GT 4.0: Security: Authorization Framework

GT 4.0: Security: Authorization Framework

Table of Contents

List of Tables

Chapter 1. GT 4.0 Security: Key Concepts

1. Overview

GSI uses public key cryptography (also known as asymmetric cryptography) as the basis for its functionality. Many of the terms and concepts used in this description of GSI come from its use of public key cryptography.

For a good overview of GSI contained in the Web Services-based components of GT4, see Globus Toolkit Version 4 Grid Security Infrastructure: A Standards Perspective[1].

A reference for detailed information about public key cryptography is available in the book Handbook of Applied Cryptography [2], by A. Menezes, P. van Oorschot, and S. Vanstone, CRC Press, 1996. Chapter 8 [3] of this book deals exclusively with public key cryptography.

The primary motivations behind GSI are:

- The need for secure communication (authenticated and perhaps confidential) between elements of a computational Grid.
- The need to support security across organizational boundaries, thus prohibiting a centrally-managed security system.
- The need to support "single sign-on" for users of the Grid, including delegation of credentials for computations that involve multiple resources and/or sites.

2. Conceptual Details

2.1. Public Key Cryptography

The most important thing to know about public key cryptography is that, unlike earlier cryptographic systems, it relies not on a single key (a password or a secret "code"), but on two keys. These keys are numbers that are mathematically related in such a way that if either key is used to encrypt a message, the other key must be used to decrypt it. Also important is the fact that it is next to impossible (with our current knowledge of mathematics and available computing power) to obtain the second key from the first one and/or any messages encoded with the first key.

By making one of the keys available publicly (a public key) and keeping the other key private (a private key[4]), a person can prove that he or she holds the private key simply by encrypting a message. If the message can be decrypted using the public key, the person must have used the private key to encrypt the message.

Important: It is critical that private keys be kept private! Anyone who knows the private key can easily impersonate the owner.

2.2. Digital Signatures

Using public key cryptography, it is possible to digitally "sign" a piece of information. Signing information essentially means assuring a recipient of the information that the information hasn't been tampered with since it left your hands.

[1] GT4-GSI-Overview.pdf
[2] http://www.cacr.math.uwaterloo.ca/hac/
[3] http://www.cacr.math.uwaterloo.ca/hac/about/chap8.pdf
[4] #priv-key

To sign a piece of information, first compute a mathematical hash of the information. (A hash is a condensed version of the information. The algorithm used to compute this hash must be known to the recipient of the information, but it isn't a secret.) Using your private key, encrypt the hash, and attach it to the message. Make sure that the recipient has your public key.

To verify that your signed message is authentic, the recipient of the message will compute the hash of the message using the same hashing algorithm you used, and will then decrypt the encrypted hash that you attached to the message. If the newly-computed hash and the decrypted hash match, then it proves that you signed the message and that the message has not been changed since you signed it.

2.3. Certificates

A central concept in GSI authentication is the *certificate*. Every user and service on the Grid is identified via a certificate, which contains information vital to identifying and authenticating the user or service.

A GSI certificate includes four primary pieces of information:

- A subject name, which identifies the person or object that the certificate represents.
- The public key belonging to the subject.
- The identity of a Certificate Authority (CA) that has signed the certificate to certify that the public key and the identity both belong to the subject.
- The digital signature of the named CA.

Note that a third party (a CA) is used to certify the link between the public key and the subject in the certificate. In order to trust the certificate and its contents, the CA's certificate must be trusted. The link between the CA and its certificate must be established via some non-cryptographic means, or else the system is not trustworthy.

GSI certificates are encoded in the X.509 certificate format, a standard data format for certificates established by the Internet Engineering Task Force (IETF). These certificates can be shared with other public key-based software, including commercial web browsers from Microsoft and Netscape.

2.4. Mutual Authentication

If two parties have certificates, and if both parties trust the CAs that signed each other's certificates, then the two parties can prove to each other that they are who they say they are. This is known as *mutual authentication*. GSI uses the Secure Sockets Layer (SSL) for its mutual authentication protocol, which is described below[5]. (SSL is also known by a new, IETF standard name: Transport Layer Security, or TLS.)

Before mutual authentication can occur, the parties involved must first trust the CAs that signed each other's certificates. In practice, this means that they must have copies of the CAs' certificates--which contain the CAs' public keys--and that they must trust that these certificates really belong to the CAs.

To mutually authenticate, the first person (*A*) establishes a connection to the second person (*B*).

To start the authentication process, *A* gives *B* his certificate.

The certificate tells *B* who *A* is claiming to be (the identity), what *A*'s public key is, and what CA is being used to certify the certificate.

[5] #s-security-key-delegation

B will first make sure that the certificate is valid by checking the CA's digital signature to make sure that the CA actually signed the certificate and that the certificate hasn't been tampered with. (This is where *B* must trust the CA that signed *A*'s certificate.)

Once *B* has checked out *A*'s certificate, *B* must make sure that *A* really is the person identified in the certificate.

B generates a random message and sends it to *A*, asking *A* to encrypt it.

A encrypts the message using his private key, and sends it back to *B*.

B decrypts the message using *A*'s public key.

If this results in the original random message, then *B* knows that *A* is who he says he is.

Now that *B* trusts *A*'s identity, the same operation must happen in reverse.

B sends *A* her certificate, *A* validates the certificate and sends a challenge message to be encrypted.

B encrypts the message and sends it back to *A*, and *A* decrypts it and compares it with the original.

If it matches, then *A* knows that *B* is who she says she is.

At this point, *A* and *B* have established a connection to each other and are certain that they know each others' identities.

2.5. Confidential Communication

By default, GSI does not establish confidential (encrypted) communication between parties. Once mutual authentication is performed, GSI gets out of the way so that communication can occur without the overhead of constant encryption and decryption.

GSI can easily be used to establish a shared key for encryption if confidential communication is desired. Recently relaxed United States export laws now allow us to include encrypted communication as a standard optional feature of GSI.

A related security feature is communication integrity. Integrity means that an eavesdropper may be able to read communication between two parties but is not able to modify the communication in any way. GSI provides communication integrity by default. (It can be turned off if desired). Communication integrity introduces some overhead in communication, but not as large an overhead as encryption.

2.6. Securing Private Keys

The core GSI software provided by the Globus Toolkit expects the user's private key to be stored in a file in the local computer's storage. To prevent other users of the computer from stealing the private key, the file that contains the key is encrypted via a password (also known as a passphrase). To use GSI, the user must enter the passphrase required to decrypt the file containing their private key.

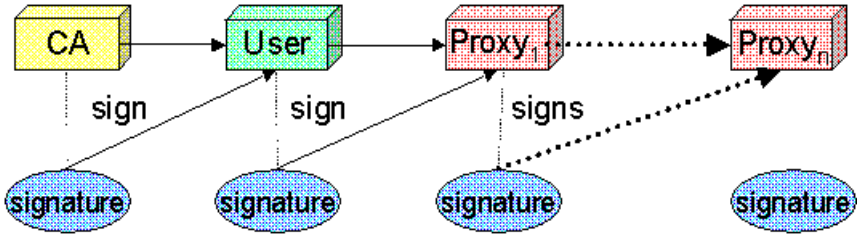
We have also prototyped the use of cryptographic smartcards in conjunction with GSI. This allows users to store their private key on a smartcard rather than in a file system, making it still more difficult for others to gain access to the key.

2.7. Delegation, Single Sign-On and Proxy Certificates

GSI provides a delegation capability: an extension of the standard SSL protocol which reduces the number of times the user must enter his passphrase. If a Grid computation requires that several Grid resources be used (each requiring

mutual authentication), or if there is a need to have agents (local or remote) requesting services on behalf of a user, the need to re-enter the user's passphrase can be avoided by creating a *proxy*.

A proxy consists of a new certificate and a private key. The key pair that is used for the proxy, i.e. the public key embedded in the certificate and the private key, may either be regenerated for each proxy or obtained by other means. The new certificate contains the owner's identity, modified slightly to indicate that it is a proxy. The new certificate is signed by the owner, rather than a CA. (See diagram below.) The certificate also includes a time notation after which the proxy should no longer be accepted by others. Proxies have limited lifetimes.



The proxy's private key must be kept secure, but because the proxy isn't valid for very long, it doesn't have to kept quite as secure as the owner's private key. It is thus possible to store the proxy's private key in a local storage system without being encrypted, as long as the permissions on the file prevent anyone else from looking at them easily. Once a proxy is created and stored, the user can use the proxy certificate[6] and private key for mutual authentication without entering a password.

When proxies are used, the mutual authentication process differs slightly. The remote party receives not only the proxy's certificate (signed by the owner), but also the owner's certificate. During mutual authentication, the owner's public key (obtained from her certificate) is used to validate the signature on the proxy certificate. The CA's public key is then used to validate the signature on the owner's certificate. This establishes a chain of trust from the CA to the proxy through the owner.

Note

GSI, and software based on it (notably the Globus Toolkit, GSI-SSH, and GridFTP), is currently the only software which supports the delegation extensions to TLS (a.k.a. SSL). The Globus Alliance has worked in the GGF and the IETF to standardize this extension in the form of Proxy Certificates (RFC 3820) [http://www.ietf.org/rfc/rfc3820.txt].

3. Related Documents

- Globus Toolkit Version 4 Grid Security Infrastructure: A Standards Perspective[7]
- Handbook of Applied Cryptography [8]

GT 4.0 Security Glossary

C

Certificate Authority (CA) An entity that issues certificates.

[6] #proxy-cert
[7] GT4-GSI-Overview.pdf
[8] http://www.cacr.math.uwaterloo.ca/hac/

CA Certificate	The CA's certificate. This certificate is used to verify signature on certificates issued by the CA. GSI typically stores a given CA certificate in `/etc/grid-security/certificates/<hash>.0`, where <hash> is the hash code of the CA identity.
CA Signing Policy	The CA signing policy is used to place constraints on the information you trust a given CA to bind to public keys. Specifically it constrains the identities a CA is trusted to assert in a certificate. In GSI the signing policy for a given CA can typically be found in `/etc/grid-security/certificates/<hash>.signing_policy`, where <hash> is the hash code of the CA identity. For more information see [add link].
certificate	A public key and information about the certificate owner bound together by the digital signature of a CA. In the case of a CA certificate the certificate is self signed, i.e. it was signed using its own private key.
Certificate Revocation List (CRL)	A list of revoked certificates generated by the CA that originally issued them. When using GSI this list is typically found in `/etc/grid-security/certificates/<hash>.r0`, where <hash> is the hash code of the CA identity.
certificate subject	A identifier for the certificate owner, e.g. "/DC=org/DC=doegrids/OU=People/CN=John Doe 123456". The subject is part of the information the CA binds to a public key when creating a certificate.
credentials	The combination of a certificate and the matching private key.

E

End Entity Certificate (EEC)	A certificate belonging to a non-CA entity, e.g. you, me or the computer on your desk.

G

GAA Configuration File	A file that configures the Generic Authorization and Access control GAA libraries. When using GSI this file is typically found in `/etc/grid-security/gsi-gaa.conf`.
grid map file	A file containing entries mapping certificate subjects to local user names. This file can also serve as a access control list for GSI enabled services and is typically found in `/etc/grid-security/grid-mapfile`. For more information see the Gridmap file[9].
grid security directory	The directory containing GSI configuration files such as the GSI authorization callout configuration and GAA configuration files. Typically this directory is `/etc/grid-security`. For more information see Grid security directory[10].
GSI authorization callout configuration file	A file that configures authorization callouts to be used for mapping and authorization in GSI enabled services. When using GSI this file is typically found in `/etc/grid-security/gsi-authz.conf`.

[9] http://www.globus.org/toolkit/docs/4.0/security/prewsaa/Pre_WS_AA_Public_Interfaces.html#prewsaa-env-gridmapfile
[10] http://www.globus.org/toolkit/docs/4.0/security/prewsaa/Pre_WS_AA_Public_Interfaces.html#prewsaa-env-gridsecurity

H

host certificate — An EEC belonging to a host. When using GSI this certificate is typically stored in `/etc/grid-security/hostcert.pem`. For more information on possible host certificate locations see the Credentials[11].

host credentials — The combination of a host certificate and its corresponding private key..

P

private key — The private part of a key pair. Depending on the type of certificate the key corresponds to it may typically be found in `$HOME/.globus/userkey.pem` (for user certificates), `/etc/grid-security/hostkey.pem` (for host certificates) or `/etc/grid-security/<service>/<service>key.pem` (for service certificates). For more information on possible private key locations see the Credentials[12]

proxy certificate — A short lived certificate issued using a EEC. A proxy certificate typically has the same effective subject as the EEC that issued it and can thus be used in its stead. GSI uses proxy certificates for single sign on and delegation of rights to other entities.

proxy credentials — The combination of a proxy certificate and its corresponding private key. GSI typically stores proxy credentials in `/tmp/x509up_u<uid>` , where <uid> is the user id of the proxy owner.

public key — The public part of a key pair used for cryptographic operations (e.g. signing, encrypting).

S

service certificate — A EEC for a specific service (e.g. FTP or LDAP). When using GSI this certificate is typically stored in `/etc/grid-security/<service>/<service>cert.pem`. For more information on possible service certificate locations see the Credentials[13].

service credentials — The combination of a service certificate and its corresponding private key.

T

transport-level security — Uses transport-level security (TLS) mechanisms.

trusted CAs directory — The directory containing the CA certificates and signing policy files of the CAs trusted by GSI. Typically this directory is `/etc/grid-security/certificates`. For more information see Grid security directory[14].

[11] http://www.globus.org/toolkit/docs/4.0/security/prewsaa/Pre_WS_AA_Public_Interfaces.html#prewsaa-env-credentials
[12] http://www.globus.org/toolkit/docs/4.0/security/prewsaa/Pre_WS_AA_Public_Interfaces.html#prewsaa-env-credentials
[13] http://www.globus.org/toolkit/docs/4.0/security/prewsaa/Pre_WS_AA_Public_Interfaces.html#prewsaa-env-credentials
[14] http://www.globus.org/toolkit/docs/4.0/security/prewsaa/Pre_WS_AA_Public_Interfaces.html#prewsaa-env-gridsecurity

U

user certificate	A EEC belonging to a user. When using GSI this certificate is typically stored in `$HOME/.globus/usercert.pem`. For more information on possible user certificate locations see Credentials[15].
user credentials	The combination of a user certificate and its corresponding private key.

[15] http://www.globus.org/toolkit/docs/4.0/security/prewsaa/Pre_WS_AA_Public_Interfaces.html#prewsaa-env-credentials

Chapter 2. GT4 WS AA Authorization Framework Release Notes

1. Component Overview

The Authorization Framework component provides a framework for container level authorization. It allows chains of authorization modules with well defined interfaces to be associated with various entities, e.g. services, in the container. It also provides multiple different authorization module implementations, ranging from support for gridmap based authorization to a module that uses the SAML protocol to query a external service for an authorization decision.

2. Feature Summary

Features new in GT 4.0

- A SAML callout authorization module enables outsourcing of authorization decisions to an authorization service (e.g. PERMIS).

Other Supported Features

- Authorization based on `grid-mapfile` and other access control lists.
- Ability to implement custom authorization modules.

Deprecated Features

- None

3. Changes Summary

3.1. Authorization when no authentication is required

We no longer invoke any authorization modules when a method is invoked and the service or resource does not impose any authentication requirements on said method.

3.2. Internationalization

The authorization framework code has been internationalized.

3.3. Typo X509_FORMAT constant in SAMLAuthorization-Constants

The typo in the above constants has been fixed. This implies that if the framework is run against an Authorization Service written using previous versions of the toolkit (that use this constant), the comparison will fail.

4. Bug Fixes

- Bug 2367:[1] No relative path for grid-mapfile in Security Descriptor.
- Bug 3187:[2] typo in X509_FORMAT constant in SAML Authorization constants.

5. Known Problems

None

6. Technology Dependencies

The WS Authentication and Authorization component depends on the following GT components:

- WS Authentication and Authorization Message-Level Security

The WS Authentication and Authorization component depends on the following 3rd party software:

- OpenSAML

7. Tested Platforms

Tested Platforms for WS Authorization Framework:

- Linux (Red Hat 7.3)
- Windows 2000
- Solaris 9

8. Backward Compatibility Summary

Protocol changes in the Authorization Framework since GT 3.2

- Addition of the SAML authorization callout

API changes since GT 3.2

- None

Exception changes since GT 3.2

- None

Schema changes since GT 3.2

- None

[1] http://bugzilla.globus.org/globus/show_bug.cgi?id=2367
[2] http://bugzilla.globus.org/globus/show_bug.cgi?id=3187

9. For More Information

Click here[3] for more information about this component.

[3] index.html

Chapter 3. GT 4.0.1 Incremental Release Notes: WS A&A Authorization Framework

1. Introduction

These release notes are for the incremental release 4.0.1. It includes a summary of changes since 4.0.0, bug fixes since 4.0.0 and any known problems that still exist at the time of the 4.0.1 release. This page is in addition to the top-level 4.0.1 release notes at http://www.globus.org/toolkit/releasenotes/4.0.1.

For release notes about 4.0 (including feature summary, technology dependencies, etc) go to the Authorization Framework 4.0 Release Notes[1].

2. Changes Summary

No changes have occurred for WS Authorization Framework.

3. Bug Fixes

The following bug was fixed for WS Authorization Framework:

- Bug 3528:[2] Action Operation Namespace in SAML Authorization Callout

4. Known Problems

No new problems are known to exist for WS Authorization Framework.

5. For More Information

Click here[3] for more information about this component.

[1] http://www.globus.org/toolkit/docs/4.0/security/authzframe/WS_AA_Authz_Release_Notes.html
[2] http://bugzilla.globus.org/globus/show_bug.cgi?id=3528
[3] index.html

Chapter 4. GT 4.0.2 Incremental Release Notes: WS A&A Authorization Framework

1. Introduction

These release notes are for the incremental release 4.0.2. It includes a summary of changes since 4.0.1, bug fixes since 4.0.1 and any known problems that still exist at the time of the 4.0.2 release. This page is in addition to the top-level 4.0.2 release notes at http://www.globus.org/toolkit/releasenotes/4.0.2.

For release notes about 4.0 (including feature summary, technology dependencies, etc) go to the Authorization Framework 4.0 Release Notes[1].

2. Changes Summary

No changes have occurred for WS Authorization Framework.

3. Bug Fixes

No bugs were fixed for WS Authorization Framework.

4. Known Problems

No new problems are known to exist for WS Authorization Framework.

5. For More Information

Click here[2] for more information about this component.

[1] http://www.globus.org/toolkit/docs/4.0/security/authzframe/WS_AA_Authz_Release_Notes.html
[2] index.html

Chapter 5. GT 4.0.3 Incremental Release Notes: WS A&A Authorization Framework

1. Introduction

These release notes are for the incremental release 4.0.3. It includes a summary of changes since 4.0.2, bug fixes since 4.0.2 and any known problems that still exist at the time of the 4.0.3 release. This page is in addition to the top-level 4.0.3 release notes at http://www.globus.org/toolkit/releasenotes/4.0.3.

For release notes about 4.0 (including feature summary, technology dependencies, etc) go to the Authorization Framework 4.0 Release Notes[1].

2. Changes Summary

No changes have occurred for WS Authorization Framework since GT 4.0.2.

3. Bug Fixes

No bugs have been fixed for WS Authorization Framework since GT 4.0.2.

4. Known Problems

No new problems are known to exist for WS Authorization Framework at the time of the GT 4.0.3 release.

5. For More Information

Click here[2] for more information about this component.

[1] http://www.globus.org/toolkit/docs/4.0/security/authzframe/WS_AA_Authz_Release_Notes.html
[2] index.html

Chapter 6. GT 4.0.4 Incremental Release Notes: WS A&A Authorization Framework

1. Introduction

These release notes are for the incremental release 4.0.4. It includes a summary of changes since 4.0.3, bug fixes since 4.0.3 and any known problems that still exist at the time of the 4.0.4 release. This page is in addition to the top-level 4.0.4 release notes at http://www.globus.org/toolkit/releasenotes/4.0.4.

For release notes about 4.0 (including feature summary, technology dependencies, etc) go to the Authorization Framework 4.0 Release Notes[1].

2. Changes Summary

No changes, other than bug fixes, have been made since GT 4.0.3 release

3. Bug Fixes

- Bug 4731:[2]close() not invoked on authorization chain.
- Fixed a bug in HostAuthorization class to deal with service certificates correctly.

4. Known Problems

No new problems are known to exist for WS Authorization Framework at the time of the GT 4.0.4 release.

5. For More Information

Click here[3] for more information about this component.

[1] http://www.globus.org/toolkit/docs/4.0/security/authzframe/WS_AA_Authz_Release_Notes.html
[2] http://bugzilla.globus.org/globus/show_bug.cgi?id=4731
[3] index.html

Chapter 7. GT4 WS AA Admin Guide

1. Introduction

This guide contains advanced configuration information for system administrators working with the Authorization Framework. It provides references to information on procedures typically performed by system administrators, including installing, configuring, deploying, and testing the installation.

Important

This information is in addition to the basic Globus Toolkit prerequisite, overview, installation, security configuration instructions in the GT 4.0 System Administrator's Guide[1]. Read through this guide before continuing!

This component determines the authorization enforced on the server and the client side. Admin configuration could include determining the container/service level authorization mechanism and setting up and managing authorization policy, e.g. entries in *grid map file* and so on.

The Security Descriptors[2] document describes how to configure an authorization mechanism, as well as the configuration required for each of the mechanisms that are distributed with GT 4.0.

2. Building and Installing

This component is built and installed as a part of Java WS Core[3].

3. Configuring

3.1. Configuration overview

The authorization framework can be configured at the resource, service or container level. Configuration settings are specified in security descriptors. The descriptors can be read from a file or can be created programmatically. Refer to Configuring Client Security Descriptor for more details.

Authorization configuration involves setting a chain of authorization schemes (also known as Policy Decision Points (PDPs)). When an authorization decision needs to be made the PDPs in the chain are evaluated in turn to arrive at a permit or deny decision. The combining rule for results from the individual PDPs is currently "deny overrides"; that is, all PDPs have to evaluate to permit for the chain to evaluate as permit.

3.2. Syntax of the interface

Refer to Section 4.2, “Configuring authorization mechanism ”.

4. Deploying

This component is deployed as a part of Java WS Core[4].

[1] ../../admin/docbook/
[2] security_descriptor.html
[3] ../../common/javawscore/admin-index.html#installing
[4] ../../common/javawscore/admin-index.html#deploying

5. Testing

To execute security tests ensure that Ant with JUnit is configured[5].

All the security tests require a valid credential. Refer to Security section of GT Administrator guide[6] for details on acquiring credentials.

The security tests are included in `$GLOBUS_LOCATION/lib/wsrf_test_unit.jar`. This jar contains tests for both the Java WS Core component and the WS Authentication and Authorization components contained in the Java WS Core package.

To execute the tests, pass the above jar file to the test script as described in the Java WS Core Developer's Guide[7]. To ensure that only security tests are run, set `-DsecurityTestsOnly=true`.

By default the tests require that the container and the tests are using the same credentials, i.e self authorization is done on secure calls.

The tests allow for another configuration in which the container can be configured with *host credentials* and the tests can be run with any credentials.

- Configure the container to use host credentials using the security descriptor as described in the container descriptor[8] section.
- Edit `$GLOBUS_LOCATION/etc/globus_wsrf_test_unit/server-config.wsdd`.
  - Comment out the configured descriptor in *SecurityTestService* and *AuthzCalloutTestService* that specifies self authorization.

    ```
    <!-- Does self authz by default -->

    <!-- parameter name="securityDescriptor"
        value="@config.dir@/security-config.xml"/ -->
    ```

  - Uncomment the configuration for identity authorization and set the value of the property *idenAuthz-identity* to the subject DN of the credentials used to run the tests.

    ```
    <!-- For use only when identity authz is used-->

    <parameter name="securityDescriptor"
        value="@config.dir@/identity-security-config.xml"/>

    <parameter name="idenAuthz-identity"
        value="Identity used by client"/>
    ```

- Start a secure and insecure standalone container.

  ```
  $ cd $GLOBUS_LOCATION
  ```

[5] ../../common/javawscore/admin-index.html#s-javawscore-admin-testing
[6] http://www.globus.org/toolkit/docs/4.0/admin/docbook/ch05.html
[7] http://www.globus.org/toolkit/docs/4.0/common/javawscore/developer-index.html#s-javawscore-developer-runningtests
[8] security_descriptor.html#s-authzfram-secdesc-descProgram

```
$ bin/globus-start-container -nosec
```

On another window,

```
$ cd $GLOBUS_LOCATION
$ bin/globus-start-container
```

- To run tests against external containers, secure and insecure, on localhost ports 8180 and 8181 respectively, the command would be:

```
ant -f share/globus_wsrf_test/runtests.xml runServer
       -Dtests.jar=$GLOBUS_LOCATION/lib/wsrf_test_unit.jar
       -DsecurityTestsOnly=true
       -Djunit.jvmarg=-Dsecurity.test.client.authz=host
       -Dsecurity.test.authz.identity="container/suject/DN"
       -Dsecurity.test.enc.cred="/server/public/key"
       -Dtest.server.url=http://127.0.0.1:8181/wsrf/services/
       -Dsecure.test.server.url=https://127.0.0.1:8180/wsrf/services/
```

- For example, if the container security descriptor has credentials configured as */etc/grid-security/containercert.pem* and */etc/grid-security/containerkey.pem* and the DN of the credential is */DC=org/OU=Services/CN=-testDN/CN=some.host.edu*, the command to run would be:

```
ant -f share/globus_wsrf_test/runtests.xml runServer
       -Dtests.jar=$GLOBUS_LOCATION/lib/wsrf_test_unit.jar
       -DsecurityTestsOnly=true
       -Djunit.jvmarg=-Dsecurity.test.client.authz=host
       -Dtest.server.url=http://127.0.0.1:8181/wsrf/services/
       -Dsecure.test.server.url=https://127.0.0.1:8180/wsrf/services/
      -Dsecurity.test.authz.identity="/DC=org/OU=Services/CN=-testDN/CN=some.host.edu"
    -Dsecurity.test.enc.cred=/etc/grid-security/containercert.pem
```

6. Security Considerations

6.1. Client side authorization

Client authorization of the server is done after the completion of the operation when GSI Secure Message authentication is used. If you require client authorization to be done prior, use GSI Secure Conversation or GSI *Transport security*.

6.2. Host authorization

During host authorization, the toolkit treats DNs "hostname-*.edu" as equivalent to "hostname.edu". This means that if a service was setup to do host authorization and hence accept the certificate "hostname.edu", it would also accept certificates with DNs "hostname-*.edu".

The feature is in place to allow a multi-homed host folowing a "hostname-interface" naming convention, to have a single host certificate. For example, host "grid.test.edu" would also accept likes of "grid-1.test.edu" or "grid-foo.test.edu".

Note

The wildcard character "*" matches only name of the hostand not domain components. This means that "hostname.edu" will not match "hostname-foo.sub.edu", but will match "host-foo.edu".

Note

If a host was set up to accept "hostname-1.edu", it will not accept any of "hostname-*.edu".

A bug[9] has been opened to see if this feature needs to be modfiied.

7. Troubleshooting

- Globus Toolkit Administrator Guide - Security Section[10]
- Java WS Core Troubleshooting guide[11]
- Message and Transport Level Security Troubleshooting guide[12]

[9] http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=2969
[10] http://www.globus.org/toolkit/docs/4.0/admin/docbook/ch06.html
[11] http://www.globus.org/toolkit/docs/4.0/common/javawscore/admin-index.html#s-javawscore-admin-troubleshooting
[12] http://www.globus.org/toolkit/docs/4.0/security/message/admin-index.html#s-message-admin-troubleshooting

Chapter 8. GT4 WS AA User's Guide

1. Introduction

Users who run clients can programmatically set up the authorization scheme to enforce on a per invocation basis. The properties and configuration information required depends on the configured authorization scheme. Section 5, “Semantics and syntax of domain-specific interface” describes the configuration steps in more detail.

2. Command line tools

There is no support for this type of interface.

3. Graphical user interfaces

This component has no graphical user interfaces.

4. Troubleshooting

4.1. Authorization failed

1. Using self authorization: Ensure that the client is running with the same credentials as the effective server-side credential (resource, service, container credential, in the order of occurrence).
2. Using host authorization:
   - Ensure that the effective server-side credential (resource, service, container credential, in the order of occurrence) is the host credential of the machine on which the service is running.
   - Ensure that the client is not using 127.0.0.1 as the host address to access the service, but the actual host name.
3. Using identity authorization: Ensure that the DN matches the server's DN exactly. If using the command line interface quotes might have to be placed around the DN string for spaces to be maintained.

4.2. No authorization with delegation fails

When using GSI Secure Conversation delegation of credentials cannot be done if no authorization of the server is done (that is, if client side authorization is set to none). Use any other form of authorization while delegating.

Alternatively, Delegation Service[1] can be used to delegate credentials in scenarios where delegated credentials are required but no authorization of the server is required.

Important

Delegating credentials without authorizing server is not recommended since a malicious server can obtain the client's credential.

[1] http://www.globus.org/toolkit/docs/4.0/security/delegation/

Chapter 9. GT4 WS AA Developer's Guide

1. Introduction

The authorization framework enforces the configured authorization policy on the service and client side. The framework allows developers to configure a chain of authorization mechanisms either programmatically or declaratively using security descriptors. It also allows for plugging in new authorization schemes (in addition to using those that are provided with the framework). Moreover, the framework allows for this configuration to be done at resource, service or container level, each taking precedence in the order specified and scoped as the name suggests.

2. Before you begin

2.1. Feature summary

Features new in GT 4.0

- A SAML callout authorization module enables outsourcing of authorization decisions to an authorization service (e.g. PERMIS).

Other Supported Features

- Authorization based on `grid-mapfile` and other access control lists.
- Ability to implement custom authorization modules.

Deprecated Features

- None

2.2. Tested platforms

Tested Platforms for WS Authorization Framework:

- Linux (Red Hat 7.3)
- Windows 2000
- Solaris 9

2.3. Backward compatibility summary

Protocol changes in the Authorization Framework since GT 3.2

- Addition of the SAML authorization callout

API changes since GT 3.2

- None

Exception changes since GT 3.2

- None

Schema changes since GT 3.2

- None

2.4. Technology dependencies

The WS Authentication and Authorization component depends on the following GT components:

- WS Authentication and Authorization Message-Level Security

The WS Authentication and Authorization component depends on the following 3rd party software:

- OpenSAML

2.5. Security considerations

2.5.1. Client side authorization

Client authorization of the server is done after the completion of the operation when GSI Secure Message authentication is used. If you require client authorization to be done prior, use GSI Secure Conversation or GSI *Transport security*.

2.5.2. Host authorization

During host authorization, the toolkit treats DNs "hostname-*.edu" as equivalent to "hostname.edu". This means that if a service was setup to do host authorization and hence accept the certificate "hostname.edu", it would also accept certificates with DNs "hostname-*.edu".

The feature is in place to allow a multi-homed host folowing a "hostname-interface" naming convention, to have a single host certificate. For example, host "grid.test.edu" would also accept likes of "grid-1.test.edu" or "grid-foo.test.edu".

Note

The wildcard character "*" matches only name of the hostand not domain components. This means that "hostname.edu" will not match "hostname-foo.sub.edu", but will match "host-foo.edu".

Note

If a host was set up to accept "hostname-1.edu", it will not accept any of "hostname-*.edu".

A bug[1] has been opened to see if this feature needs to be modfiied.

2.5.3. Client side authorization

Client authorization of the server is done after the completion of the operation when GSI Secure Message authentication is used. If you require client authorization to be done beforehand, use GSI Secure Conversation or GSI Transport security.

[1] http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=2969

3. Architecture and design overview

3.1. Server-side authorization

The authorization framework on the server side consists of an engine that evaluates a chain of configured authorization schemes, also known as Policy Decision Points (PDPs), to determine if the client making the invocation can access the resource/service. The chain can also be configured with Policy Information Points (PIPs), which can be used to glean information that would be useful in the decision making process.

The authorization engine is invoked as part of a handler chain, immediately after authentication of the invocation (`java:org.globus.wsrf.impl.security.authorization.AuthorizationHandler`). If no security mechanism is used for an operation, authorization is not done for that operation. The toolkit contains a set of PDPs that implement some authorization schemes or callout to some authorization services. Custom PDPs and PIPs can be written and plugged in as a part of the chain.

Any PDP has to implement the interface `org.globus.wsrf.security.authorization.PDP` and contain the logic to return a permit or deny based on information such as the subject DN, the service being accessed and the operation being performed. To manage configuration information, each PDP can be bootstrapped with an object implementing the `org.globus.wsrf.security.authorization.PDPConfig` interface. The interface has get and set methods which can be used to retrieve and set scoped parameters for the PDP.

PIPs have to implement the interface `org.globus.wsrf.security.authorization.PIP` with the functionality to collect attributes from the invocation context that are relevant to making the authorization decision.

A chain of PDPs and PIPs, with relevant configuration information, can be configured at resource, service or container level. If no chain is specified at resource level, service level is used; if nothing is specified at service level, the container level configuration is used. The engine evaluates each PDP and PIP in the order specified and a deny-override mechanism is used to render a decision. If one PDP returns a deny, the decision rendered is deny.

3.2. Client-side authorization

Client side authorization is done as a part of the authentication handler. GSI Secure Message authentication does client-side authorization only after the operation is completed. The other two authentication schemes supported, GSI Secure Conversation and GSI Transport, authorize the server prior to the operation. The toolkit supports self, gridmap, host and identity authorization on the client side. The authorization to be used is set programmatically on the Stub and the handler enforces it.

4. Public interface

The semantics and syntax of the APIs and WSDL for the component, along with descriptions of domain-specific structured interface data, can be found in the public interface guide[2].

5. Usage scenarios

Configuring authorization information can be done using a programmatic interface and is described in detail in these two sections of the Security Descriptor[3] document: Programmatic altering of descriptors[4] and Resource Security Descriptor[5].

[2] WS_AA_Authz_Public_Interfaces.html
[3] security_descriptor.html
[4] security_descriptor.html#s-authzfram-secdesc-descProgram
[5] security_descriptor.html#s-authzframe-secdesc-resDesc

If the authorization framework is set on either the service or container level and is using one of the schemes that are distributed with the toolkit, it is recommended that declarative configuration using security descriptor files be used.

6. Tutorials

- Article titled "Authorization processing for Globus Toolkit Java Web services"[6] published at IBM Developer Works provides an overview of the GT4 Web Services Authorization options, architecture and custom authorization approaches.

7. Debugging

Log output from the authorization framework is a useful tool for debugging issues. Logging in the code is based on the Jakarta Commons Logging[7] API and is described in more detail here[8]. As described in the above section, configuration files need to be edited to enable logging at different levels. For example, to see all logging for server side authorization, the following lines need to be added to container logging configuration file. To see client-side authorization framework logging, the same line needs to be added to *$GLOBUS_LOCATION/log4j.properties*.

```
log4j.category.org.globus.wsrf.impl.security.authorization=DEBUG
```

The authorization module uses Java CoG Kit[9] for some of the functionality. To turn on logging for that functionality, the following can be added to the relevant logging file, depending on whether it is the client or the server side logging.

```
log4j.category.org.globus.gsi.jaas=DEBUG
log4j.category.org.globus.gsi.gssapi=DEBUG
log4j.category.org.globus.security.gridmap=DEBUG
```

8. Troubleshooting

8.1. No grid map file

Check to ensure that a *grid map file* name is configured as shown here[10] and that the security descriptor with grid map file information is configured correctly.

8.2. Authorization failed. Peer is anonymous

If an anonymous client is used, only some authorization schemes can authorize them. Self authorization, identity authorization, gridmap authorization and host authorization will fail if anonymous clients are used. Ensure that an appropriate authorization scheme is used if anonymous clients are allowed.

[6] http://www-128.ibm.com/developerworks/grid/library/gr-gt4auth/
[7] http://jakarta.apache.org/commons/logging/
[8] ../../common/javawscore/developer-index.html#s-javawscore-developer-debugging
[9] http://www.globus.org/cog/java/
[10] security_descriptor.html#s-authzframe-secdesc-configGridmap

8.3. "org.globus.ogsa.security.authorization" property must be of "org.globus.ogsa.impl.security.authorization.Authorization" type

This error can occur on the client side if the authorization instance set on the stub does not implement interface `org.globus.ogsa.impl.security.authorization.Authorization`. A common error is importing authorization classes from `org.globus.gsi.gssapi.auth` rather than from `org.globus.ogsa.impl.security.authorization`. It is recommended that client security descriptors be used to set these properties, so the right classes and properties are setup by the framework.

9. Related Documentation

Associated standards for WS Authentication and Authorization Framework:

- Simple Assertion Markup Language[11]
- SAML Schema Protocol[12]

[11] http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security
[12] http://www.oasis-open.org/committees/download.php/3407/oasis-sstc-saml-schema-protocol-1.1.xsd

Chapter 10. GT4.0 WS AA Authz Factsheet

1. Brief component overview

The Authorization Framework component provides a framework for container level authorization. It allows chains of authorization modules with well defined interfaces to be associated with various entities, e.g. services, in the container. It also provides multiple different authorization module implementations, ranging from support for gridmap based authorization to a module that uses the SAML protocol to query a external service for an authorization decision.

2. Summary of features

Features new in GT 4.0

- A SAML callout authorization module enables outsourcing of authorization decisions to an authorization service (e.g. PERMIS).

Other Supported Features

- Authorization based on `grid-mapfile` and other access control lists.
- Ability to implement custom authorization modules.

Deprecated Features

- None

3. Usability summary

Usability improvements for WS Authentication and Authorization Framework:

- Allow for specifying the *grid map file* location relative to the current directory or *GLOBUS_LOCATION* in the security descriptor.
- Provide logging output in the form of warning messages when any authorization scheme in the authorization chain denies access.

4. Backward compatibility summary

Protocol changes in the Authorization Framework since GT 3.2

- Addition of the SAML authorization callout

API changes since GT 3.2

- None

Exception changes since GT 3.2

- None

Schema changes since GT 3.2

- None

5. Technology dependencies

The WS Authentication and Authorization component depends on the following GT components:

- WS Authentication and Authorization Message-Level Security

The WS Authentication and Authorization component depends on the following 3rd party software:

- OpenSAML

6. Tested platforms

Tested Platforms for WS Authorization Framework:

- Linux (Red Hat 7.3)
- Windows 2000
- Solaris 9

7. Associated standards

Associated standards for WS Authentication and Authorization Framework:

- Simple Assertion Markup Language[1]
- SAML Schema Protocol[2]

8. For More Information

Click here[3] for more information about this component.

[1] http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security
[2] http://www.oasis-open.org/committees/download.php/3407/oasis-sstc-saml-schema-protocol-1.1.xsd
[3] index.html

Chapter 11. GT4 WS AA Public Interfaces

1. Semantics and syntax of APIs

1.1. Programming Model Overview

Independent authorization settings can be configured on the server and client side. The security programming model on the server side is declarative and all configuration is done by setting a security descriptor. The descriptor can be a resource, service or container descriptor, depending on the required scope for the authorization. On the other hand, on the client side the configuration is done by setting required properties on the stub used to make the method invocation. The security properties, and hence the authorization settings, can be set directly as properties on the stub, or a client security descriptor that encapsulates the individual properties may be written and in turn passed to the framework via a property on the stub object.

1.2. Component API

- Stable API
  - org.globus.wsrf.security.Constants
  - org.globus.wsrf.security.SecureResource
  - org.globus.wsrf.security.SecurityManager
  - org.globus.wsrf.security.authorization.PDP
  - org.globus.wsrf.security.authorization.PIP
  - org.globus.wsrf.security.authorization.PDPConfig
  - org.globus.wsrf.security.authorization.PDPConstants
  - org.globus.wsrf.security.authorization.Interceptor
  - org.globus.wsrf.security.impl.authorization.Authorization
- Less stable API
  - org.globus.wsrf.impl.security.descriptor.ClientSecurityDescriptor
  - org.globus.wsrf.impl.security.descriptor.ServiceSecurityDescriptor
  - org.globus.wsrf.impl.security.descriptor.ResourceSecurityDescriptor

Documentation for these interfaces can be found here[1].

[1] http://www.globus.org/api/javadoc-4.0.0/globus_java_ws_core

2. Semantics and syntax of the WSDL

2.1. SAML Authorization Callout

The authorization framework as such does not have a WSDL interface. On the other hand one of the authorization scheme in the toolkit, a callout to an authorization service compliant with the specification published by the OGSA Authorization Working Group (OGSA-AuthZ)[2] requires a WSDL interface for the service. The callout makes a query on the configured authorization service, which returns an authorization decision.

2.2. Protocol overview

The authorization service takes a query as input and returns an authorization decision. The Security Assertion Markup Language (SAML)[3] is used for expressing the query and the decision. If any fault occurs, it is embedded as a part of the decision. The decision can be a permit, deny or indeterminate.

2.3. Operations

- `SAMLRequest`: Used to send queries to the authorization service, which after processing returns an authorization decision. All faults are embedded as part of the decision that is returned, i.e. no fault is declared at the WSDL level.
- `GetResourceProperty`: Gets the value of a specific resource property. This operation throws the following faults:
  - ResourceUnknownFault[4]
  - InvalidResourcePropertyQNameFault[5]
- `SetResourceProperties`: Sets the value for resource properties. This operation throws the following faults:
  - ResourceUnknownFault[6]
  - InvalidSetResourcePropertiesRequestContentFault[7]
  - UnableToModifyResourcePropertyFault[8]
  - InvalidResourcePropertyQNameFault[9]
  - SetResourcePropertyRequestFailedFault[10]
- `QueryResourceProperties`: Used for the querying of resource properties using a query expression. This operation throws the following faults:
  - ResourceUnknownFault[11]

[2] https://forge.gridforum.org/projects/ogsa-authz
[3] http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security
[4] #ResourceUnknownFault
[5] #InvalidResourcePropertyQNameFault
[6] #ResourceUnknownFault
[7] #InvalidSetResourcePropertiesRequestContentFault
[8] #UnableToModifyResourcePropertyFault
[9] #InvalidResourcePropertyQNameFault
[10] #SetResourcePropertyRequestFailedFault
[11] #ResourceUnknownFault

- InvalidResourcePropertyQNameFault[12]
- UnknownQueryExpressionDialectFault[13]
- InvalidQueryExpressionFault[14]
- QueryEvaluationErrorFault[15]

2.4. Resource properties

- `supportedPolicies`: Contains identifiers for any or all access control policies that the authorization service is capable of rendering decisions regarding.
- `supportsIndeterminate`: Indicates whether the authorization service may return an "indeterminate" authorization decision. If set to flase, only permit or deny is returned.
- `signatureCapable`: Indicates if the authorization service is capable of signing the decision returned. If not, only unsigned decisions are returned.

2.5. Faults

- ResourceUnknownFault[16]
- InvalidSetResourcePropertiesRequestContentFault[17]
- UnableToModifyResourcePropertyFault[18]
- InvalidResourcePropertyQNameFault[19]
- SetResourcePropertyRequestFailedFault[20]
- UnknownQueryExpressionDialectFault[21]
- InvalidQueryExpressionFault[22]
- QueryEvaluationErrorFault[23]

2.6. WSDL and Schema Definition

- OGSA-AuthZ Authorization Service WSDL [24]

[12] #InvalidResourcePropertyQNameFault
[13] #UnknownQueryExpressionDialectFault
[14] #InvalidQueryExpressionFault
[15] #QueryEvaluationErrorFault
[16] http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceProperties-1.2-draft-04.pdf
[17] http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceProperties-1.2-draft-04.pdf
[18] http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceProperties-1.2-draft-04.pdf
[19] http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceProperties-1.2-draft-04.pdf
[20] http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceProperties-1.2-draft-04.pdf
[21] http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceProperties-1.2-draft-04.pdf
[22] http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceProperties-1.2-draft-04.pdf
[23] http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceProperties-1.2-draft-04.pdf
[24] http://viewcvs.globus.org/viewcvs.cgi/wsrf/schema/core/security/authorization/authz_port_type.wsdl?rev=1.9&only_with_tag=globus_4_0_0&content-type=text/vnd.viewcvs-markup

3. Command-line tools

There is no support for this type of interface.

4. Overview of Graphical User Interfaces

This component has no graphical user interfaces.

5. Semantics and syntax of domain-specific interface

5.1. Interface introduction

On the server side the authorization configuration is configured using security descriptors, either programmatically or using files. Refer to Section 3.5, “Configuring authorization mechanisms” and Section 3.6, “Writing a custom authorization mechanism” for detailed documentation.

On the client side the authorization configuration is set on the stub instance used for method invocation. Properties can be set directly on the stub or set using a client security descriptor. The next section discusses setting property directly on stub and writing custom authorization schemes, while Section 4.2, “Configuring authorization mechanism ” describes using clients side security descriptors to configure a distributed authorization schema.

5.2. Syntax of the interface

5.2.1. Using distributed client-side authorization schemes

The following client side authorization schemes are distributed as a part of the toolkit: no authorization, self authorization, host authorization and identity authorization.

- If setting *properties directly on the stub*, there are two properties that maybe used, based on that authentication scheme used.
  - If GSI Secure Transport or GSI Secure Conversation is used, the `org.globus.axis.gsi.GSIConstants.GSI_AUTHORIZATION` property must be set on the stub. The value of this property must be an instance of an object that extends from `org.globus.gsi.gssapi.auth.GSSAuthorization`. All distributed authorization schemes have implementation in `org.globus.gsi.gssapi.auth` package.
  - For all other authentication schemes, the `org.globus.wsrf.impl.security.Constants.AUTHORIZATION` property must be set on the stub. The value of this property must be an instance of an object that implements the `org.globus.wsrf.impl.security.authorization.Authorization` interface.
  - Example:

```
// Create endpoint reference EndpointReferenceType
endpoint = new EndpointReferenceType();
// Set address of service
String counterAddr =
   "http://localhost:8080/wsrf/services/CounterService";
   // Get handle to stub object
```

```
CounterPortType port =
   locator.getCounterPortTypePort(endpoint);
   // set client authorization to self
   ((Stub)port)._setProperty(Constants.AUTHORIZATION, SelfAuthorization.getI
```

- Setting properties using the client descriptor is described in Configuring Client Security Descriptor and Section 4.2, "Configuring authorization mechanism "

5.2.2. Writing custom client-side authorization scheme

Other than the distributed client authorization scheme, custom client-side authorization schemes can be written and can be set as value for appropriate property on the stub.

Note

Security descriptors cannot be used to configure custom authorization schemes on client side.

- If the authentication scheme to be used is *GSI Secure Transport* or *GSI Secure Conversation*, the custom authorization scheme should extend from extends from `org.globus.gsi.gssapi.auth.GSSAuthorization`.

```
public class TestAuthorization extends GSSAuthorization {

  // Provide some way to instantiate this class. Can use constructor
  // with arguments to pass in parameters.
   public TestAuthorization() {

   }

   public GSSName getExpectedName(GSSCredential cred, String host)
      throws GSSException {

      // Return the expected GSSName of the remote entity.
   }


    public void authorize(GSSContext context, String host)
throws AuthorizationException {

       // Perform the authorization steps.
       // context.getSrcName() provides the local GSSName
       // context.getTargName() provides the remote GSSName

       // if authorization fails, throw AuthorizationException
   }
}
```

The following describes the steps done for client side authorization during context establishment:

- Prior to initialization of context establishment the relevant handler (HTTPSSender in case of GSI Secure Transport or SecContextHandler in case of GSI Secure Conversation), invokes *getExpectedName* on the instance of GSSAuthorization set on the Stub.
- During context establishment, if the expected target name from previous step is not null, it is compared with the remote peer's GSSName. If it is not a match, context establishment is abandoned and an error is thrown.

  If the expected target name is null, then a match is not done, unless the option of delegation is used. That is, if GSI Secure Conversation with delegation is used, then the expected target name cannot be null and must match the remote peer's identity.
- Once the context has been established, the *authorize* method is invoked.

Note

Client authorization is done prior to invocation.

To configure the custom authorization scheme:

```
((Stub)port)._setProperty(GSIConstants.GSI_AUTHORIZATION, new TestAuthorization())
```

Various authorization scheme implementations in package `org.globus.gsi.gssapi.auth` would serve as examples. View CVS Link[25]

- For all authentication schemes other than GSI Secure Transport, the `org.globus.wsrf.impl.security.Constants.AUTHORIZATION` property must be set on the stub. The value of this property must be an instance of an object that implements the `org.globus.wsrf.impl.security.authorization.Authorization` interface.

```
public class TestAuthorization implements Authorization {

  // Provide some way to instantiate this class. Can use constructor
  // with arguments to pass in parameters.
   public TestAuthorization() {

   }

   public GSSName getName(MessageContext ctx)
      throws GSSException {

      // Return the expected GSSName of the remote entity.
   }


    void authorize(Subject peerSubject, MessageContext context)
                          throws AuthorizationException {

       // Perform the authorization steps.
       // peerSubject provides the remote Subject
       // Use SecurityManager API to get local Subject
```

[25] http://viewcvs.globus.org/viewcvs.cgi/jglobus/src/org/globus/gsi/gssapi/auth/?root=Java+COG&pathrev=globus_4_0_branch

```
            // if authorization fails, throw AuthorizationException
        }
    }
```

The following describes the steps done for client side authorization:

- The client side handler WSSecurityClientHandler, invokes *authorize* method on the authorization instance.

Note

Client authorization is done after the invocation.

To configure the custom authorization scheme:

```
((Stub)port)._setProperty(Constants.AUTHORIZATION, new TestAuthorization());
```

Various authorization scheme implementations in package `org.globus.wsrf.impl.security.authorization` would serve as examples. ViewCVS Link[26].

5.2.3. Default client-side authorization

If no authorization mechanism has been specified, Host authorization is used.

6. Configuration interface

6.1. Configuration overview

The authorization framework can be configured at the resource, service or container level. Configuration settings are specified in security descriptors. The descriptors can be read from a file or can be created programmatically. Refer to Configuring Client Security Descriptor for more details.

Authorization configuration involves setting a chain of authorization schemes (also known as Policy Decision Points (PDPs)). When an authorization decision needs to be made the PDPs in the chain are evaluated in turn to arrive at a permit or deny decision. The combining rule for results from the individual PDPs is currently "deny overrides"; that is, all PDPs have to evaluate to permit for the chain to evaluate as permit.

6.2. Syntax of the interface

Refer to Section 4.2, “Configuring authorization mechanism ”.

7. Environment variable interface

There is no support for this type of interface.

[26] http://viewcvs.globus.org/viewcvs.cgi/wsrf/java/core/source/src/org/globus/wsrf/impl/security/authorization/?pathrev=globus_4_0_branch

Chapter 12. GT4 WS AA Authz Quality Report

1. Test coverage reports

- Clover report[1]

2. Code analysis reports

- PMD report[2]
- FindBugs report[3]

3. Outstanding bugs

This component has no outstanding bugs.

4. Bug Fixes

- Bug 2367:[4] No relative path for grid-mapfile in Security Descriptor.
- Bug 3187:[5] typo in X509_FORMAT constant in SAML Authorization constants.

5. Performance reports

Secure access of WSRF and WSN operations using GSI Transport and GSI Secure Message have been measured. Test reports are here[6].

[1] http://www.mcs.anl.gov/~gawor/javawscore/HEAD/clover/clover-reports-html/
[2] http://www.mcs.anl.gov/~gawor/javawscore/HEAD/pmd/pmd_report.html
[3] http://www.mcs.anl.gov/~gawor/javawscore/HEAD/findbugs/findbugs_report.html
[4] http://bugzilla.globus.org/globus/show_bug.cgi?id=2367
[5] http://bugzilla.globus.org/globus/show_bug.cgi?id=3187
[6] ../../common/javawscore/java_ws_core_wsrf_perf.html

Chapter 13. Security Descriptors

1. Introduction

Security descriptors contain various security properties like credentials, the *grid map file* location, required authentication and authorization mechanisms and so on. There are four types of security descriptors in the code base for setting container, service, resource and client security properties:

container security descriptor	determines the container level security requirement that needs to be enforced.
service security descriptor	determines the service level security requirement that needs to be enforced.
resource security descriptor	determines the resource level security requirement that needs to be enforced.
client security descriptor	determines the security properties that need to be used for a particular invocation.

Each of these is represented as an object and can be altered programmatically. Service and container security descriptors can be configured as XML files in the deployment descriptor as shown below. Resource security descriptors can only be created dynamically, either programmatically or from a descriptor file. Client security descriptor can be configured as a XML file and set as property on Stub. If the security descriptor file is altered at runtime, it will *not* be reloaded

2. Configuring security descriptors

The following shows how the service and container security descriptor files are configured.

The container security descriptor needs to be set in the core server-config.wsdd. That file is in wsrf/java/core/source/deploy-server.wsdd if editing source, prior to deploy or $GLOBUS_LOCATION/etc/globus_wsrf_core/server-config.wsdd in a binary install. The parameter element to be added is:

```
<globalConfiguration>
   ...
   <parameter name="containerSecDesc"
              value="etc/container-security-config.xml">
   ...
<globalConfiguration>
```

A sample container security descriptor file, global_security_descriptorn.xml is shipped as a part of the toolkit.

Service security descriptor should be configured as follows:

```
<service name="MyDummyService" provider="Handler" style="document">
   ...
   <parameter name="securityDescriptor" value="org/globus/wsrf/impl/security/descriptor/se
   ...
</service>
```

If the security descriptor is configured to be read from a file, it is loaded as follows:

1. As a file if an absolute file path is specified.

2. As a resource (can be included as part of jar file).

3. As a file, assuming that the specified path is relative to the installation root, typically pointed to by environment variable `GLOBUS_LOCATION`.

The security descriptor files need to comply with the service security descriptor schema[1] or container security descriptor schema[2] as appropriate. The resource security descriptor file uses the same schema as the service security descriptor. In all cases, the security descriptor is contained within the `<securityConfig xmlns="http://www.globus.org">` element.

2.1. Configuring Client Security Descriptor

Client security descriptors are configured as shown below:

```
// Client security descriptor file
String CLIENT_DESC = "org/globus/wsrf/samples/counter/client/client-security-config.xml";
//Set descriptor on Stub
((Stub)port)._setProperty(Constants.CLIENT_DESCRIPTOR_FILE, CLIENT_DESC);
```

The client security descriptor files need to comply with the client security descriptor schema[3].

3. Writing server-side security descriptor files

The next few sections deal with writing server side security descriptor files—that is, container, service and resource descriptor files to set various properties. Please note that not all parameters are applicable for all three types of descriptors and are appropriately annotated in the relevant sections. The few parameters relevant only for the container security descriptor are described in Section 5[4].

3.1. Configuring credentials

The container and each service can each be configured with a separate set of credentials. The credentials can be set using either: a) the path to a proxy file, or b) the path to a certificate and key file. If the configured credential file is modified/updated at runtime, the credentials will be automatically reloaded. The credentials can be configured by adding one of the following blocks to the container or service security descriptor.

Example for option (a):

```
<securityConfig xmlns="http://www.globus.org">
   ...
   <credential>
      <key-file value="keyFile"/>
      <cert-file value="certFile"/>
   </credential>
```

[1] service_security_descriptor.xsd
[2] container_security_descriptor.xsd
[3] client_security_descriptor.xsd
[4] #s-authzfram-secdesc-descProgram

```
   ...
</securityConfig>
```

Example for option (b):

```
<securityConfig xmlns="http://www.globus.org">
   ...
   <proxy-file value="proxyFile"/>
   ...
</securityConfig>
```

The framework will look for credentials in the following order:

1. Resource credentials
2. *Service credentials*
3. Container credentials
4. Default credentials. This uses the underlying security library to acquire the credentials. It will typically result in finding the *proxy certificate* of the user that is running the container.

3.2. Configuring grid map files

The container and each service can be configured with a separate grid map file in each of their security descriptors as shown below:

```
<securityConfig xmlns="http://www.globus.org">
   ...
   <gridmap value="gridMapFile"/>
   ...
</securityConfig>
```

The framework will first look for a gridmap configured by the resource, then by the service and then by the container. For services configured to perform grid map file authorization, the grid map file can be updated dynamically using the SecurityManager API. Also, if the gridmap file changes at runtime it will be automatically reloaded.

3.3. Configuring authentication methods

This can only be done at service or resource level.

The authentication methods a service requires are specified using the `<auth-method>` element. The authentication methods can also be configured on a per method basis. This is done by specifying the `<auth-method>` element within a `<method name="operation name">` element. The value of the *name* attribute can be set to either the operation name (preferred) or the operation name with a given namespace.

Currently, the following authentication methods are supported:

Table 13.1. Authentication methods

`<none/>`	Indicates that no authentication is required. This method *cannot* be specified with any other authentication method.
`<GSISecureMessage/>`	Indicates the GSI Secure Message authentication method. The `<protection-level>` sub element can be used to specify a protection level that must be applied to the message:
`<integrity/>`	Indicates that the message must be integrity protected (signed).
`<privacy/>`	Indicates that the message must be privacy protected (encrypted and signed).
`<GSISecureConversation/>`	Indicates the GSI Secure Conversation authentication method (with integrity or privacy protection). The `<protection-level>` sub element can be used to indicate a specific protection level that must be applied to the message:
`<integrity/>`	Indicates that the message must be integrity protected (signed).
`<privacy/>`	Indicates that the message must be privacy protected (signed and encrypted).
`<GSITransport/>`	Indicates the GSI Secure Transport authentication method. The `<protection-level>` sub element can be used to specify a specific protection level that must be applied to the message:
`<integrity/>`	Indicates that the message must be integrity protected (signed). This is always true in this mechanism.
`<privacy/>`	Indicates that the message must be privacy protected (encrypted and signed).

Notes:

- Multiple authentication methods can be specified under the `<auth-method>` element (except for the `<none/>` method, see above). As long as one of the specified authentication methods is used, access to the service is allowed.
- If *no* `<protection-level>` sub element is specified, then all protection levels are available to clients. However, if the `<protection-level>` sub element *is* specified, then the service will only accept the protection levels listed under said element.
- The `org.globus.wsrf.impl.security.authentication.SecurityPolicyHandler` handler *must* be installed properly in order for this to work. This handler is installed by default.
- If a security descriptor is *not* specified, authentication method enforcement is *not* performed.

Example:

```
<securityConfig xmlns="http://www.globus.org">

   <method name="findServiceData">
      <auth-method>
         <none/>
      </auth-method>
   </method>

   <method name="destroy">
      <auth-method>
```

```
         <GSISecureMessage/>
         <GSISecureConversation>
            <protection-level>
               <integrity/>
            </protection-level>
         </GSISecureConversation>
      </auth-method>
   </method>

   <!-- default auth-method for any other method -->
   <auth-method>
      <GSISecureConversation/>
   </auth-method>
</securityConfig>
```

In the above example:

- the `findServiceData()` operation does not require any authentication.
- the `destroy()` operation requires either *GSI Secure Message* authentication with either level of protection or *GSI Secure Conversation* authentication with integrity protection.
- all other operations must be authenticated with *GSI Secure Conversation* with either level of protection.

3.4. Configuring the run-as mode

The `<run-as>` element is used to configure the JAAS run-as identity under which the service method will be executed. The run-as identity can be configured on a per method basis. Currently, the following run-as identities are supported:

Table 13.2. Run-as methods

`<caller-identity/>`	The service method will be run with the security identity of the client. The caller Subject will contain the following:
	• If using *GSI Secure Message*: a GlobusPrincipal (the identity of the signer) is added to the principal set of the caller-identity Subject. Also, the signer's certificate chain is added to the public credentials set of the Subject object.
	• If using *GSI Secure Conversation*: a GlobusPrincipal (the identity of the initiator) is added to the principal set of the Subject.
	  • If client authentication was performed, the client's certificate chain will be added to the public credentials set of the Subject object.
	  • Also, if delegation was performed, the delegated credential is added to the private credential set of the Subject object.
	• If grid map file authorization was performed, a UserNamePrincipal is added to the principal set of the Subject object.

`<system-identity/>`	The service method will be run with the security identity of the container.
`<service-identity/>`	The service method will be run with the security identity of the service itself (if the service has one, otherwise the container identity will be used).
`<resource-identity/>`	The service method will be run with the security identity of the resource. If no resource is specified or if the resource does not have a configured subject, credentials in this order of occurrence will be used: service credential, container credential.

Notes:

- *resource-identity* is the default setting.
- The `org.globus.wsrf.impl.security.authentication.SecurityPolicyHandler` handler *must* be installed properly in order for this to work. It is installed by default.
- If the security descriptor is *not* specified, then the run-as identity is not set and there will be no JAAS subject associated with the execution of the operation. This means that any method calls that require credentials and that are invoked by the service method itself will fail.

Example:

```
<securityConfig xmlns="http://www.globus.org">
   <method name="add">
      <run-as>
         <caller-identity/>
      </run-as>
   </method>

   <method name="subtract">
      <run-as>
         <system-identity/>
      </run-as>
   </method>
```

```
   <!-- default run-as for any other method -->
   <run-as>
      <service-identity/>
   </run-as>
</securityConfig>
```

In the above example:

- the `add()` operation will be run with the caller's identity.
- the `subtract()` call will be run with the system identity.
- all other operations will be run with the service identity (if the service has one set).

3.5. Configuring authorization mechanisms

The container and each service can be configured with a chain of authorization mechanisms (also known as Policy Decision Points (PDPs)), using the *authz* element. Each PDP name is scoped and the format is *prefix:FQDN of the PDP*. For example, *self:org.globus.wsrf.impl.security.authorization.SelfAuthorization*. The prefix is used to allow multiple instances of the same PDP to exist in the same PDP chain. The authorization is deemed to be a permit if each of the authorization mechanisms in the chain returns a permit.

Example:

```
<securityConfig xmlns="http://www.globus.org">
   ...
   <authz value="foo1:org.foo.authzMechanism bar1:org.bar.barMechanism"/>
   ...
<securityConfig/>
```

Each PDP is instantiated with some configuration information that can be used to get any further information that the PDP may need to make authorization decisions. If the authorization chain is configured at the container level, then the parameters are picked up from the global configuration section of the container deployment descriptor. If the authorization chain is configured at the service level, the PDPs will pick up parameters from the relevant service section of the deployment descriptor. Resource level configuration has to be done programmatically and is described here[5]. In all three cases, the prefix specified in the authorization chain configuration is used to get the right property. For example, all properties for *foo1:org.foo.authzMechanism* are picked up from properties that have been scoped with the prefix *foo1-*.

The following PDPs are a part of the toolkit and are configured as shown. The framework maps and plugs in the scoped name of the PDP at the time of authorization.

[5] #s-authzframe-secdesc-resDesc

Table 13.3. Builtin PDPs

`none`	No authorization is performed.
`self`	• PDP Name: `selfAuthz:org.globus.wsrf.impl.security.authorization.SelfAuthori` • This scheme does not require any additional configuration information. • Only clients that present the same identity as the identity in the current JAAS subject associated with the servi The current JAAS subject is determined by the value of the run-as element in the service security descriptor (s
`gridmap`	• PDP Name: `grimapAuthz:org.globus.wsrf.impl.security.authorization.GridMapAu` • A grid map file must be configured as described in Section 3.2, "Configuring grid map files". • Grid map file authorization is performed, i.e. a mapping must exist for the client identity in the configured *grid*
`identity`	• PDP Name: `idenAuthz:org.globus.wsrf.impl.security.authorization.IdentityAut` • The property `idenAuthz-identity` set to the expected identity must be configured in the service or contai authorization respectively. • The client identity must match the value of this property.
`host`	• PDP Name: `hostAuthz:org.globus.wsrf.impl.security.authorization.HostAuthori` • The property `hostAuthz-url` set to the expected host name must be configured in the service or container authorization respectively. • Host based authorization is done and should match the expected host set in the property.
`samlCallout`	• PDP Name: `samlAuthz:org.globus.wsrf.impl.security.authorization.SAMLAuthori` • This scheme calls out to a configured OGSA-AuthZ[7] compliant authorization service. • When this PDP is specified via the `samlCallout` alias or the `org.globus.wsrf.impl.security.a` prefix `samlAuthz` is used when acquiring configuration settings. • The SAML authorization callout PDP can be configured by specifying parameters in the service entry in the dep configuration properties are described in Table 13.4, "SAML Callout PDP Parameters".
`userName`	• PDP Name: `userNameAuthz:org.globus.wsrf.impl.security.authorization.Usernam` • This scheme does not require any additional configuration information. • This uses the configured JAAS Login Module[8] to authorize the user based on username and password. The PDP and password information to the Login module.

[6] #s-authzframe-secdesc-configRunas
[7] https://forge.gridforum.org/projects/ogsa-authz
[8] http://java.sun.com/products/jaas/

someScope:org.globus.wsrf.impl.security.authorization.LocalConfigPDP	• PDP Name: someScope:org.globus.wsrf.impl.security.authorization.LocalConfig • The property authzConfigFile should be set to a file containing mappings between users and their allowed oper are separated by semicolon (;). The file may be modified without restarting the hosting environemnt. Example: /O\=Grid/O\=Globus/OU\=Sample\Org/CN\=AdminUser={http://www.globus.org/cou /O\=Grid/O\=Globus/OU\=Sample\Org/CN\=User={http://www.globus.org/counter} Note that white spaces and equal signs (=) need to be escaped with backslash (\) • This parses the configure file to determine if a specific user is allowed to access an operation.

Table 13.4. SAML Callout PDP Parameters

<prefix>-authzService	The URL of the authorization service.
<prefix>-authzServiceIdentity	The identity to use for authorizing the authorization service. If no identity was specified then the service is authorized using the identity associated with the entity performing the callout (self-authorization).
<prefix>-authzServiceCertificateFile	A filename identifying the certificate to use when encrypting messages and verifying responses signed at the SAML level. This is only required if using GSI Secure Message with privacy protection or if the user requested SAML signing.
<prefix>-authzServiceCertificate	This parameter is equivalent to the above, but can only be used programmatically. It must be set to a value of type java.security.cert.X509Certificate.
<prefix>-securityMechanism	The security mechanism to use. Recognized values are none, msg (GSI-Secure Message) and conv (GSI-Secure Conversation). *Transport security* may be indicated by specifying a HTTPS URL in the <prefix>-authzService property. This property defaults to transport security if indicated by the URL and GSI-Secure Message otherwise.
<prefix>-protectionLevel	The protection level to use. Recognized values are sig (integrity protection) and enc (privacy and integrity protection). Defaults to sig.
<prefix>-samlAuthzReqSigned	Determines if the request is internally signed or not. SAML requests can include a signature in the request and response documents. This is separate from any security mechanism applied at either the SOAP or transport level. Recognized values are true and false. Defaults to false.
<prefix>-samlAuthzSimpleDecision	Determines whether to request a simple decision statement or not. More information on this setting can be found here[9]. Recognized values are true and false. Defaults to true.

Other than these, any custom authorization scheme could be configured with its own configuration information. Refer to Section 3.6, "Writing a custom authorization mechanism", for details on writing a custom authorization mechanism.

[9] https://forge.gridforum.org/projects/ogsa-authz

3.6. Writing a custom authorization mechanism

The authorization handler can be configured to call out to a custom authorization class. The class must implement the interface `org.globus.wsrf.security.PDP`.

Example:

```
package org.foobar;

import ....;

public class FooPDP implements PDP
{
    private Principal authorizedIdentity;

    /* Not used by the current code */
    public String[] getPolicyNames() {
        return new String[0];
    }

    /* Not used by the current code */
    public Node getPolicy(Node query)
        throws InvalidPolicyException {
        return null;
    }

    /* Not used by the current code */
    public Node setPolicy(Node policy)
        throws InvalidPolicyException {
        return null;
    }

    public boolean isPermitted(Subject peerSubject,
                               MessageContext context,
                               QName operation)
        throws AuthorizationException {

        if (peerSubject == null) {
            return false;
        }

        Set peerPrincipals = peerSubject.getPrincipals();

        if ((peerPrincipals == null) || peerPrincipals.isEmpty()) {
            return false;
        }

        /* Check if the peer identity and the authorized
         * identity match
         */

        return peerPrincipals.contains(this.authorizedIdentity);
    }
```

```
    public void initialize(PDPConfig config,
                           String name,
                           String id)
        throws InitializeException {

        /* Read the initialization information from the service
         * specific WSDD parameter <name>-authorizedIdentity
         */

        this. authorizedIdentity =
            new GlobusPrincipal((String) config.getProperty(
                name, "authorizedIdentity"));
    }

    public void close() throws CloseException {
        this. authorizedIdentity = null;
    }
}
```

To use the above PDP one would configure a service security descriptor with the following authorization settings:

```
<securityConfig xmlns="http://www.globus.org">
   ...
   <authz value="foo1:org.foobar.FooPDP"/>
   ...
<securityConfig/>
```

This security descriptor (identified as `/.../foo-pdp-security-config.xml` below) can then be used by a service. The association is created by adding a couple of parameters to the service's WSDD entry:

```
...
<service name="MyDummyService"
             provider="Handler"
             style="document">
   ...
   <parameter name="securityDescriptor"
                    value="/.../foo-pdp-security-config.xml"/>
   <parameter name="foo1-authorizedIdentity"
                    value="/DC=org/DC=doe/OU=People/CN=John D"/>
   ...
</service>
```

Note that the parameter *`foo1-authorizedIdentity`* in the above configures the identity the PDP uses for authorizing incoming requests. The parameter name is derived by composing the prefix (*`foo1`*) used when specifying the PDP in the security descriptor with the property (*`authorizedIdentity`*) used in the PDP code.

4. Writing client side security descriptors

4.1. Configuring credentials

Client side credentials are configured similar to server side credentials as described in Section 3.1, “Configuring credentials”.

4.2. Configuring authorization mechanism

The <authz> element is used to determine the mechanism to use to authorize the server that is being contacted. The following values are currently supported:

`none`	No authorization is done.
`self`	Self authorization is done, i.e the server should be running with the same credentials as the client.
`host`	Host authorization is done, i.e the server should be running with credentials that have the host name it is running on embedded in it.
any other string	Identity authorization is done using the value as the identity, i.e the server should be running with identity specified as value.

The following sample configures self authorization:

```
<securityConfig xmlns="http://www.globus.org">
   ...
   <authz value="self"/>
   ...
</securityConfig>
```

Note

Custom client authorization schemes can be written and plugged in. But security descriptors cannot be used to configure such authorization schems. Refer to Section 5, “Semantics and syntax of domain-specific interface” for information on writing custom client-side authorization scheme.

4.3. Configuring GSI Secure Conversation

The client can be configured to do GSI Secure Conversation using the element <GSISecureConversation>. The following subelements can be used to set various properties

`<integrity>`	Sets protection level to signature.
`<privacy>`	Sets protection level to encryption (signature is also done).
`<anonymous>`	Server is accessed as anonymous.
`<delegation value="`*type of delegation*`">`	Determines the type of delegation to be done. The value can be set to `full` or `limited`. If the *delegation* element is not used, no delegation is done.

The following sample sets GSI Secure Conversation with privacy and full delegation:

```
<securityConfig xmlns="http://www.globus.org">
   ...
   <GSISecureConversation>
       <privacy/>
       <delegation value="full"/>
   </GSISecureConversation>
   ...
</securityConfig>
```

4.4. Configuring GSI Secure Message

The client can be configured to do GSI Secure Message using the element <GSISecureMessage>. The following subelements can be used to set various properties:

`<integrity>`	Sets protection level to signature
`<privacy>`	Sets protection level to encryption (signature is also done)
`<peer-credential value="`*path to file with credentials to encrypt with*`">`	Sets the path to the file containing the credential to use if privacy protection is chosen.

The following sample sets GSI Secure Message with integrity:

```
<securityConfig xmlns="http://www.globus.org">
   ...
   <GSISecureMessage>
       <integrity/>
   </GSISecureMessage>
   ...
</securityConfig>
```

5. Programmatic altering of security descriptors

The security descriptor (container, security and resource) can be created and altered programmatically (as opposed to writing a security descriptor file). For the service and container descriptor, we recommend writing a security descriptor file so that the security properties are initialized at start up.

Table 13.5. Descriptor classes

Container Security Descriptor	This is represented by `org.globus.wsrf.impl.security.descriptor.ContainerSecurityDescriptor`. If a container security descriptor file is configured as described in Section 2, "Configuring security descriptors ", then an object is created and stored. To alter the values, use the API provided in `org.globus.wsrf.impl.security.descriptor.ContainerSecurityConfig`.
Service Security Descriptor	This is represented by `org.globus.wsrf.impl.security.descriptor.ServiceSecurityDescriptor`. If a service security descriptor file is configured as described in Section 2, "Configuring security descriptors ", then an object is created and stored. To alter the values, use the API provided in `org.globus.wsrf.impl.security.descriptor.ServiceSecurityConfig`.
Resource Security Descriptor	This is represented by `org.globus.wsrf.impl.security.descriptor.ResourceSecurityDescriptor`. To initialize the descriptor, i.e. load credentials and gridmap, use the API in `org.globus.wsrf.impl.security.descriptor.ResourceSecurityConfig`. Refer to the description of resource security descriptors in Section 6, "Resource security descriptors" for more details.
Client Security Descriptor	This is represented by `org.globus.wsrf.impl.security.descriptor.ClientSecurityDescriptor`. To initialize the descriptor, use the API in `org.globus.wsrf.impl.security.descriptor.ClientSecurityConfig`.

6. Resource security descriptors

Resource level security can be set up using a resource security descriptor. A resource security descriptor overrides any service or container level security settings. To make a resource secure, it needs to implement `org.globus.wsrf.impl.security.SecureResource`. This interface has a method that returns an instance of `org.globus.wsrf.impl.security.descriptor.ResourceSecurityDescriptor`. If `null` is returned, it is assumed that no security is set on the resource.

Secure resources must implement org.globus.wsrf.security.SecureResource interface.

1. A ResourceSecurityDescriptor object can be created and initialized in the resource's constructor. The object should be returned as a part of the getSecurityDescriptor method.

```
public MyDummyResource implements SecureResource {

   private ResourceSecurityDescriptor desc = null;

   public MyDummyResource() throws Exception {


        this.desc = new ResourceSecurityDescriptor();
```

```
            // set security properties on the above object using get/set methods
            // in the API
        }

        public ResourceSecurityDescriptor getSecurityDescriptor() {
            return this.desc;
        }
    }
```

2. A ResourceSecurityDescriptor object can be created similar to above, but initialized from a file and set in the constructor.

```
    public MyDummyResource implements SecureResource {

       private ResourceSecurityDescriptor desc = null;

           ResourceSecurityConfig securityConfig =
               new ResourceSecurityConfig("/path/to/security/file");
           try {
             securityConfig.init();
           } catch (ConfigException exp) {
              // handle exception
           }
           this.desc = securityConfig.getSecurityDescriptor();
        }

        public ResourceSecurityDescriptor getSecurityDescriptor() {
            return this.desc;
        }
    }
```

The resource security descriptor is identical to the service security descriptor and exposes an API to set and get all properties that are described in Section 3, “Writing server-side security descriptor files ”. A resource security descriptor object can also be created by reading settings from a descriptor file. The file needs to be written as described in Section 3, “Writing server-side security descriptor files ”.

Examples:

The following code snippet creates a resource descriptor object directly:

```
ResourceSecurityDescriptor desc = new ResourceSecurityDescriptor();
desc.setRejectLimitedProxy("true");
```

The following code snippet creates a resource descriptor object from a file:

```
ResourceSecurityConfig config = new ResourceSecurityConfig("resDescFileName");
config.init();
ResourceSecurityDescriptor desc = config.getSecurityDescriptor();
```

There are two attributes of the security descriptor, *credentials* and *gridmap*, that can be specified as objects (`javax.security.auth.Subject` and `org.globus.security.gridmap.GridMap`, respectively) or as paths to credentials and the grid map file. Similarly, the *service authorization chain* object or a comma separated

list of PDP names can be specified. In each of these cases, if the properties are configured as filenames or PDP names as the case may be, the helper API in `org.globus.wsrf.impl.security.descriptor.ResourceSecurityConfig` can be used to load the classes. The credentials, grid map file and PDPs specified in the authorization chain are loaded if the property `initialized` in the descriptor is set to `false`.

For example, the code snippet below creates a descriptor that has a grid map file and an authorization chain. When `config.init()` is called, the grid map file is loaded and an instance of the service authorization chain class is created. The configuration information for the service authorization chain is by default picked up from the global deployment descriptor. To provide for other PDP configurations it needs to be set programmatically, as shown below.

```
ResourceSecurityDescriptor desc = new ResourceSecurityDescriptor();
desc.setGridMapFile("foo/bar/gridmap");
desc.setAuthz("customAuthz:org.globus.some.customAuthz foo1:org.foo.barAuthz");
ResourceSecurityConfig config = new ResourceSecurityConfig(desc);
config.init();
```

If the descriptor property changes, a reload can be forced by setting `setInitialized` to `false`:

```
desc.setInitialized(false);
desc.setGridMapFile("foo/bar/newGridMap");
config.init();
```

GridMap and Subject objects can also be set directly, i.e. without configuring files to be read:

```
desc.setInitialized(false);
GridMap map = new GridMap();
map.map("Some user DN", "userid");
desc.setGridMap(map);
```

Service Authorization can also be set directly by creating an object of type `org.globus.wsrf.impl.security.authorization.ServiceAuthorizationChain`. The chain needs to be initialized with one or more objects implementing the `org.globus.wsrf.security.authorization.PDPConfig` interface. The `org.globus.wsrf.impl.security.descriptor.ResourceSecurityDescriptor` class has an API to initialize a PDP using the PDPConfig class. The distribution has a few sample classes that implement the `org.globus.wsrf.security.authorization.PDPConfig` interface and are described below:

- `org.globus.wsrf.impl.security.authorization.ContainerPDPConfig`: Obtains configuration information from the global deployment descriptor.
- `org.globus.wsrf.impl.security.authorization.ServicePropertiesPDPConfig`: Obtains configuration information from a service's deployment descriptor.
- `org.globus.wsrf.impl.security.authorization.ResourcePDPConfig`: Obtains configuration information from a hashmap stored in memory.

Examples:

This sample creates a authorization chain and sets it on the resource security descriptor:

```
// Create a resource security descriptorResourceSecurityDescriptor
ResourceSecurityDescriptor desc = new ResourceSecurityDescriptor();
// Configure a chain of PDPsString
String authzChain = "identityAuthz custom:org.something.CustomAuthz";
// Create configuration object that implements PDPConfig
ResourcePDPConfig config = new ResourcePDPConfig(authzChain);
// Set properties that are required by the PDPs on the configuration object.
```

```
// Property used by Identity authorization: scope, property name, property value
config.setProperty("idenAuthz", "identity", "O=this, OU=is expected, CN=identity");
// Property used by CustomAuthz: scope, property name, property value
config.setProperty("custom", "someProp", "foo");
desc.setAuthzChain(authzChain, config, "Name of Chain", "Some id");
```

7. Container-only security configuration

Other than the security properties that have been described in Security Descriptor[10], two more properties are exclusive to the container security descriptor.

- When *GSI Secure Conversation* is used, a security context is established. A sweeper task is run every 10 minutes to delete all expired contexts. This interval can be set (in milliseconds) in the container security descriptor as shown below:

```
<securityConfig xmlns="http://www.globus.org">
   ...
   <context-lifetime value="10000"/>
   ...
</securityConfig>
```

- For message level security one may also set the amount of time for which to track received messages for the purpose of preventing replay attacks. Messages outside of this window will be rejected automatically, whereas messages within this window are checked against recently received messages through the use of the message UUID. This window can be configured (in milliseconds) as shown below:

```
<securityConfig xmlns="http://www.globus.org">
   ...
   <replay-attack-interval value="100"/>
   ...
</securityConfig>
```

8. Other configuration

The container security descriptor can be set up at the command line (rather than configured in the deployment descriptor as described here) by using the *-containerDesc* option when starting up the container using *globus-start-container*.

```
bin\globus-start-container -containerDesc path/to/desc
```

[10] #Descriptor

Chapter 14. GT 4.0 Migrating Guide for WS A &A Authorization Framework

The following provides available information about migrating from previous versions of the Globus Toolkit.

1. Migrating from GT2

This component did not exist in GT2.

2. Migrating from GT3

While the GT4 version of this component has similar features to the GT3 version, some of the configuration methodology has changed and some features have been enhanced. Refer to Section 3.5, “Configuring authorization mechanisms” for changes in configuration.

Chapter 15. Command line Interface

There is no support for this type of interface.

GT 4.0 Security Glossary

C

Certificate Authority (CA)	An entity that issues certificates.
CA Certificate	The CA's certificate. This certificate is used to verify signature on certificates issued by the CA. GSI typically stores a given CA certificate in `/etc/grid-security/certificates/<hash>.0`, where <hash> is the hash code of the CA identity.
CA Signing Policy	The CA signing policy is used to place constraints on the information you trust a given CA to bind to public keys. Specifically it constrains the identities a CA is trusted to assert in a certificate. In GSI the signing policy for a given CA can typically be found in `/etc/grid-security/certificates/<hash>.signing_policy`, where <hash> is the hash code of the CA identity. For more information see [add link].
certificate	A public key and information about the certificate owner bound together by the digital signature of a CA. In the case of a CA certificate the certificate is self signed, i.e. it was signed using its own private key.
Certificate Revocation List (CRL)	A list of revoked certificates generated by the CA that originally issued them. When using GSI this list is typically found in `/etc/grid-security/certificates/<hash>.r0`, where <hash> is the hash code of the CA identity.
certificate subject	A identifier for the certificate owner, e.g. "/DC=org/DC=doegrids/OU=People/CN=John Doe 123456". The subject is part of the information the CA binds to a public key when creating a certificate.
credentials	The combination of a certificate and the matching private key.

E

End Entity Certificate (EEC)	A certificate belonging to a non-CA entity, e.g. you, me or the computer on your desk.

G

GAA Configuration File	A file that configures the Generic Authorization and Access control GAA libraries. When using GSI this file is typically found in `/etc/grid-security/gsi-gaa.conf`.
grid map file	A file containing entries mapping certificate subjects to local user names. This file can also serve as a access control list for GSI enabled services and is typically found in `/etc/grid-security/grid-mapfile`. For more information see the Gridmap file[1].

[1] http://www.globus.org/toolkit/docs/4.0/security/prewsaa/Pre_WS_AA_Public_Interfaces.html#prewsaa-env-gridmapfile

grid security directory — The directory containing GSI configuration files such as the GSI authorization callout configuration and GAA configuration files. Typically this directory is `/etc/grid-security`. For more information see Grid security directory[2].

GSI authorization callout configuration file — A file that configures authorization callouts to be used for mapping and authorization in GSI enabled services. When using GSI this file is typically found in `/etc/grid-security/gsi-authz.conf`.

H

host certificate — An EEC belonging to a host. When using GSI this certificate is typically stored in `/etc/grid-security/hostcert.pem`. For more information on possible host certificate locations see the Credentials[3].

host credentials — The combination of a host certificate and its corresponding private key..

P

private key — The private part of a key pair. Depending on the type of certificate the key corresponds to it may typically be found in `$HOME/.globus/userkey.pem` (for user certificates), `/etc/grid-security/hostkey.pem` (for host certificates) or `/etc/grid-security/<service>/<service>key.pem` (for service certificates). For more information on possible private key locations see the Credentials[4]

proxy certificate — A short lived certificate issued using a EEC. A proxy certificate typically has the same effective subject as the EEC that issued it and can thus be used in its stead. GSI uses proxy certificates for single sign on and delegation of rights to other entities.

proxy credentials — The combination of a proxy certificate and its corresponding private key. GSI typically stores proxy credentials in `/tmp/x509up_u<uid>` , where <uid> is the user id of the proxy owner.

public key — The public part of a key pair used for cryptographic operations (e.g. signing, encrypting).

S

service certificate — A EEC for a specific service (e.g. FTP or LDAP). When using GSI this certificate is typically stored in `/etc/grid-security/<service>/<service>cert.pem`. For more information on possible service certificate locations see the Credentials[5].

service credentials — The combination of a service certificate and its corresponding private key.

[2] http://www.globus.org/toolkit/docs/4.0/security/prewsaa/Pre_WS_AA_Public_Interfaces.html#prewsaa-env-gridsecurity
[3] http://www.globus.org/toolkit/docs/4.0/security/prewsaa/Pre_WS_AA_Public_Interfaces.html#prewsaa-env-credentials
[4] http://www.globus.org/toolkit/docs/4.0/security/prewsaa/Pre_WS_AA_Public_Interfaces.html#prewsaa-env-credentials
[5] http://www.globus.org/toolkit/docs/4.0/security/prewsaa/Pre_WS_AA_Public_Interfaces.html#prewsaa-env-credentials

T

transport-level security — Uses transport-level security (TLS) mechanisms.

trusted CAs directory — The directory containing the CA certificates and signing policy files of the CAs trusted by GSI. Typically this directory is `/etc/grid-security/certificates`. For more information see Grid security directory[6].

U

user certificate — A EEC belonging to a user. When using GSI this certificate is typically stored in `$HOME/.globus/usercert.pem`. For more information on possible user certificate locations see Credentials[7].

user credentials — The combination of a user certificate and its corresponding private key.

[6] http://www.globus.org/toolkit/docs/4.0/security/prewsaa/Pre_WS_AA_Public_Interfaces.html#prewsaa-env-gridsecurity
[7] http://www.globus.org/toolkit/docs/4.0/security/prewsaa/Pre_WS_AA_Public_Interfaces.html#prewsaa-env-credentials