGA1UEAxMZREZOLVZlcmVpbiBQQ0EgR3JpZCAtIEcwMTAeFw0xNDExMjYxNTEyM
TJaFw0xNTEyMjQxNTEyMTJaMHQxCzAJBgNVBAYTAkRFMRQwEgYDVQQKEwtHcmlkR2VybWFueTEnMCUGA1UECxMeRm9yc2No
dW5nc3plbnRydW0gSnVlbGljaCBHbWJIMSYwJAYDVQQDEx11bml0eS5ldWRhdC1hYWkuZnotanVlbGljaC5kZTCCAiIwDQY
JKoZIhvcNAQEBBQADggIPADCCAgoCggIBALo75fzUgoLdTmD8r1fKnDiPCEJQDzGonWAdc3A/oP6NglmcjU40zRjVJyamK1
ASNtN8URMAS4474LFHTGlO6bZTadm5RaiSya6MTxtteioZjCMbcwWcPxltphtvVNPAR8wsC6UQwpJCSYom3/aJFRkcj1IP+
l/EaIfXBnP/BBbFBKrjaTa0D1yFu4lWtgJH8cr5gxd0fPyNoZb9UlfEhv875TrDwQ0XGCxwbJSF10K/16aYspTGMzUh12Vu
oPqZ6Fole8/w+UJIMQVpXncvWaA+/BNXFLQLpUF2426ZBs77PL6JI8QbI1Y4JMowhtvXkqGP3Y82r/YvJLwpa7h/zXivPjY
Qg/Q1OfT6RlaUlrOjdiOurCPEgisW0MLN3RccenuGv3VzOhCCao+CAl+CEdep71REr+N1zCwbDKi3hS453H6CQVI6Wlat+G
WrJGIJGNWkgqetKY966NJAsa9MBoAN6k/+roYkqFGxaox6pn7uwSYHsDJ9/WYi7w66UnX9YzQzO6ksediKEC4ZyM4jgcVma
FI6Ev2V9h7p8qXT5yhFP68wON9kQfcDIrV+F2k+8UuRtrpI8/8kTlQKhtNG3GeYjmn8oT48lY0RaEgF39PgPHGNUDcDhkGc
0Mtc94RCl6lpYgutui0CuLrjMloOx0qy8dRvSEgBNX9Qir5F4q3RAgMBAAGjggIzMIICLzAMBgNVHRMBAf8EAjAAMA4GA1U
```


EUDAT receives funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 654065.

dDwEB/wQEAwIEsDAdBgNVHSUEFjAUBggrBgEFBQcDAgYIKwYBBQUHAwEwHQYDVR0OBBYEFHzC+hMxHtDjXEBWhj6uN4OAQy
c1MB8GA1UdIwQYMBaAFJbs3K2aw/5Qozwi5T3Cxf/K2SLGMCgGA1UdEQQhMB+CHXVuaXR5LmV1ZGF0LWFhaS5mei1qdWVsa
WNoLmRlMCoGA1UdIAQjMCEwEQYPKwYBBAGBrSGCLAEBAwEGMAwGCiqGSIb3TAUCAgEwgYMGA1UdHwR8MHowO6A5oDeGNWh0
dHA6Ly9jZHAxLnBjYS5kZm4uZGUvZ3JpZC1yb290LWNhL3B1Yi9jcmwvY2FjcmwuY3JsMDug0aA3hjVodHRwOi8vY2RwMi5
wY2EuZGZuLmRlL2dyaWQtcm9vdC1jYS9wdWIvY3JsL2NhY3JsLmNybDCB0wYIKwYBBQUHAQEEgcYwgcMwMwYIKwYBBQUHMA
GGJ2h0dHA6Ly9vY3NwLnBjYS5kZm4uZGUvT0NTUC1TZXJ2ZXIvT0NTUDBFBggrBgEFBQcwAoY5aHR0cDovL2NkcDEucGNhL
mRmbi5kZS9ncmlkLXJvb3QtY2EvcHViL2NhY2VydC9jYWNlcnQuY3J0MEUGCCsGAQUFBzAChjlodHRwOi8vY2RwMi5wY2Eu
ZGZuLmRlL2dyaWQtcm9vdC1jYS9wdWIvY2FjZXJ0L2NhY2VydC5jcnQwDQYJKoZIhvcNAQELBQADggEBAD4GJ9WMOas8AcW
go7ZM5VCQasUhK5VIEICowOTv78Q+GkBnnV7zu3wyvvpuzEZSsB5CXcwRH2YCJnRzK85CFMM12BeI+hbEbAQ4z+hyhLJ8Mg
uL7HoiVh4qzCUeI0U2yIhhxwfDSyVbMGSiu8EYyIAWdxySPb6+BEQWbgoZUMNEYKMX4/QO7G3qPNqSATzcnEWJ/KS8bR8ED
VBkV56aQRFyb5mwn2JZ7vEwKfl7Q5U2gczF86qSd94JX2T+I9DQCpbyuZpJohOU+t9obHFFyzgvdBDstMPd/oNtCnwpyc4E
xC9ar9Hy2RFFobEgcv3WMOa40sNVyMX9A3gX4lQeDOI=',
    ),
  ),
);

Read more

EUDAT receives funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No. 654065.