GT 4.0: Security: Message & Transport Level Security

GT 4.0: Security: Message & Transport Level Security

Table of Contents

1. Key Concepts 1
    1. Overview 1
    2. Conceptual Details 1
    3. Related Documents 4
2. 4.0.0 Release Notes 8
    1. Component Overview 8
    2. Feature Summary 8
    3. Changes Summary 8
    4. Bug Fixes 9
    5. Known Problems 9
    6. Technology Dependencies 9
    7. Tested Platforms 9
    8. Backward Compatibility Summary 10
    9. For More Information 10
3. 4.0.1 Release Notes 11
    1. Introduction 11
    2. Changes Summary 11
    3. Bug Fixes 11
    4. Known Problems 11
    5. For More Information 11
4. 4.0.2 Release Notes 12
    1. Introduction 12
    2. Changes Summary 12
    3. Bug Fixes 12
    4. Known Problems 12
    5. For More Information 12
5. 4.0.3 Release Notes 13
    1. Introduction 13
    2. Changes Summary 13
    3. Bug Fixes 13
    4. Known Problems 13
    5. For More Information 13
6. 4.0.4 Release Notes 14
    1. Introduction 14
    2. Changes Summary 14
    3. Bug Fixes 14
    4. Known Problems 14
    5. For More Information 14
7. Admin Guide 15
    1. Introduction 15
    2. Building and Installing 15
    3. Configuring 15
    4. Deploying 16
    5. Testing 16
    6. Security Considerations 16
    7. Troubleshooting 16
8. User's Guide 19
    1. Introduction 19
    2. Command-line tools 19
    3. Graphical user interfaces 19
    4. Troubleshooting 19

9. Developer's Guide 22
1. Introduction 22
2. Before you begin 22
3. Architecture and design overview 23
4. Public interface 28
5. Usage scenarios 28
6. Tutorials 28
7. Debugging 28
8. Troubleshooting 28
9. Related Documentation 29
10. Fact Sheet 30
1. Brief overview 30
2. Summary of features 30
3. Usability summary 30
4. Backward compatibility summary 30
5. Technology dependencies 31
6. Tested platforms 31
7. Associated standards 31
8. For More Information 32
11. Public Interfaces 33
1. Semantics and syntax of APIs 33
2. Semantics and syntax of the WSDL 33
3. Framework-level Protocols 35
4. Command-line tools 36
5. Overview of Graphical User Interfaces 36
6. Semantics and syntax of domain-specific interface 36
7. Configuration interface 42
8. Environment variable interface 42
12. Quality Profile 43
1. Test coverage reports 43
2. Code analysis reports 43
3. Outstanding bugs 43
4. Bug Fixes 43
5. Performance reports 44
13. Migration Guide 45
1. Migrating from GT2 45
2. Migrating from GT3 45
GT 4.0 Security Glossary 46

List of Tables

11.1. Client side security properties 39

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.

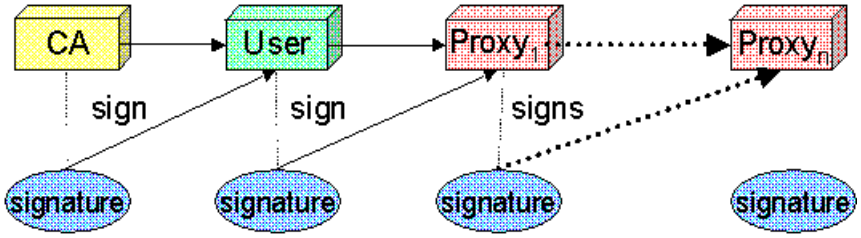
We have also prototyped the use of cryptographic smartcards in conjunction with GSI. This allows users to store their private key on a smartcard rather than in a file system, making it still more difficult for others to gain access to the key.

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 Message & Transport Level Security Release Notes

1. Component Overview

The Web Services portion of GT 4.0 uses SOAP over HTTP for communicating messages.

WS Authentication and Authorization Message-Level Security implements the WS-Security standard and the WS-SecureConversation specification to provide message protection for SOAP messages. Features include authentication of the sender, encryption of the message, integrity protection of the message and replay protection.

WS Authentication and Authorization Transport-Level Security provides a secure channel by using HTTP over SSL/TLS (HTTPS) for transporting the messages. This security mechanism supports all of the security features provided by SSL/TLS with the addition of support for X.509 *Proxy Certificates*.

2. Feature Summary

Features new in GT 4.0

- Compliance with published IBM/Microsoft WS-Trust and WS-SecureConversation specifications
- Compliance with the Web Services Security 1.0 standard
- HTTPS support

Other Supported Features

- Message encryption, integrity protection and replay attack prevention
- Establishment of a session key for light-weight message protection

Deprecated Features

- GT 3.2 SecureConversation protocol

3. Changes Summary

3.1. Support for anonymous authentication

Support for anonymous authentication was added to the transport security (HTTPS) authentication mechanism.

3.2. Internationalization

The message and transport level security code has been internationalized.

4. Bug Fixes

- Bug 2178:[1] Any SOAP headers used for dispatching need to be secured
- Bug 2179:[2] Fix up replay attack prevention
- Bug 2193:[3] No local subject error
- Bug 2207:[4] Missing security error 'timestampNotOk'
- Bug 2371:[5] Better error reporting for security descriptor parsing errors
- Bug 2523:[6] Core should cause error/warning for bad containerSecDesc file

5. Known Problems

No known problems at this time.

6. Technology Dependencies

WS Authentication and Authorization Message & Transport Level Security depends on the following GT components:

- The C implementation depends on C WS Core.
- The Java implementation depends on Java WS Core.

WS Authentication and Authorization Message & Transport Level Security depends on the following 3rd party software:

- Apache WSFX Security Libraries
- PureTLS Libraries
- BouncyCastle JCE provider
- Cryptix Libraries
- Apache XML Security Libraries

7. Tested Platforms

WS Authentication and Authorization Message & Transport Level Security should work on any platform that supports J2SE 1.3.1 or higher.

Tested Platforms for WS Authentication and Authorization Message & Transport Level Security

- Linux (Red Hat 7.3)

[1] http://bugzilla.globus.org/globus/show_bug.cgi?id=2178
[2] http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=2179
[3] http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=2193
[4] http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=2207
[5] http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=2371
[6] http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=2523

- Windows 2000
- Solaris 9

8. Backward Compatibility Summary

Protocol changes in WS Authentication and Authorization Message-Level Security since GT 3.2

- WS-SecureConversation updated to reflect published IBM/Microsoft specification.
- Web Services Security updated to reflect published OASIS standard (1.0).

API changes since GT 3.2

- N/A

Exception changes since GT 3.2

- N/A

Schema changes since GT 3.2

- N/A

9. For More Information

Click here[7] for more information about this component.

[7] index.html

Chapter 3. GT 4.0.1 Incremental Release Notes: Message/Transport-level Security

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 Java WS Core 4.0 Release Notes[1].

2. Changes Summary

Other than bug fixes, no changes have occurred for Message/Transport-level security.

3. Bug Fixes

The following bugs were fixed for Message/Transport-level security:

- Bug 3501:[2] Reject limited proxy configuration not honored if Secure Transport is used
- Bug 3504:[3] Secure notifications fail without default credentials.

4. Known Problems

No new problems are known to exist for Message/Transport-level security.

5. For More Information

Click here[4] for more information about this component.

[1] http://www.globus.org/toolkit/docs/4.0/security/message/WS_AA_Message_Level_Release_Notes.html
[2] http://bugzilla.globus.org/globus/show_bug.cgi?id=3501
[3] http://bugzilla.globus.org/globus/show_bug.cgi?id=3504
[4] index.html

Chapter 4. GT 4.0.2 Incremental Release Notes: Message/Transport-level Security

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 Java WS Core 4.0 Release Notes[1].

2. Changes Summary

Other than bug fixes, no changes have occurred for Message/Transport-level security.

3. Bug Fixes

The following bugs were fixed for Message/Transport-level security:

- Bug 3891:[2] Public credentials of client is not populated in PEER_SUBJECT object when secure transport is used.
- Bug 4188:[3] Container does not recover from expired proxy.

4. Known Problems

No new problems are known to exist for Message/Transport-level security.

5. For More Information

Click here[4] for more information about this component.

[1] http://www.globus.org/toolkit/docs/4.0/security/message/WS_AA_Message_Level_Release_Notes.html
[2] http://bugzilla.globus.org/globus/show_bug.cgi?id=3891
[3] http://bugzilla.globus.org/globus/show_bug.cgi?id=4188
[4] index.html

Chapter 5. GT 4.0.3 Incremental Release Notes: Message/Transport-level Security

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 Message/Transport-level Security 4.0 Release Notes[1].

2. Changes Summary

Other than bug fixes, no changes have occurred for Message/Transport-level security since GT 4.0.2.

3. Bug Fixes

The following bugs have been fixed for Message/Transport-level security since GT 4.0.2:

- Bug 4647:[2] Insecure temporary file handling in the Globus Toolkit
- Bug 4648:[3] grid-proxy-init proxy credentials race condition

4. Known Problems

No new problems are known to exist for Message/Transport-level security at the time of the 4.0.3 release.

5. For More Information

Click here[4] for more information about this component.

[1] http://www.globus.org/toolkit/docs/4.0/security/message/WS_AA_Message_Level_Release_Notes.html
[2] http://bugzilla.globus.org/globus/show_bug.cgi?id=4647
[3] http://bugzilla.globus.org/globus/show_bug.cgi?id=4647
[4] index.html

Chapter 6. GT 4.0.4 Incremental Release Notes: Message/Transport-level Security

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 Message/Transport-level Security 4.0 Release Notes[1].

2. Changes Summary

No changes have been made to Message/Transport-level security since GT 4.0.3.

3. Bug Fixes

No bugs have been fixed for Message/Transport-level security since GT 4.0.3.

4. Known Problems

No new problems are known to exist at the time of the GT 4.0.4 release.

5. For More Information

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

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

Chapter 7. GT4 Message & Transport Level Security Admin Guide

1. Introduction

This guide contains advanced configuration information for system administrators working with Message/Transport-level Security. 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!

The main administration issues for this component deal with configuring credential related settings. There are multiple mechanisms for doing this:

- Security Descriptors
  - Container Security Descriptor: This is the *preferred* mechanism.
  - Service Security Descriptor
- CoG properties
- Environment variables
- Relying on default behavior. The only default behaviors available concern the proxy file and trusted certificates locations.

More information on these mechanisms can be found in the public interface guide[2].

2. Building and Installing

The GT4 Message & Transport Level Security component is currently installed as part of the GT4 Java WS Core component. More information on installing this component can be found here[3].

3. Configuring

3.1. Configuration overview

Configuration of service-side security settings can be achieved in two ways. The preferred way is to provide these settings in a security descriptor, although it is also possible to manipulate security settings via CoG properties.

[1] ../../admin/docbook/
[2] WS_AA_Message_Level_Public_Interfaces.html
[3] http://www.globus.org/toolkit/docs/4.0/common/javawscore/admin-index.html#installing

3.2. Syntax of the interface

Information on the syntax of security descriptors can be found here[4].

Information on available CoG security properties can be found here[5].

4. Deploying

The GT4 Message & Transport Level Security component is currently deployed as part of the GT4 Java WS Core component.

5. Testing

The GT4 Message & Transport Security tests are intermingled with tests for the GT4 Authorization Framework and thus are also executed when running the authorization tests. Information on how to run the authorization tests can be found here[6].

6. Security Considerations

There is no information at this time.

6.1. File permissions

The Java security code currently does not enforce secure permissions and, implicitly, file ownership requirements on any of the security related files, e.g. configuration and credential files. It is thus important that administrators ensure that the relevant files have correct permissions and ownership. Permissions should generally be as restrictive as possible, i.e. *private keys* should be readable only by the file owner and other files should be writable by owner only, and the files should generally be owned by the globus user (the requirements that the C code enforces are documented here[7]).

7. Troubleshooting

7.1. Credential Errors

The following are some common problems that may cause clients or servers to report that credentials are invalid:

7.1.1. Your proxy credential may have expired

Use **grid-proxy-info** to check whether the *proxy credential* has actually expired. If it has, generate a new proxy with **grid-proxy-init**.

7.1.2. The system clock on either the local or remote system is wrong

This may cause the server or client to conclude that a credential has expired.

[4] http://www.globus.org/toolkit/docs/4.0/security/authzframe/security_descriptor.html
[5] http://www.globus.org/cog/distribution/1.2/FAQ.TXT
[6] http://www.globus.org/toolkit/docs/4.0/security/authzframe/admin-index.html#s-authzframe-admin-testing
[7] http://www.globus.org/toolkit/docs/4.0/security/prewsaa/Pre_WS_AA_Public_Interfaces.html#s-prewsaa-iface-config-permissions

7.1.3. Your *end-user certificate* may have expired

Use **grid-cert-info** to check your certificate's expiration date. If it has expired, follow your CA's procedures to get a new one.

7.1.4. The permissions may be wrong on your proxy file

If the permissions on your proxy file are too lax (for example, if others can read your proxy file), Globus Toolkit clients will not use that file to authenticate. You can "fix" this problem by changing the permissions on the file or by destroying it (with **grid-proxy-destroy**) and creating a new one (with **grid-proxy-init**). However, it is still possible that someone else has made a copy of that file during the time that the permissions were wrong. In that case, they will be able to impersonate you until the proxy file expires or your permissions or end-user certificate are revoked, whichever happens first.

7.1.5. The permissions may be wrong on your private key file

If the permissions on your end user certificate *private key* file are too lax (for example, if others can read the file), **grid-proxy-init** will refuse to create a *proxy certificate*. You can "fix" this by changing the permissions on the private key file; however, you will still have a much more serious problem: it's possible that someone has made a copy of your private key file. Although this file is encrypted, it is possible that someone will be able to decrypt the private key, at which point they will be able to impersonate you as long as your end user certificate is valid. You should contact your CA to have your end-user certificate revoked and get a new one.

7.1.6. The remote system may not trust your CA

Verify that the remote system is configured to trust the CA that issued your end-entity certificate. See the Administrator's Guide[8] for details.

7.1.7. You may not trust the remote system's CA

Verify that your system is configured to trust the remote CA (or that your environment is set up to trust the remote CA). See the Administrator's Guide[9] for details.

7.1.8. There may be something wrong with the remote service's credentials

It is sometimes difficult to distinguish between errors reported by the remote service regarding your credentials and errors reported by the client interface regarding the remote service's credentials. If you can't find anything wrong with your credentials, check for the same conditions (or ask a remote administrator to do so) on the remote system.

7.2. Some tools to validate certificate setup

7.2.1. Check that the user certificate is valid

```
openssl verify -CApath /etc/grid-security/certificates
  -purpose sslclient ~/.globus/usercert.pem
```

[8] http://www.globus.org/toolkit/docs/4.0/admin/docbook/
[9] http://www.globus.org/toolkit/docs/4.0/admin/docbook/

7.2.2. Connect to the server using s_clien

```
openssl s_client -ssl3 -cert ~/.globus/usercert.pem -key ~/.globus/userkey.pem -CApath /et
```

Here <host:port> denotes the server and port you connect to.

If it prints an error and puts you back at the command prompt, then it typically means that the *server* has closed the connection, i.e. that the server was not happy with the client's certificate and verification. Check the SSL log on the server.

If the command "hangs" then it has actually opened a telnet style (but secure) socket, and you can "talk" to the server.

You should be able to scroll up and see the subject names of the server's verification chain:

```
depth=2 /DC=net/DC=ES/O=ESnet/OU=Certificate Authorities/CN=ESnet Root CA 1
verify return:1
depth=1 /DC=org/DC=DOEGrids/OU=Certificate Authorities/CN=DOEGrids CA 1
verify return:1
depth=0 /DC=org/DC=doegrids/OU=Services/CN=wiggum.mcs.anl.gov
verify return:1
```

In this case there were no errors. Errors would give you an extra line next to the subject name of the certificate that caused the error

7.2.3. Check that the server certificate is valid

Requires root login on server.

```
    openssl verify -CApath /etc/grid-security/certificates -purpose sslserver /etc/grid-se
```

Chapter 8. GT4 Message & Transport Level Security User's Guide

1. Introduction

The main user issues for this component deal with configuring credential-related settings. There are multiple mechanisms for doing this:

- Command line options (these are application-specific)
- CoG properties
- Environment variables
- Relying on default behavior. The only default behaviors available concern the proxy file and trusted certificates locations.

More information on these mechanisms can be found in the public interface guide[1].

2. Command-line tools

This component has no command line tools.

3. Graphical user interfaces

This component has no associated GUIs.

4. Troubleshooting

4.1. Credential Errors

The following are some common problems that may cause clients or servers to report that credentials are invalid:

4.1.1. Your proxy credential may have expired

Use **grid-proxy-info** to check whether the *proxy credential* has actually expired. If it has, generate a new proxy with **grid-proxy-init**.

4.1.2. The system clock on either the local or remote system is wrong

This may cause the server or client to conclude that a credential has expired.

4.1.3. Your *end-user certificate* may have expired

Use **grid-cert-info** to check your certificate's expiration date. If it has expired, follow your CA's procedures to get a new one.

[1] WS_AA_Message_Level_Public_Interfaces.html

4.1.4. The permissions may be wrong on your proxy file

If the permissions on your proxy file are too lax (for example, if others can read your proxy file), Globus Toolkit clients will not use that file to authenticate. You can "fix" this problem by changing the permissions on the file or by destroying it (with **grid-proxy-destroy**) and creating a new one (with **grid-proxy-init**). However, it is still possible that someone else has made a copy of that file during the time that the permissions were wrong. In that case, they will be able to impersonate you until the proxy file expires or your permissions or end-user certificate are revoked, whichever happens first.

4.1.5. The permissions may be wrong on your private key file

If the permissions on your end user certificate *private key* file are too lax (for example, if others can read the file), **grid-proxy-init** will refuse to create a *proxy certificate*. You can "fix" this by changing the permissions on the private key file; however, you will still have a much more serious problem: it's possible that someone has made a copy of your private key file. Although this file is encrypted, it is possible that someone will be able to decrypt the private key, at which point they will be able to impersonate you as long as your end user certificate is valid. You should contact your CA to have your end-user certificate revoked and get a new one.

4.1.6. The remote system may not trust your CA

Verify that the remote system is configured to trust the CA that issued your end-entity certificate. See the Administrator's Guide[2] for details.

4.1.7. You may not trust the remote system's CA

Verify that your system is configured to trust the remote CA (or that your environment is set up to trust the remote CA). See the Administrator's Guide[3] for details.

4.1.8. There may be something wrong with the remote service's credentials

It is sometimes difficult to distinguish between errors reported by the remote service regarding your credentials and errors reported by the client interface regarding the remote service's credentials. If you can't find anything wrong with your credentials, check for the same conditions (or ask a remote administrator to do so) on the remote system.

4.2. Some tools to validate certificate setup

4.2.1. Check that the user certificate is valid

```
openssl verify -CApath /etc/grid-security/certificates
  -purpose sslclient ~/.globus/usercert.pem
```

4.2.2. Connect to the server using s_clien

```
openssl s_client -ssl3 -cert ~/.globus/usercert.pem -key ~/.globus/userkey.pem -CApath /et
```

Here <host:port> denotes the server and port you connect to.

[2] http://www.globus.org/toolkit/docs/4.0/admin/docbook/
[3] http://www.globus.org/toolkit/docs/4.0/admin/docbook/

If it prints an error and puts you back at the command prompt, then it typically means that the *server* has closed the connection, i.e. that the server was not happy with the client's certificate and verification. Check the SSL log on the server.

If the command "hangs" then it has actually opened a telnet style (but secure) socket, and you can "talk" to the server.

You should be able to scroll up and see the subject names of the server's verification chain:

```
depth=2 /DC=net/DC=ES/O=ESnet/OU=Certificate Authorities/CN=ESnet Root CA 1
verify return:1
depth=1 /DC=org/DC=DOEGrids/OU=Certificate Authorities/CN=DOEGrids CA 1
verify return:1
depth=0 /DC=org/DC=doegrids/OU=Services/CN=wiggum.mcs.anl.gov
verify return:1
```

In this case there were no errors. Errors would give you an extra line next to the subject name of the certificate that caused the error

4.2.3. Check that the server certificate is valid

Requires root login on server.

```
    openssl verify -CApath /etc/grid-security/certificates -purpose sslserver /etc/grid-se
```

Chapter 9. GT4 Message & Transport Level Security Developer's Guide

1. Introduction

This component contains mainly framework level code and as such developing services and clients utilizing this component does in general involve either programmatically or declaratively driving the framework level security code. Now, what does this entail? On the programmatic side of things it involves acquiring credentials, passing these credentials on to the framework and setting various authentication and protection related flags, either in a descriptor or as properties on a stub object. On the declarative side it involves setting up security descriptors, both client and service side, to prescribe the security policy used to drive the security framework code.

2. Before you begin

2.1. Feature summary

Features new in GT 4.0

- Compliance with published IBM/Microsoft WS-Trust and WS-SecureConversation specifications
- Compliance with the Web Services Security 1.0 standard
- HTTPS support

Other Supported Features

- Message encryption, integrity protection and replay attack prevention
- Establishment of a session key for light-weight message protection

Deprecated Features

- GT 3.2 SecureConversation protocol

2.2. Tested platforms

WS Authentication and Authorization Message & Transport Level Security should work on any platform that supports J2SE 1.3.1 or higher.

Tested Platforms for WS Authentication and Authorization Message & Transport Level Security

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

2.3. Backward compatibility summary

Protocol changes in WS Authentication and Authorization Message-Level Security since GT 3.2

- WS-SecureConversation updated to reflect published IBM/Microsoft specification.
- Web Services Security updated to reflect published OASIS standard (1.0).

API changes since GT 3.2

- N/A

Exception changes since GT 3.2

- N/A

Schema changes since GT 3.2

- N/A

2.4. Technology dependencies

WS Authentication and Authorization Message & Transport Level Security depends on the following GT components:

- The C implementation depends on C WS Core.
- The Java implementation depends on Java WS Core.

WS Authentication and Authorization Message & Transport Level Security depends on the following 3rd party software:

- Apache WSFX Security Libraries
- PureTLS Libraries
- BouncyCastle JCE provider
- Cryptix Libraries
- Apache XML Security Libraries

2.5. Security considerations

There is no information at this time.

3. Architecture and design overview

3.1. Transport Security

The toolkit by default is deployed with our implementation of transport security, which is based on HTTP over SSL, also known as HTTPS, with modifications to path validation to enable X.509 *Proxy Certificate* support. In contrast to the GT3 version of the toolkit, the default transport security enabled in the toolkit does not support delegation of proxy certificates as part of the security handshake.

However the underlying security libraries and handlers required for secure transport with delegation, also known as HTTPG, is still supported and shipped as a part of the CoG library. The GT4 Java WS code base and configuration can be modified to use the HTTPG protocol is required.

Transport security is implemented by layering on top of the *GSISocket* class provided in JGlobus. This class deals with the security related aspects of connection establishment as well as message protection. The socket interface serves as a abstraction layer that allows the HTTP protocol handling code to be unaware of the underlying security properties of the connection.

Container-level credentials are required and, irrespective of security settings on the service being accessed, these credentials are used for handshake.

3.1.1. Server Side Security

On the server side transport security is enabled by simply switching a non-secure socket implementation with the *GSISocket* implementation. In addition to this change some code was added to propagate authentication information and message protection settings to the relevant security handlers, in particular the authorization and security policy handlers.

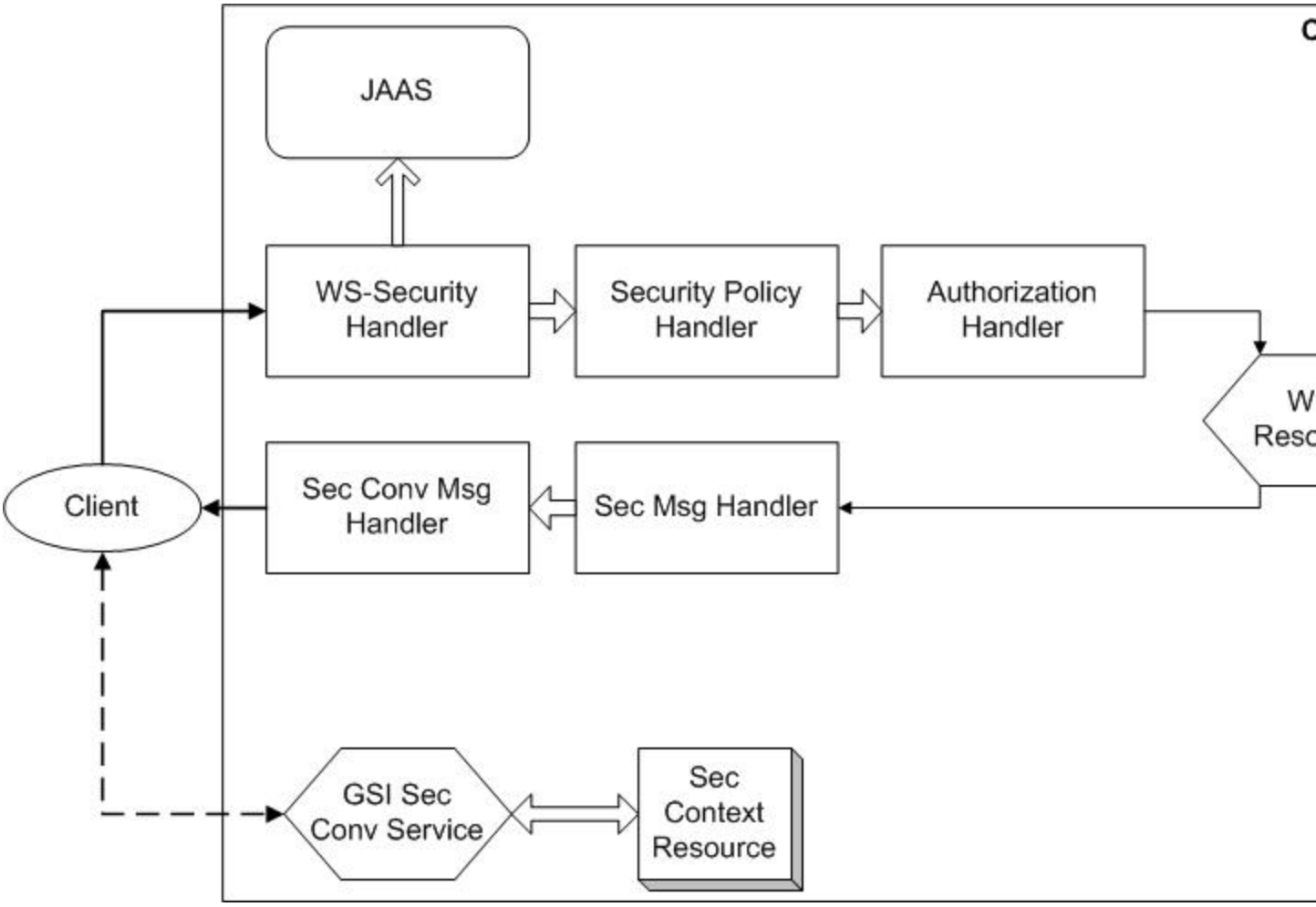
3.1.2. Client Side Security

On the client side transport security is similarly enabled by switching a non-secure socket implementation with the *GSISocket* implementation and registering a protocol handler for HTTPS that uses the secure socket implementation. In practice this means that any messages targeted at a HTTPS endpoint will, irregardless of any stub properties, be authenticated and protected. It also means that any messages sent to a HTTP endpoint will not be secured, again irregardless of any stub properties. Stub properties are only used to communicate the desired message protection level, i.e. either integrity only or integrity and privacy.

3.2. Message Level Security

3.2.1. Server Side Security

This section aims to describe the message flow and processing that occurs for a security-enabled service. The figure below shows the JAX-RPC handlers that are involved in security related message processing on a server.



GT4 provides two mechanisms, GSI Secure Conversation and GSI Secure Message security, for authentication and secure communication.

In the GSI Secure Conversation approach the client establishes a context with the server before sending any data. This context serves to authenticate the client identity to the server and to establish a shared secret using a collocated GSI Secure Conversation Service. Once the context establishment is complete the client can securely invoke an operation on the service by signing or encrypting outgoing messages using the shared secret captured in the context.

The GSI Secure Message approach differs in that no context is established before invoking an operation. The client simply uses existing keying material, such as an X509 *end entity certificate*, to secure messages and authenticate itself to the service.

Securing of messages in the GSI Secure Conversation approach, i.e. using a shared secret, requires less computational effort than using existing keying material in the GSI Secure Message approach. This allows the client to trade off the extra step of establishing a context to enable more computationally efficient messages protection once that context has been established.

3.2.2. Message Processing

When a message arrives from the client the SOAP engine invokes several security related handlers.

The first of these handlers, the WS-Security handler, searches the message for any WS-Security headers. From these headers it extracts any keying material, which can be either in the form of an X509 certificate and associated certificate chain or a reference to a previously established secure conversation session. It also checks any signatures and/or decrypts elements in the SOAP body. The handler then populates a peer JAAS subject object with principals and any associated keying material whose veracity was ascertained during the signature checking or decryption step.

The next handler that gets invoked, the security policy handler, checks that incoming messages fulfill any security requirements the service may have. These requirements are specified, on a per-operation basis, as part of a security descriptor during service deployment. The security policy handler will also identify the correct JAAS subject to associate with the current thread of execution. Generally this means choosing between the peer subject populated by the WS-Security handler, the subject associated with the hosting environment and the subject associated with the service itself. The actual association is done by the pivot handler, a non-security handler not shown in the figure that handles the details of delivering the message to the service.

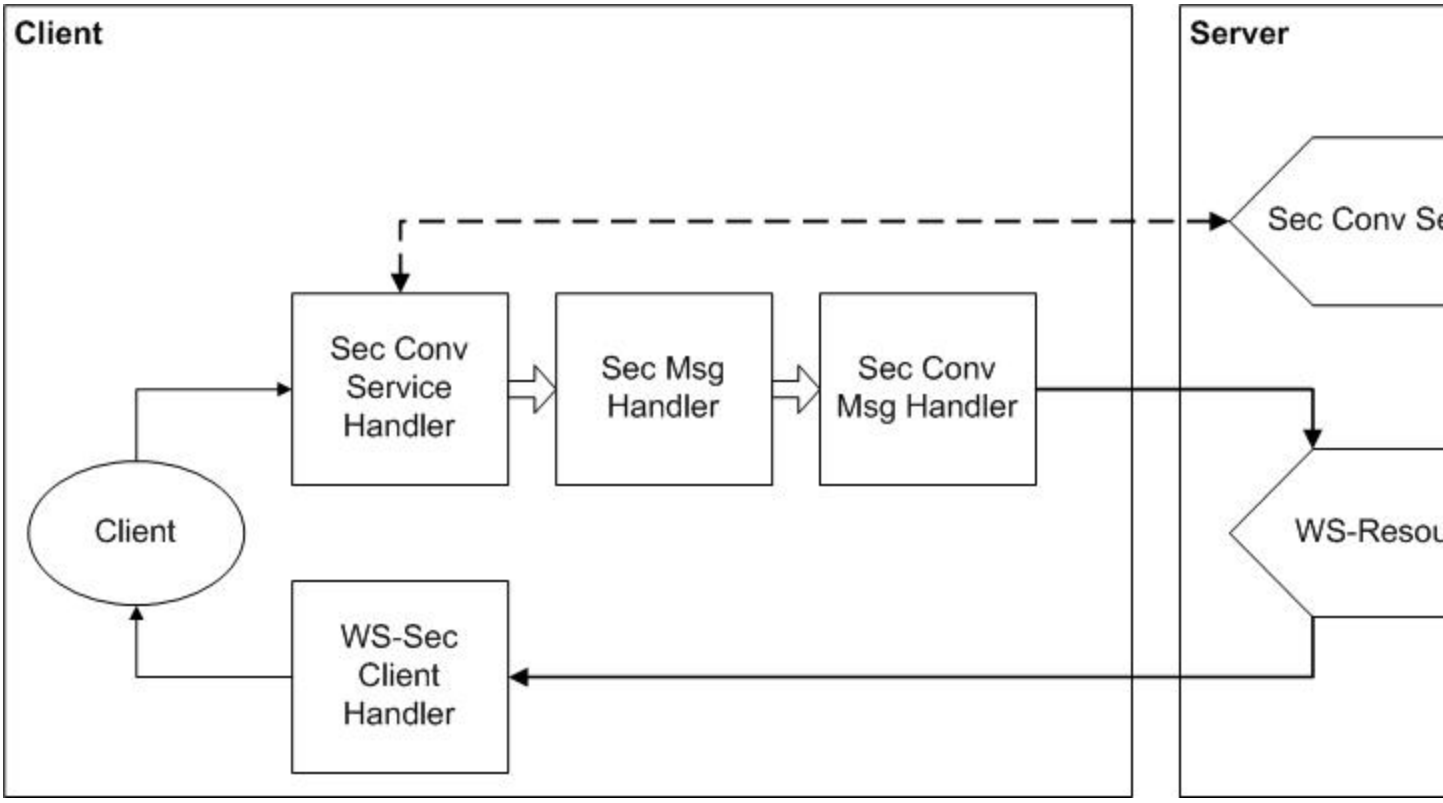
The security policy handler is followed by an authorization handler. This handler verifies that the principal established by the WS-Security handler is authorized to invoke the service. The type of authorization that is performed is specified as part of a deployment descriptor. More information can be found in there authorization framework documentation[1].

Once the message has passed the authorization handler it is finally handed off to the actual service for processing (discounting any non security related handlers, which are outside the scope of this document). Replies from the service back to the client are processed by two outbound handlers: the GSI Secure Conversation message handler and the GSI Secure Message handler. The GSI Secure Conversation message handler deals with encrypting and signing messages using a previously established security context, whereas the GSI Secure Message handler deals with messages by signing or encrypting the messages using X509 certificates. The operations that are actually performed depend on the message properties associated with the message by the inbound handlers, i.e. outbound messages will have the same security attributes as inbound messages. That being said, a service has the option of modifying the message properties if so desired. These handlers are identical to the client side handlers described in the following section.

3.2.3. Client Side Security

This section describes the security related message processing for Java-based GT4 clients. In contrast to the server side, where security is specified via deployment descriptors, client side security configuration is handled by the application. This means that a client side application has to explicitly pass information to the client side handlers on what type of security to use. This is also true for the case of services acting as clients. The below figure shows the JAX-RPC handlers that are involved in security related message processing on a server.

[1] ../authzframe/developer-index.html



3.2.4. Message Processing

The client side application can specify the use of either the GSI Secure Conversation security approach or the GSI Secure Message security approach. It does this by setting a per message property that is processed by the client side security handlers.

There are three outbound client side security handlers:

The secure conversation service handler is only operational if GSI Secure Conversation mode is in use. It establishes a security session with a secure conversation service collocated with the service with which the client aims to communicate. When the client sends the initial message to the service with a property indicating that session based security is required, this handler intercepts the message and establishes a security session. It will also authorize the service by comparing the service's principal/subject obtained during session establishment with a value provided by the client application. Once the session has been established the handler passes on the original message for further processing.

The next handler in the chain, the secure message handler, is only operational if GSI Secure Message mode is in use. It signs and/or encrypts messages using X.509 credentials.

The third outbound handler is operational only if GSI Secure Conversation mode is in use. It handles signing and/or encryption of messages using a security session established by the first handler.

The client side inbound handler (the WS-Security client handler) deals with verifying and decrypting any signed and/or encrypted incoming messages. In the case of GSI Secure Message operation it will also authorize the remote side in a similar fashion to the outbound secure conversation service handler.

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

5.1. Delegation

There are two ways a client can delegate its credential to a service: a) using Delegation Service b) using GSI Secure Conversation.

Client can delegate using the Delegation Service[3]. This method is independent of the security scheme used and can be reused across multiple invocations of the client to multiple services (provided the service are in the same hosting envorinment as the delegation service). The link provided has details on client side steps to delegate and service side code to get the delegated credential.

GSI Secure Conversation has delegation built into the protocol. Delegation can be requested by setting the `GSIConstants.GSI_MODE` property on ther Stub or using security descriptors as described here[4]. If full or limited delegation is performed, the client credential can be obtained from the message context as follows:

```
Subject subject = (Subject) msgCtx.getProperty(Constants.PEER_SUBJECT);
```

The server can be configured such that container, service or client credentials are used for the operation invoked. For the client credentials to be used, client should have delegated the credentials. Configuring this option is described here[5]. Note that this is a server side configuration. If `caller-identity` is chosen for the `run-as` configuration and the client's credentials have been successfully delegated, then the delegated credentials are associated with the current thread. The credentials in this case can be obtained as follows:

```
Subject subject = JaasSubject.getCurrentSubject();
```

6. Tutorials

There are no tutorials available at this time.

7. Debugging

There is no content available at this time.

8. Troubleshooting

There is no content available at this time.

[2] WS_AA_Message_Level_Public_Interfaces.html
[3] http://www.globus.org/toolkit/docs/4.0/security/delegation
[4] http://www.globus.org/toolkit/docs/4.0/security/authzframe/security_descriptor.html#s-authzfram-client-secdesc-secConv
[5] http://www.globus.org/toolkit/docs/4.0/security/authzframe/security_descriptor.html#s-authzframe-secdesc-configRunas

9. Related Documentation

See Section 7, “Associated standards” for a list of associated standards.

Chapter 10. GT4 Message & Transport Level Security Fact Sheet

1. Brief overview

The Web Services portion of GT 4.0 uses SOAP over HTTP for communicating messages.

WS Authentication and Authorization Message-Level Security implements the WS-Security standard and the WS-SecureConversation specification to provide message protection for SOAP messages. Features include authentication of the sender, encryption of the message, integrity protection of the message and replay protection.

WS Authentication and Authorization Transport-Level Security provides a secure channel by using HTTP over SSL/TLS (HTTPS) for transporting the messages. This security mechanism supports all of the security features provided by SSL/TLS with the addition of support for X.509 *Proxy Certificates*.

2. Summary of features

Features new in GT 4.0

- Compliance with published IBM/Microsoft WS-Trust and WS-SecureConversation specifications
- Compliance with the Web Services Security 1.0 standard
- HTTPS support

Other Supported Features

- Message encryption, integrity protection and replay attack prevention
- Establishment of a session key for light-weight message protection

Deprecated Features

- GT 3.2 SecureConversation protocol

3. Usability summary

Usability improvements for WS Authentication and Authorization Message- and Transport-Level Service:

- There is no content available at this time.

4. Backward compatibility summary

Protocol changes in WS Authentication and Authorization Message-Level Security since GT 3.2

- WS-SecureConversation updated to reflect published IBM/Microsoft specification.
- Web Services Security updated to reflect published OASIS standard (1.0).

API changes since GT 3.2

- N/A

Exception changes since GT 3.2

- N/A

Schema changes since GT 3.2

- N/A

5. Technology dependencies

WS Authentication and Authorization Message & Transport Level Security depends on the following GT components:

- The C implementation depends on C WS Core.
- The Java implementation depends on Java WS Core.

WS Authentication and Authorization Message & Transport Level Security depends on the following 3rd party software:

- Apache WSFX Security Libraries
- PureTLS Libraries
- BouncyCastle JCE provider
- Cryptix Libraries
- Apache XML Security Libraries

6. Tested platforms

WS Authentication and Authorization Message & Transport Level Security should work on any platform that supports J2SE 1.3.1 or higher.

Tested Platforms for WS Authentication and Authorization Message & Transport Level Security

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

7. Associated standards

Associated standards for WS A&A Message/Transport-level Security:

- WS-Security[1]
- WS-Security: X.509 Certificate Tokens[2]

[1] http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf
[2] http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0.pdf

- WS-Security: Username Tokens[3]
- WS-Trust[4]
- WS-Secure Conversation[5]
- WS-I Basic Security Profile[6]
- RFC 3820[7] Proxy Certificates
- RFC 2818[8] HTTP over TLS
- RFC 2246[9] TLS
- JAAS[10]

8. For More Information

Click here[11] for more information about this component.

[3] http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0.pdf
[4] ftp://www6.software.ibm.com/software/developer/library/ws-trust052004.pdf
[5] ftp://www6.software.ibm.com/software/developer/library/ws-secureconversation052004.pdf
[6] http://www.ws-i.org/Profiles/BasicSecurityProfile-1.0.html
[7] http://www.faqs.org/rfcs/rfc3820.html
[8] http://www.faqs.org/rfcs/rfc2818.html
[9] http://www.faqs.org/rfcs/rfc2246.html
[10] http://java.sun.com/products/jaas/
[11] index.html

Chapter 11. GT4 Message & Transport Level Security Public Interfaces

1. Semantics and syntax of APIs

1.1. Programming Model Overview

The security programming model differs between the client and server side. The client side model is programmatic in nature, i.e. security related code is driven by making actual function calls, whereas the server side model is declarative, i.e. security related settings are declared in a security descriptor. For more information on the available client side calls see here[1]. More information about the security descriptor can be found here[2].

1.2. Component API

- Stable interfaces:
  - org.globus.wsrf.security.Constants
  - org.globus.wsrf.security.SecureResource
  - org.globus.wsrf.security.SecurityManager
  - org.globus.wsrf.security.SecurityException
- Less stable interfaces:
  - org.globus.wsrf.impl.security.descriptor.ClientSecurityDescriptor
  - org.globus.wsrf.impl.security.descriptor.ResourceSecurityDescriptor

Documentation for these interfaces can be found here[3].

2. Semantics and syntax of the WSDL

2.1. Secure Conversation Service

2.1.1. Protocol overview

This service provides a mechanism for generating a security session, i.e the negotiation of a shared secret which may be used to secure a set of subsequent messages. It is based on the WS-Trust[4] and WS-SecureConversation[5] specifications.

[1] http://www.globus.org/toolkit/docs/4.0/security/message/WS_AA_Message_Level_Public_Interfaces.html#s-message-public-domain
[2] http://www.globus.org/toolkit/docs/4.0/security/authzframe/security_descriptor.html
[3] http://www.globus.org/api/javadoc-4.0.0/globus_java_ws_core
[4] http://www.ibm.com/developerworks/library/ws-trust/
[5] http://www-106.ibm.com/developerworks/library/ws-secon/

2.1.2. Operations

- `RequestSecurityToken`: This operation initiates a new security session negotiation. Furthermore, since the actual schema for this message is not unambiguously defined by the specifications, this is the actual schema used:

```
<xs:element name='RequestSecurityToken'>
   <xs:complexType name='RequestSecurityTokenType'>
      <xs:sequence>
         <xs:element ref='wst:TokenType'/>
         <xs:element ref='wst:RequestType'/>
         <xs:element ref='wst:BinaryExchange'/>
      </xs:sequence>
      <xs:attribute name='Context' type='xs:anyURI'/>
   </xs:complexType>
</xs:element>

<xs:element name='RequestSecurityTokenResponse'>
   <xs:complexType name='RequestSecurityTokenResponseType'>
      <xs:sequence>
         <xs:element ref='wst:TokenType'/>
         <xs:element ref='wst:RequestType'/>
         <xs:element ref='wst:BinaryExchange'/>
      </xs:sequence>
      <xs:attribute name='Context' type='xs:anyURI'/>
   </xs:complexType>
</xs:element>
```

- `RequestSecurityTokenResponse`: This operation continues a security session negotiation. Furthermore, since the actual schema for this message is not unambiguously defined by the specifications, this is the actual schema used:

```
<xs:element name='RequestSecurityTokenResponse'>
   <xs:complexType name='RequestSecurityTokenResponseType'>
      <xs:sequence>
         <xs:element ref='wst:TokenType'/>
         <xs:element ref='wst:RequestType'/>
         <xs:element ref='wst:BinaryExchange'/>
      </xs:sequence>
      <xs:attribute name='Context' type='xs:anyURI'/>
   </xs:complexType>
</xs:element>

<xs:element name='RequestSecurityTokenResponse'>
   <xs:complexType name='RequestSecurityTokenResponseType'>
      <xs:sequence>
         <xs:element ref='wst:TokenType'/>
         <xs:element ref='wst:RequestType'/>
         <xs:element ref='wst:BinaryExchange'
                     minOccurs="0"/>
         <xs:element ref='wsc:SecurityContextToken'/>
      </xs:sequence>
      <xs:attribute name='Context' type='xs:anyURI'/>
   </xs:complexType>
</xs:element>
```

In the above the second `RequestSecurityTokenResponse` element refers to the final message in the exchange.

2.1.3. Resource properties

This service has no associated resource properties

2.1.4. Faults

Both `RequestSecurityToken` and `RequestSecurityTokenResponse` throw the following faults:

- `ValueTypeNotSupportedFault`: This fault indicates that the value type attribute on the binary exchange token element is not supported by the service.
- `EncodingTypeNotSupportedFault`: This fault indicates that the encoding type attribute on the binary exchange token element is not supported by the service.
- `RequestTypeNotSupportedFault`: This fault indicates that the request type specified in the request type element is not supported by the service
- `TokenTypeNotSupportedFault`: This fault indicates that the token type specified in the token type element is not supported by the service.
- `MalformedMessageFault`: This fault indicates that the message content received by the service does not conform to the expected content. This is necessary since the schema does not give a well defined content model.
- `BinaryExchangeFault`: This fault indicates that a failure occurred during the in the underlying security constant responsible for the session negotiation.
- `InvalidContextIdFault`: This fault indicates that the context id passed in the message is not valid within the context of this service or negotiation.

2.1.5. WSDL and Schema Definitions

- WS-Trust WSDL[6]
- WS-Trust XSD[7]
- WS-SecureConversation XSD[8]
- Secure Conversation WSDL [link]

3. Framework-level Protocols

3.1. WS-Security

The framework implements the Web Services Security: SOAP Message Security[9], Web Services Security: Username Token Profile[10] and Web Services Security: X.509 Token Profile[11] specifications.

[6] http://www-106.ibm.com/developerworks/library/specification/ws-trust/ws-trust.wsdl
[7] http://www-106.ibm.com/developerworks/library/specification/ws-trust/ws-trust.xsd
[8] http://www-106.ibm.com/developerworks/library/specification/ws-secon/ws-secureconversation.xsd
[9] http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf
[10] http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0.pdf
[11] http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0.pdf

3.2. Transport (HTTPS) Security

The transport security solution used by the framework consists of HTTP over SSL/TLS (HTTPS) using X.509 certificates. The path validation step has been augmented to support the Proxy Certificate Profile (RFC3820[12]).

4. Command-line tools

This component has no command line tools.

5. Overview of Graphical User Interfaces

This component has no associated GUIs.

6. Semantics and syntax of domain-specific interface

6.1. Interface introduction

Client side security is set up by either setting individual properties on the javax.xml.rpc.Stub object used for the web service method invocation or by setting properties on a client side security descriptor object, which in turn is propagated to client side security handlers by making it available as a stub object property. Here are examples for the two approaches:

- Setting property on the stub:

```
// Create endpoint reference
EndpointReferenceType endpoint = new EndpointReferenceType();
// Set address of service
String counterAddr =
   "http://localhost:8080/wsrf/services/CounterService";
// Get handle to port
CounterPortType port =
   locator.getCounterPortTypePort(endpoint);
// set client authorization to self ((Stub)port)._setProperty(Constants.AUTHORIZATION, S
```

- Setting properties using a client descriptor:

```
// Client security descriptor file String CLIENT_DESC = "org/globus/wsrf/samples/counter
EndpointReferenceType endpoint = new EndpointReferenceType();
// Set address of service
String counterAddr =
   "http://localhost:8080/wsrf/services/CounterService";
// Get handle to port
CounterPortType port =
   locator.getCounterPortTypePort(endpoint);
//Set descriptor on Stub ((Stub)port)._setProperty(Constants.CLIENT_DESCRIPTOR_FILE, CLI
```

[12] ftp://ftp.rfc-editor.org/in-notes/rfc3820.txt

The descriptor file is defined by the following Client Security Descriptor Schema[13].

Note

If the client needs to use transport security, the following API must be used to register the Axis transport handler for "https":

```
import org.globus.axis.util.Util;
static {
        Util.registerTransport();
    }
```

[13] http://www.globus.org/toolkit/docs/4.0/security/authzframe/client_security_descriptor.xsd

6.2. Syntax of the interface

Table 11.1. Client side security properties

#	Feature	Description	Stub Configuration

1	WS-Security Username/Password	Enable WS-Security username/password authentication.	Properties:
			`org.globus.wsrf.security.Constants.USERNAME`
			Value equals the username.
			`org.globus.wsrf.security.Constants.PASSWORD`
			Value equals the password.

2	GSI Transport	Enable GSI Transport with specified message protection level.	Property: `org.globus.gsi.GSIConstants.GSI_TRANSPORT` Values equal one of the following: • `Constants.ENCRYPTION` • `Constants.SIGNATURE`
3	GSI Secure Message	Enable GSI Secure Message with specified message protection level.	Property: `org.globus.wsrf.security.Constants.GSI_SEC_MSG` Values equal one of the following: • `Constants.ENCRYPTION` • `Constants.SIGNATURE` You can set the SOAP Actor of the signed message using the `x509Actor` property, but we do **not** recommend this unless you know what you are doing.

#	Feature	Description	Stub Configuration
4	GSI Secure Conversation	Enable GSI Secure Conversation with specified message protection level.	Property: `org.globus.wsrf.security.Constants.GSI_SEC_CONV` Values equal one of the following: • Constants.ENCRYPTION • Constants.SIGNATURE Furthermore, you can set the SOAP Actor of the GSI signed/encrypted SOAP message by using the `gssActor` property. We recommend that you **not** do this unless you *really* know what you are doing.
5	Credentials	Allows for setting credentials for authentication. If not specified, the default user proxy is used. Please see the Security Library Compatibility Document[14] for some hints on loading and managing different GSI credentials.	Property: `org.globus.axis.gsi.GSIConstants.GSI_CREDENTIALS` Value equals the Instance of `org.ietf.jgss.GSSCredential`.

[14] http://www.globus.org/cog/distribution/1.1/compatibility.html

#	Feature	Description	Stub Configuration
6	Delegation	Sets the GSI delegation mode. *Used for GSI Secure Conversation only.* If limited or full delegation is chosen, then some form of client side authorization needs to be done (i.e client side authorization cannot be set to none).	Property: `org.globus.axis.gsi.GSIConstants.GSI_MODE` Value equals one of following: 1. `GSIConstants.GSI_MODE_NO_DELEG`: No delegation is performed. 2. `GSIConstants.GSI_MODE_LIMITED_DELEG`: Limited delegation is performed 3. `GSIConstants.GSI_MODE_FULL_DELEG`: Full delegation is performed.
7	Anonymous authentication	Enables anonymous authentication. *This option only applies to GSI Secure Conversation and GSI Transport.*	Property: `org.globus.wsrf.security.Constants.GSI_ANONYMOUS` Value equals one of following: 1. Boolean.FALSE: Anonymous authentication is disabled. 2. Boolean.TRUE: Anonymous authentication is enabled.

#	Feature	Description	Stub Configuration

8	Peer credentials	Sets the credential that is used to encrypt the message (typically, the recipient's *public key*). *Used for GSI Secure Message only.*	Property:
			`org.globus.wsrf.impl.security.authentication.Constants.PEER_SUBJECT`
			
			Value equals the instance of `javax.security.auth.Subject`.
			
			The credential object needs to be wrapped in `org.globus.wsrf.impl.security.authentication.encryption` and added to the set of public credentials of the `Subject` object.
			
			For example, if `publicKeyFilename` was the file that had the recipient's public key:
			
			`Subject subject = new Subject();`
			`X509Certificate serverCert = CertUtil.loadCertificate(publicKeyFilename);`
			`EncryptionCredentials encryptionCreds = new EncryptionCredentials(`
			`new X509Certificate[] { serverCert });`
			`subject.getPublicCredentials().add(encryptionCreds);`
			`stub._setProperty(Constants.PEER_SUBJECT, subject);`

7. Configuration interface

7.1. Configuration overview

Configuration of service-side security settings can be achieved in two ways. The preferred way is to provide these settings in a security descriptor, although it is also possible to manipulate security settings via CoG properties.

7.2. Syntax of the interface

Information on the syntax of security descriptors can be found here[15].

Information on available CoG security properties can be found here[16].

8. Environment variable interface

- X509_USER_PROXY <path>, where <path> is the path of the *proxy credential*.
- X509_CERT_DIR <path>, where <path> is the path to a directory containing trusted, that is CA, certificates.

Note that the above environment variable does not supersede any settings provided in security descriptors.

[15] http://www.globus.org/toolkit/docs/4.0/security/authzframe/security_descriptor.html
[16] http://www.globus.org/cog/distribution/1.2/FAQ.TXT

Chapter 12. GT4 Message & Level Security Quality Profile

1. Test coverage reports

- Clover report[1]

2. Code analysis reports

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

3. Outstanding bugs

- Bug 2362:[4] location of user proxy for java inconsistencies
- Bug 2445:[5] Holder problem
- Bug 2651:[6] /dev/random vs. /dev/urandom
- Bug 2743:[7] grid-mapfile location should be in global security descriptor

4. Bug Fixes

- Bug 2178:[8] Any SOAP headers used for dispatching need to be secured
- Bug 2179:[9] Fix up replay attack prevention
- Bug 2193:[10] No local subject error
- Bug 2207:[11] Missing security error 'timestampNotOk'
- Bug 2371:[12] Better error reporting for security descriptor parsing errors
- Bug 2523:[13] Core should cause error/warning for bad containerSecDesc file

[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/bugzilla/show_bug.cgi?id=2362
[5] http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=2445
[6] http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=2651
[7] http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=2743
[8] http://bugzilla.globus.org/globus/show_bug.cgi?id=2178
[9] http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=2179
[10] http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=2193
[11] http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=2207
[12] http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=2371
[13] http://bugzilla.globus.org/bugzilla/show_bug.cgi?id=2523

5. Performance reports

Secure access of WSRF and WSN operations using GSI Transport and GSI Secure Message have been measured. Test reports are here[14].

[14] ../../common/javawscore/java_ws_core_wsrf_perf.html

Chapter 13. GT 4.0 Migration Guide for WS A &A Message and Transport Level Security

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

1. Migrating from GT2

1.1. Host credentials

GT2 and GT3 services were set up to run with root owned *host credentials*. In GT4 most, but not all, services will run as the globus user. To allow the globus user to start services using host credentials the globus user needs to be able to access them. This requirement can be satisfied by making a copy of the root owned host credentials, i.e. the *host certificate* and *private key*, owned by the globus user. In GT4 this copy is assumed to be `/etc/grid-security/container{cert,key}.pem`.

2. Migrating from GT3

2.1. Host credentials

The information from Migrating from GT2 applies to migrating from GT3 as well.

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