UNICORE Hardening Guide

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME
	2012-03-15		

Contents

1 Introduction

This guide will provide a list of actions to be taken in any UNICORE installation in order to harden it and make full use of all security features within UNICORE. Only a hardened UNICORE installation should be exposed to the outside world and participate in Grid infrastructures.

Each section is structured in a similar manner. First, we will mention the general security measure to take in order to make UNICORE systems secure. Following this will be a recipe of how to implement counter measures. Lastly, we will describe how the measure effectively works and the affected components.

2 Do not use nor accept demo certificates

As a general rule, keystores and truststore must not contain test certificates such as the Demo user or server certificates. The same holds for the Demo CA certificate. These certificates have been created for quickly getting a new UNICORE installation up and running. Once the installations are put into real use, these certificates must not be used anymore.

Demo certificates must also be removed from the XUUDB or UVOS services, if they have been added to these before.

2.1 Implementation

- Remove all demo CA certificates from all truststores in both clients and servers
- Remove all demo user keys from all keystores
- Remove all demo server keys from all keystores
- Do not map demo certificates in the XUUDB or UVOS services

2.2 Explanation of the counter measure

If a site does not accept publicly available certificates, site administrators can be sure who is accessing their resources. Accepting the publicly available demo certificates only authenticates these certificates, but no individual user.

A similar thing holds for clients, that cannot be certain about the identity of a server, if they accept demo certificates.

2.3 Affected components

- All

3 Secure JMX Services

The Java Management Extensions (JMX) can be enabled on several UNICORE components. They allow an administrator to look at the internal state of UNICORE services and, to an extent, change the UNICORE configuration during runtime.

3.1 Implementation

As a general rule it is always good to deactivate unused services. So if you don't intend to monitor or manage your server using JMX, then deactivate it. JMX usually listens for connections at two levels: locally on a socket, and on the network. Listening on the network can easily be disabled by commenting the respective lines in the server startup script bin/start.sh.

Current UNICORE installations (>=6.4.x)disable JMX via network, if the port is unset.

```
#
# JMX remote management port
#
# Leave empty (="") to disable JMX
#
JMX_PORT=9128
```

which is later on used to enable or disable JMX

```
if [ "" != "$JMX_PORT" ]
  then
  #
  # enable JMX (use jconsole to connect)
  #
  OPTS=$OPTS" -Dcom.sun.management.jmxremote"
  #enable JMX remote access protected by username/password
  OPTS=$OPTS" -Dcom.sun.management.jmxremote.port=${JMX_PORT} -Dcom.sun.management. ↩
      jmxremote.authenticate=true"
  OPTS=$OPTS" -Dcom.sun.management.jmxremote.ssl=false"
  OPTS=$OPTS" -Dcom.sun.management.jmxremote.password.file=conf/jmxremote.password"
  #
  #make sure jmx password file is owner read-only to avoid startup problems
  chmod go-rw conf/jmxremote.password
fi
```

Older installations (up to 6.3.x) have a slightly different setup.

```
#
# enable JMX (use jconsole to connect)
#
OPTS=$OPTS" -Dcom.sun.management.jmxremote"
#enable JMX remote access protected by username/password
OPTS=$OPTS" -Dcom.sun.management.jmxremote.port=9128 -Dcom.sun.management.jmxremote. ↩
    authenticate=true"
OPTS=$OPTS" -Dcom.sun.management.jmxremote.ssl=false"
OPTS=$OPTS" -Dcom.sun.management.jmxremote.password.file=conf/jmxremote.password"
```

The most relevant setting here is -Dcom.sun.management.jmxremote. This must not occur in the Java command line.

If you need to avoid local JMX connections, then the following unsupported setting may help (Source: http://stackoverflow.com/-questions/1255049/disabling-local-jmx-connections-on-jvm). It needs to be added to the Java command line, i.e. the OPTS variable.

```
-XX:+DisableAttachMechanism
```

If you want to leave JMX activated, we strongly suggest the following settings to secure the service.

JMX service passwords are usually configured in the jmxremote.password file in the conf directory of the respective component. If you are using packaged installations provided for some Linux distributions, these files are located in /etc/unicore/<component>/jmxremote.password.

Two roles are defined in this file: monitorRole and controlRole. Each of these has a password associated with it. Change them to be unique, even a randomized string is better than the default passwords.

3.2 Affected components

- Gateway
- UNICORE/X
- Registry

4 Enable access control in the registry

By default, any service can be registered in the registry. This eases the administrative effort required to run an infrastructure. However, one should be sure about who can register services in the registry.

4.1 Implementation

At the moment, it is only possible to secure the Registry against unauthorized entries by untrusted clients or services. In the uas.config file of the registry, set the following property to true.

```
#
# NEW in 6.2.1: control access control on the Registry
# (off by default, for backwards compatibility)
#
uas.security.accesscontrol.Registry=true
```

This setting requires the server to be registered in an attribute source for the registry. A variety of attribute sources can be configured in UNICORE. As the registry usually comes with no attribute sources configured, the easiest way may be to use a plain file for this. This is described in the following. These changes are required in the uas.config file:

```
uas.security.attributes.order=FILE

#
# configuration of the 'FILE' attribute source
#
uas.security.attributes.FILE.class=eu.unicore.uas.security.file.FileAttributeSource
# path to the mapfile
uas.security.attributes.FILE.file=/etc/unicore/registry/simpleuudb
# how to match entries: strict or regexp
uas.security.attributes.FILE.matching=strict
```

The uas.security.attributes.FILE.file property references /etc/unicore/registry/simpleuudb, which should contain an entry for each of the service containers that should be allowed to add or update entries in the registry. The role for each of these entries must be *server*.

```
<?xml version="1.0" encoding="UTF-8"?>
<fileAttributeSource>
  <entry key="CN=zam052v01.zam.kfa-juelich.de,OU=EMI UNICORE unicorex,OU=Forschungszentrum  ↩
      Juelich GmbH,O=GridGermany,C=DE">
    <attribute name="role">
      <value>server</value>
    </attribute>
  </entry>
  <entry key="CN=....">
    <attribute name="role">
      <value>server</value>
    </attribute>
  </entry>
</fileAttributeSource>
```

The DN needs to be put in RFC2253 notation, which can be retrieved using openssl on the certificate file (PEM):

```
openssl x509 -noout -subject -nameopt rfc2253 -in <unicorex.pem>
```

Put multiple `entry` elements for multiple servers.

If you are already using an XUUDB or UVOS service as an attribute source, you can configure this analogous to the UNICORE/X configuration of attribute sources. Don't forget to add the required certificates or DNs to the XUUDB or UVOS and give them the server role (set attribute role to server).

4.2 Explanation of the counter measure

Allowing only registered services to add entries in the registry avoids this problem of arbitrary services being advertised to the user or malicious users removing valid entries from the registry.

This counter measure is only really effective in conjunction with Section 5.

4.3 Affected components

- Registry
- Eventually the clients, when they consume maliciously advertised services

5 Only allow the Gateway to connect to the UNICORE/X and Registry

In some situations, connections to UNICORE services are possible from other machines than just the one where the UNICORE gateway resides. In those situations, anyone with direct access to the same network, i.e. mostly internal staff, could access the services directly, bypassing the gateway. This should be avoided by measures described in this section.

5.1 Implementation

The UNICORE/X can be configured to request signed consignor assertions, checking the signatures with the Gateway's public key. Thus, even if the service container can be reached via the network, it will be impossible to send "fake" consignor assertions without knowing private information of the Gateway.

The respective setting in the uas.config file is:

```
#
# Verify the signature of the SAML assertions issued by the gateway
#
# If your UNICORE/X server is accessible from the "outside", you should
# set this to "true", and configure the gateway to sign its assertions
# Otherwise, a smart user could pretend to be somebody else by adding
# fake unsigned assertions to his messages
#
uas.security.consignor.checksignature=true
```

The Gateway needs to be configured to sign consignor assertions in its gateway.properties file:

```
#
# If the network behind gateway is secure leave the following settings unchanged
# (yes - it is the usual case). However if you wish to secure consignor
# assertions issued by gateway by signing them, change the following to true.
signConsignorToken = true
```

The Gateway certificate for verification is established at server startup when the UNICORE/X connects to the Gateway for the first time.

If you want to configure the precise Gateway certificate that will be accepted, then you can place it in a the truststore file and set the alias with the following setting in uas.config:

```
# If uas.security.consignor.checksignature==true then the gateway certificate
# for checking signatures is needed. It can be obtained either automatically
# if uas.onstartup.wait==true by checking the gw's address, or the certificate
# can be placed in the truststore under an alias which should be entered in the
# following property:
#uas.security.gateway.alias=
```

If you are really paranoid, you could additionally secure the services against connections from entities other than the gateway by restricting access to the ports at the network level, e.g. only allow connections on the respective port from the gateway machine.

This Linux IPTables line may serve as an example:

```
iptables -A INPUT ! -s <gateway ip or hostname> -p <service port> -j DROP
```

Beware that the exclamation mark (!) needs to be escaped in some shells.

5.2 Affected components

- UNICORE/X (up to 6.3.2)
- Registry (up to 6.3.2)
- Workflow Service (up to 6.3.2)
- Service Orchestrator (up to 6.3.2)

6 Be aware of permissions you grant to UNICORE administrators

This section is only valid for those sites where non-root users are granted permissions to administer the TSI, which needs to run with root privileges. Of course, you should trust your TSI administrators anyway, but if you only want to allow them to e.g. start and stop the TSI, then you should restrict their permissions accordingly.

6.1 Implementation

Only allow UNICORE admins to use particular parameters when acting as root. This can be achieved by restricting the set of parameters allowed to be passed to specific TSI scripts in the /etc/sudoers file.

```
tsi-admin ALL=/opt/unicore/tsi/bin/kill_tsi ""
```

This would disallow the tsi-admin user to pass any arguments to the kill_tsi script, thus only allowing to kill the TSI installation residing in /opt/unicore/tsi. Similarly, in case multiple TSIs are running on the same machine, one could restrict the user to kill only particular ones.

```
tsi-admin1 ALL=/opt/unicore/tsi/bin/kill_tsi /etc/unicore/tsi1/conf1
tsi-admin2 ALL=/opt/unicore/tsi/bin/kill_tsi /etc/unicore/tsi1/conf1
tsi-admin2 ALL=/opt/unicore/tsi/bin/kill_tsi /etc/unicore/tsi2/conf2
```

In this case, tsi-admin2 can kill two TSIs, while tsi-admin1 can only kill one of them. Of course, similar entries would have to be made for starting TSIs.

6.2 Affected components

- TSI

7 Activate CRL checking

Certificate Revocation Lists (CRLs) are a means for a certificate authority (CA) to revoke certificates that it issues. These lists can be retrieved by entities trusting the CA to issue valid certificates and used to reject certificates that are on the list.

There are many reasons why certificates may need to be revoked. Most obviously this is required if the private key material of a user that belongs to the certificate has been obtained by a third party or is believed to have been obtained.

Ignoring CRLs results in certificates being accepted until their official end of life. While this is true for most certificates, as they will never be revoked during their lifetime, it will not cover the few cases where malicious third parties have obtained the certificate of an entity and make use of it.

7.1 Implementation

Create a configuration file for CRL checking and reference it at application startup as a system property. The full documentation is available in the Gateway documentation. Any component using UNICORE security libraries is capable of supporting CRL checking.

Clients can benefit from checking CRLs as they can recognize whether server certificates have been revoked.

7.2 Affected Components

- Gateway (most prominently)
- others

8 Keep your Java versions updated

The Java programming language and runtime environment form the basis for many of UNICORE's components. Be sure to keep the runtime environment up to date.

8.1 Implementation

Update Java to the latest version, at lest JRE 6 update 24 to the best of our knowledge.

8.2 Explanation of the counter measure

JRE 6 update 24 and subsequent versions do not contain the vulnerability.

8.3 Affected Components

- UNICORE/X
- Registry
- XUUDB
- Gateway