UFTP AUTHENTICATION SERVICE

UNICORE Team

Document Version:	1.1.0
Component Version:	1.1.0
Date:	03 11 2014

Contents

1 Installation **1**

1.1 Prerequisites 1

1.2 Basic configuration (memory etc) 2

1.3 Starting and stopping the service 2

2 Configuration **2**

2.1 User authentication 2

2.2 Configure the UFTPD server(s) to be accessed 4

2.3 Attribute mapping 5

3 Checking the installation **7**

4 Installing the Auth server in an existing UNICORE/X server **7**

The UFTP authentication service ("Auth server") is a simple HTTP-based service for authenticating users and initiating UFTP transfers. It is indended to be used with a standalone UFTP client.

- the Auth server checks the credentials provided by the user, and (if successful) maps the user to a distinguished name (DN).
- this DN is then assigned further attributes such as the user/group ID used by UFTPD, using the configured UNICORE attribute sources

The Auth server is based on the UNICORE Services Environment, and all usual UNICORE features and security configuration options are available as well. For example, the Auth server can be deployed behind a UNICORE Gateway, or it can be configured to use Unity for authenticating users.

One Auth server can be used to authenticate transfers for multiple UFTPD servers.

This manual focuses on the configuration items specific to the Auth server. If you need more in-depth information on general configuration issues, please refer to the UNICORE/X manual, available via unicore.eu

1 Installation

1.1 Prerequisites

The Auth server should be run as a non-root user (e.g. *unicore*).

It requires

- Java 1.7
- UFTPD 2.0 or later

The Auth server needs an X.509 certificate and truststore for communicating with UFTPD.

Users must be able to access the Auth server's https port. It is possible to deploy the Auth server behind a UNICORE Gateway (7.1 or later).

The UFTP Auth service comes with all required scripts and config files to be run standalone. To install, unzip the downloaded package into a directory of your choice.

Note
You can run the service in an existing UNICORE/X server (version 7.1 or later is required). Please see Section 4 below for details.

1.2 Basic configuration (memory etc)

The `startup.properties` configuration file contains basic settings such as the Java command, JVM memory etc. Please review it.

The Auth server host and port are configured in the `wsrflite.xml` configuration file:

```
  <!-- example host and port -->
  <property name="container.host" value="uftp.yoursite.com"/>
  <property name="container.port" value="9000"/>
```

Also in the `wsrflite.xml` configuration file, the server's X.509 private key and the truststore settings need to be configured.

1.3 Starting and stopping the service

Use the shell scripts in the `bin` folder to start or stop the service.

2 Configuration

The following items need to be configured:

- user authentication: configure the Auth server to authenticate users using username/password, ssh key or via Unity
- UFTPD server(s) to be accessed

2.1 User authentication

The enabled authentication options and their order are configured in `wsrflite.xml` (or `uas.config`), here we use the XML syntax of `wsrflite.xml`.

```
<property name="container.security.rest.authentication.order"
          value="FILE | SSHKEY | UNITY"/>
```

The available options can be combined!

2.1.1 Username-password file

To use a file containing username, password and the DN,

```
<property name="container.security.rest.authentication.order"
          value="FILE"/>

<property name="container.security.rest.authentication.FILE.class"
          value="eu.unicore.services.rest.security. ↩
              FilebasedAuthenticator"/>
<property name="container.security.rest.authentication.FILE.file"
          value="conf/rest-users.txt"/>
```

This configures to use the file `conf/rest-users.txt`. The file format is

```
#
# on each line:
# username:hash:salt:DN
#
demouser:<...>:<...>:CN=Demo User, O=UNICORE, C=EU
```

i.e. each line gives the username, the hashed password, the salt and the user's DN, separated by colons. To generate entries, i.e. to hash the password correctly, the *md5sum* utility can be used. For example, if your intended password is *test123*, you could do

```
$> SALT=(tr -dc A-Za-z0-9_ < /dev/urandom | head -c 16 | xargs)
$> echo "Salt is ${SALT}"
$> echo -n "${SALT}test123" | md5sum
```

which will output the salted and hashed password. Here we generate a random string as the salt. Enter these together with the username, and the DN of the user into the password file.

2.1.2 Unity authentication

You can also hook up with Unity, passing on the username/password and retrieving an authentication assertion.

```
<property name="container.security.rest.authentication.order"
          value="UNITY"/>

<!-- Unity settings -->
<property name="container.security.rest.authentication.UNITY.class"
          value="eu.unicore.services.rest.security. ↩
              UnitySAMLAuthenticator"/>
<property name="container.security.rest.authentication.UNITY. ↩
    address"
          value="https://localhost:2443/unicore-soapidp/ ↩
              saml2unicoreidp-soap/AuthenticationService"/>
<!-- validate the received assertions? -->
<property name="container.security.rest.authentication.UNITY. ↩
    validate"
          value="true"/>
```

2.1.3 SSH Key validation

This authentication option is based on the validation of a token using the user's public SSH key. The token will be checked, and if successful, the user will be assigned a distinguished name for later authorisation.

To use this option, you need to configure the REST authentication accordingly:

```
<!-- authN -->
<property name="container.security.rest.authentication.order"
          value="SSHKEY"/>
<!-- SSH key settings -->
<property name="container.security.rest.authentication.SSHKEY.class ↩
    "
          value="eu.unicore.uftp.authserver.authenticate. ↩
              SSHKeyAuthenticator"/>
<property name="container.security.rest.authentication.SSHKEY.file"
          value="conf/ssh-users.txt"/>
```

A file is required (in the example above we use *conf/ssh-users.txt*) which contains the mappings and the ssh public keys in a simple format:

```
# Example SSH users file used with the SSHKEY authentication method

#
#format: username:sshkey:DN
#
demouser:ssh-rsa keydata_was_omitted testkey:CN=Demo User, O= ↩
    UNICORE, C=EU
```

The SSH key is in the same one-line format used in the `.ssh/authorized_keys` file.

You can enter multiple lines per username, to accommodate the case that a user has different SSH keys available. For example

```
# Example SSH users file with multiple keys per user

demouser:ssh-rsa <...omitted keydata...>:CN=Demo User, O=UNICORE, C ↩
    =EU
demouser:ssh-dss <...omitted keydata...>:CN=Demo User, O=UNICORE, C ↩
    =EU
otheruser:ssh-rsa <...omitted keydata...>:CN=Other User, O=UNICORE, ↩
     C=DE
```

2.2 Configure the UFTPD server(s) to be accessed

For each UFTPD server you wish to authenticate users for, you'll need to configure the relevant properties in your container config file

The `authservice.servers` property is a list of server names. These should be meaningful, since users will need to use them, too. The other properties are used to configure the UFTPD command address and the UFTPD listen address. Please refer to the UFTPD configuration and manual for details.

For example, we want to configure two UFTPD servers named "CLUSTER" and "TEST":

```
<!-- configured UFTPD server(s) -->
<property name="authservice.servers" value="CLUSTER TEST"/>
<property name="authservice.server.CLUSTER.host" value="cluster. ↩
    your.org"/>
<property name="authservice.server.CLUSTER.port" value="64433"/>
<property name="authservice.server.CLUSTER.commandHost" value=" ↩
    cluster.your.org"/>
<property name="authservice.server.CLUSTER.commandPort" value ↩
    ="64434"/>
<property name="authservice.server.CLUSTER.ssl" value="true"/>
<property name="authservice.server.CLUSTER.description" value=" ↩
    Production UFTPD server on CLUSTER"/>

<property name="authservice.server.TEST.host" value="localhost"/>
<property name="authservice.server.TEST.port" value="64433"/>
<property name="authservice.server.TEST.commandHost" value=" ↩
    localhost"/>
<property name="authservice.server.TEST.commandPort" value ↩
    ="64434"/>
<property name="authservice.server.TEST.ssl" value="false"/>
<property name="authservice.server.TEST.description" value="Test  ↩
    UFTPD server"/>
```

To allow the Auth server access to the command port of UFTPD, you need to add an entry to UFTPD's ACL file. This is explained in the UFTPD manual.

2.3 Attribute mapping

After successful authentication, the user is assigned attributes such as the Unix account and group which is used for file access.

The Unix account and group are taken from the configured attribute sources (e.g. XUUDB). Since it is possible to access multiple UFTPD servers using a single Auth server, it may be required to configure different attributes for different UFTPD servers. This is easily possible using the file attribute source (map file).

It is also possible to control which directories and files that a user can access. This is done by configuring the allowed and/or the forbidden file path patterns.

The following map file entry gives a full example.

```
<entry key="CN=Demo User,O=UNICORE,C=EU">
   <attribute name="role">
```

```
        <value>user</value>
     </attribute>

     <!-- default Unix account and group -->
     <attribute name="xlogin">
        <value>somebody</value>
     </attribute>
     <attribute name="group">
        <value>users</value>
     </attribute>

      <!-- UFTP specific attributes -->

      <attribute name="uftp.CLUSTER.xlogin">
         <value>user1</value>
      </attribute>
      <attribute name="uftp.CLUSTER.group">
         <value>hpc</value>
      </attribute>

      <!-- optional rate limit (bytes per second) -->
      <attribute name="uftp.CLUSTER.rateLimit">
         <value>10M</value>
      </attribute>

      <!-- optional includes -->
      <attribute name="uftp.CLUSTER.includes">
         <value>/tmp/*:/work/*</value>
      </attribute>
      <!-- optional excludes -->
      <attribute name="uftp.CLUSTER.excludes">
         <value>/home/*:/etc/*</value>
      </attribute>

   </entry>
```

Here, the "CLUSTER" must match a configured UFTPD server, see also Section 2.2. Available attributes are

- xlogin, group: Unix account and group to be used for this user
- rateLimit: the number of bytes per second (per transfer) can be limited. You can use the units "K", "M", and "G" for kilo, mega or gigabytes, respectively.
- includes: file path patterns (separated by ":") that are allowed. If not given, all the user's files can be accessed.
- excludes: file path patterns (separated by ":") that are forbidden. If not given, no files are explicitely excluded.

3 Checking the installation

You can check that the server works using a simple HTTP client such as `curl` to access the Auth server's base URL, provided you have configured username/password authentication.

The command

```
$> curl -k https://<host:port>/rest/auth \
   -H "Accept: application/json" \
   -u username:password
```

should produce a JSON document containing information about the configured UFTPD servers, such as

```
{"TEST": {
  "description": "Default UFTPD server for testing",
  "gid": "users",
  "href": "https://localhost:9000/rest/auth/TEST",
  "uid": "somebody"
}}
```

Note
If you do not get any output, try adding the "-i" option to the curl command, most probably the username/password is incorrect.

4 Installing the Auth server in an existing UNICORE/X server

This option is interesting if you are already running a UNICORE installation and want to allow your users a simpler way of transferring files using the standalone UFTP client. This requires UNICORE/X version 7.1 or later!

- copy the jar files to the lib directory of UNICORE/X
- copy the XACML policy file `30uftpAuthService.xml` to the conf/xacml2Policies directory
- edit wsrflite.xml (or uas.config) and setup user authentication and UFTPD details as described above
- if you want to place the Auth server behind a UNICORE gateway for easy firewall transversal, you need to configure an entry in the Gateway connections config file, and set the container base URL property in the Auth servers `wsrflite.xml`