GT 4.2.0 Java WS A&A Developer's Guide

DRAFT

GT 4.2.0 Java WS A&A Developer's Guide

Introduction

fixme

DRAFT

Table of Contents

1. Overview 1
1. Authentication overview 1
2. Authorization framework overview 1
2. Before you begin 2
1. Feature Summary 2
2. Tested platforms 3
3. Backward compatibility summary 3
4. Technology dependencies 3
5. Security considerations for Java WS A&A 4
3. Usage scenarios 6
1. Delegation 6
2. Embedding Key Information in EPRs 6
3. Obtaining peer credentials on the server side 6
4. Obtaining peer credentials from message context on the client side 7
5. Using SAML Authorization Assertions 8
6. Using Multiple Message Protection Schemes 9
4. Tutorials 10
5. Architecture and design overview 11
1. Authentication/message-level architecture 11
2. Authorization architecture 15
6. APIs 17
1. Authorization Programming Model 17
2. Authentication/message protection Programming Model 17
3. API 17
7. Services and WSDL 19
1. Secure Conversation Service 19
2. SAML Authorization Callout 21
8. Framework-level Protocols 23
1. WS-Security 23
2. Transport (HTTPS) Security 23
9. Configuring client authentication and message/transport security 24
1. Interface introduction 24
2. Syntax of the interface 26
10. Authorization domain-level interface 31
1. Interface introduction 31
2. Syntax of the interface 31
11. Configuring 36
1. Configuring authorization 36
2. Configuring authentication/message protection 37
12. Environment variable interface 38
1. Environmental variables for WS Authentication & Authorization (Java) 38
13. PDP Reference 39
1. Introduction 39
2. Access Control List PDP 39
3. GridMapAuthorization 40
4. Host Authorization 41
5. IdentityAuthorization 42
6. No Authorization 42
7. ResourceProperties Authorization 43
8. SAML Authorization Callout 44
9. SAML Authorization Assertion PDP 46

DRAFT

10. Self Authorization 46
11. Username Authorization 47
14. PIP Reference 51
1. Introduction 51
2. Container PIP 51
3. X509Bootstrap 53
4. SAML Authorization Assertion PIP 54
5. Parameter PIP 55
15. Debugging 57
1. Debugging authorization 57
16. Troubleshooting 59
1. Error Messages For Java WS A&A 60
17. Related Documentation 63
Glossary 64

DRAFT

List of Figures

5.1. The new certificate is signed by the owner, rather than a CA. 12
5.2. JAX-RPC handlers involved in security related message processing on a server. 14

DRAFT

List of Tables

9.1. Client side security properties 27
11.1. Configuring server side authentication and message/transport security 37
14.1. Attribute I 52
14.2. Attribute II 52
14.3. Attribute III 52
14.4. Attribute I 52
14.5. Attribute II 53
14.6. Attribute III 53
14.7. Attribute I 54
14.8. Attribute II 54
14.9. Attribute I 55
14.10. Attribute II 55
14.11. Attribute I 56
16.1. Java WS A&A Errors 61

DRAFT

Chapter 1. Overview

1. Authentication overview

Java WS authentication 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. Authorization framework overview

The authorization framework enforces the configured authorization policy on the service and client side.

On the service 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.

On the client side, a pluggable framework for authorization of service is provided.

DRAFT

Chapter 2. Before you begin

1. Feature Summary

1.1. Authentication/message protection features

Features new in GT 4.2.0

None.

Other Supported Features

- Compliance with published IBM/Microsoft WS-Trust and WS-SecureConversation specifications
- Compliance with the Web Services Security 1.0 standard
- HTTPS support
- Message encryption, integrity protection and replay attack prevention
- Establishment of a session key for light-weight message protection

Deprecated Features

- None.

1.2. Authorization features

Features new in GT 4.2.0:

- *Enhanced server-side attributed-based authorization framework:* The server-side authorization framework has been reworked to support attribute based authorization with delegation of rights. The framework allows for configuring a chain of Policy Information Points(PIPs) and Policy Decision Points(PDPs) and a combining alogorithm that processes the individual decisions returned by the PDPs. Some of the key changes from the previous versions are:
  - Java Server side authorization framework has been moved to an independent module. Refer to Changes Summary for details.
  - Authorization framework uses a set of attributes to identify entities
  - The authorization engine uses Java Security provider framework to allow different combining algorithms to be plugged in.
  - A default implementation of permit override combining algorithm, which looks for a permit decision chain, to allow for fine grained delegation of rights.

  Refer Chapter 5, Architecture and design overview for detailed information on the architecture.
- *Host or Self Authoriation:* Support for a pluggable PDP that does host authorization, and if that fails, tries self authorization.

- The security descriptor framework, used to configure security properties for the security framework has been enhanced. Detailed information about the framework is provided Java WS A&A Security Descriptor Framework.

Other Supported Features

- Authorization based on `grid-mapfile` and other access control lists.
- Ability to implement custom authorization modules.
- A SAML callout authorization module enables outsourcing of authorization decisions to an authorization service (e.g. PERMIS).

Deprecated Features

- None

2. Tested platforms

Java WS A&A should work on any platform that supports J2SE 1.3.1 or higher.

Tested Platforms for Java WS A&A:

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

3. Backward compatibility summary

3.1. Authentication compatibility

Since GT 4.0.x release, some incompatible changes have been made:

- Security Descriptors: The security descriptor schema has changed since GT 4.0.x and the descriptors from GT 4.0.x cannot be used as is.
- Secure Conversation port type: The WS Addressing version in Java WS Core has been updated and the secure conversation port type has changed to reflect this. Therefore, GT 4.0.x secure conversation clients are incompatible with GT 4.2.x servers and vice versa.

3.2. Authorization compatibility

The authorization framework has been reworked as described in Change Summary. The configuration and authorization interfaces have since changed and a Migration Guide is provided.

4. Technology dependencies

Java WS A&A depends on the following GT components:

- Java WS Core.

Authentication and message-protection depends on the following 3rd party software:

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

The authorization framework depends on the following 3rd party software:

- OpenSAML

5. Security considerations for Java WS A&A

5.1. Security considerations for authorization

5.1.1. 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*.

5.1.2. 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.

5.2. Security considerations for Message/Transport-level Security

5.2.1. 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

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

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 in Configuring GSI).

Also refer to Section 5, “Known Problems” for details on any other open issues.

DRAFT

Chapter 3. Usage scenarios

1. Delegation

There are two ways a client can delegate its credential to a service:

- using Delegation Service, and
- using GSI Secure Conversation.

A client can delegate using the Delegation Service. This method is independent of the security scheme used and can be reused across multiple invocations of the client to multiple services (provided the services are in the same hosting environment 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 the Stub or using security descriptors as described in Section 1.2.3, “Configuring GSI Secure Conversation ”. 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, the client should have delegated the credentials. Configuring this option is described in Section 1.6.2, “Configuring the run-as mode”. 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();
```

2. Embedding Key Information in EPRs

GT provides an API to embed key information in an Endpoint Reference, as defined in the OGSA Basic Security Profile. The key information is embedded in the extensibility element of the EPR rather than the meta-data element as defined in the specification, since the toolkit uses older version of the WS Addressing specification.

This information would be useful to ascertain the expected identity of the service for authorizing the service or to get the public certificate of the resource to be used for encrypting the request to the service. The optional `usage` element in the emebedded key information indicates the use of the embedded keys, either for signature or encryption.

The API is in class `org.globus.wsrf.impl.security.util.EPRUtil`. The method to embed the certificates is called `insertCertificates` and the method to extract the key information is called `extractCertificates`. Please refer to API documentation for details on using the methods.

3. Obtaining peer credentials on the server side

The security handlers populate a Subject object with peer information. The following code can be used to access the peer credentials. Note that the message context needs to be associated with the thread.

```
import org.globus.wsrf.security.SecurityManager;
```

```
import javax.security.auth.Subject;

org.apache.axis.MessageContext mctx =
org.apache.axis.MessageContext.getCurrentContext();
SecurityManager manager = SecurityManager.getManager(mctx);
Subject subject = manager.getPeerSubject();
```

The following code snippet shows how the certificate chain can be extracted from the peer subject:

```
java.util.Set set =
subject.getPublicCredentials(X509Certificate[].class);
Iterator iterator = set.iterator();
while (iterator.hasNext()) {
X509Certificate[] certArray = (X509Certificate[]) iterator.next();
System.out.println("Cert array " + certArray.length);
}
```

To obtain the peer principal (for example, the Distinguished Name from X509 Certificate), the following code snippet can be used:

```
org.apache.axis.MessageContext mctx =
org.apache.axis.MessageContext.getCurrentContext();
SecurityManager manager = SecurityManager.getManager(mctx);
Principal principal = manager.getCallerPrincipal();
String caller = manager.getCaller();
```

If credentials were delegated as described above, private credentials are also populated:

```
java.util.Set set = subject.getPrivateCredentials();
```

4. Obtaining peer credentials from message context on the client side

- **GSI Secure Conversation:** With this mechanism, the peer credentials can be obtained once the handshake is completed:

```
import org.globus.wsrf.impl.security.authentication.Constants;
import org.globus.wsrf.impl.security.authentication.secureconv.service.S
import org.ietf.jgss.GSSContext;
import org.globus.gsi.gssapi.GSSContants;

// Get current secure context from message context
SecurityContext secContext =
messageContext.getProperty(Constants.CONTEXT);
GSSContext gssContext = secContext.getContext();
```

```
Vector peerCerts =
gssContext.inquireByOid(GSSContants.X509_CERT_CHAIN);
```

- **GSI Secure Transport:** With this mechanism, the peer credentials can be obtained once the handshake is completed:

```
import org.ietf.jgss.GSSContext;
import org.globus.gsi.gssapi.GSSContants;
import org.globus.wsrf.impl.security.authentication.Constants;

// Get current secure context from message context
GSSContext gssContext =
messageContext.getProperty(Constants.TRANSPORT_SECURITY_CONTEXT);
Vector peerCerts =
gssContext.inquireByOid(GSSContants.X509_CERT_CHAIN);
```

- **GSI Secure Message:** With this mechanism, the peer credentials can be obtained only when the response is received:

```
import org.globus.wsrf.impl.security.authentication.Constants;

// Get peer subject from current message context
Subject subject =
(Subject) messageCtx.getProperty(Constants.PEER_SUBJECT);
Set peerCerts =
subject.getPublicCredentials(X509Certificate[].class);
```

5. Using SAML Authorization Assertions

SAML Authorization assertions can be used to transport authorization decision statements to the point of enforcement. Such assertions can be obtained from any entity that manages use rights. For example, Community Authorization Service (CAS), a Globus Toolikit higher level service, can be used to manage user rights and issue such assertions. On the remote side, PDPs/PIPs might be used to extract the assertion and enforce the rights. Some PDPs/PIPs with such functionality are distributed and details can be found in Chapter 14, PIP Reference and Chapter 13, PDP Reference.

The toolkit provides two mechanisms to do this:

- **Using proxy certificates:** In this mechanism the SAML Authorization Assertions are embedded as non-critical assertions in the proxy certificate. These assertions can then be extracted from the certificate, to enforce access rights at the resource.

  A command line client `$GLOBUS_LOCATION/bin/globus-embed-assertion` implemented using `org.globus.wsrf.client.EmbedAssertion` can be used to embed a SAML Assertion stored in a file into a credential.

- **Using SOAP Header:** The SAML Assertion can also be embedded in the SOAP Header. The toolkit uses the WS-Security SAML Token Profile to embed the assertion.

Section 2, “Syntax of the interface” describes the configuration required to embded an assertion into the message header. At the remote end, the security handlers detect the presence of the SAML Assertion in the header and store it as a string in the Message Context property `Constants.SAML_ASSERTION_STR.`

6. Using Multiple Message Protection Schemes

Multiple message protection schemes can be used in a single invocation, although it is worth noting that this will cause a performance penalty.

For example, both Secure Transport and Secure Conversation can be done on the same invocation by using the following:

```
stub._setProperty(Constants.GSI_SEC_CONV, Constants.INTEGRITY);
stub._setProperty(Constants.GSI_TRANSPORT, Constants.PRIVACY);
```

Note

These two mechanisms share a single property for authorization. There is a bug open to provide independent support: Bug 4350[1]

Similarly Secure Messages can be used in tandem with other message protection mechanisms.

[1] http://bugzilla.mcs.anl.gov/globus/show_bug.cgi?id=4350

Chapter 4. Tutorials

There are no tutorials available at this time.

DRAFT

Chapter 5. Architecture and design overview

1. Authentication/message-level architecture

1.1. 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 part of the CoG library. The GT4 Java WS code base and configuration can be modified to use the HTTPG protocol as 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 an 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 the handshake.

1.1.1. 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.

1.1.2. Client-Side Security

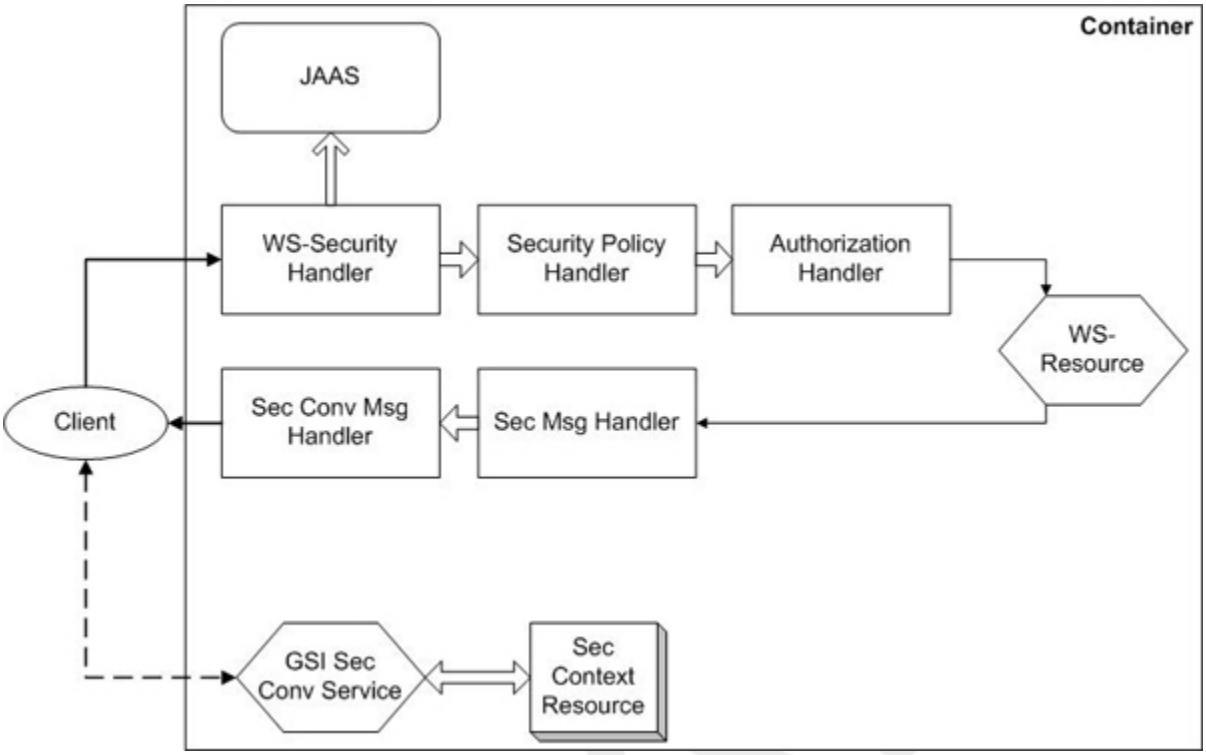
On the client-side, transport security is similarly enabled by switching a non-secure socket implementation with the *GSISocket* implementation and registering a protocol handler for HTTPS that uses the secure socket implementation. In practice, this means that any messages targeted at an HTTPS endpoint will, irregardless of any stub properties, be authenticated and protected. It also means that any messages sent to an 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.

1.2. Message Level Security

1.2.1. 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.

Figure 5.1. The new certificate is signed by the owner, rather than a CA.



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.

1.2.2. Message Processing

When a message arrives from the client, the SOAP engine invokes several security-related handlers:

1. 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.

2. 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.

3. 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 the authorization framework documentation.

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.

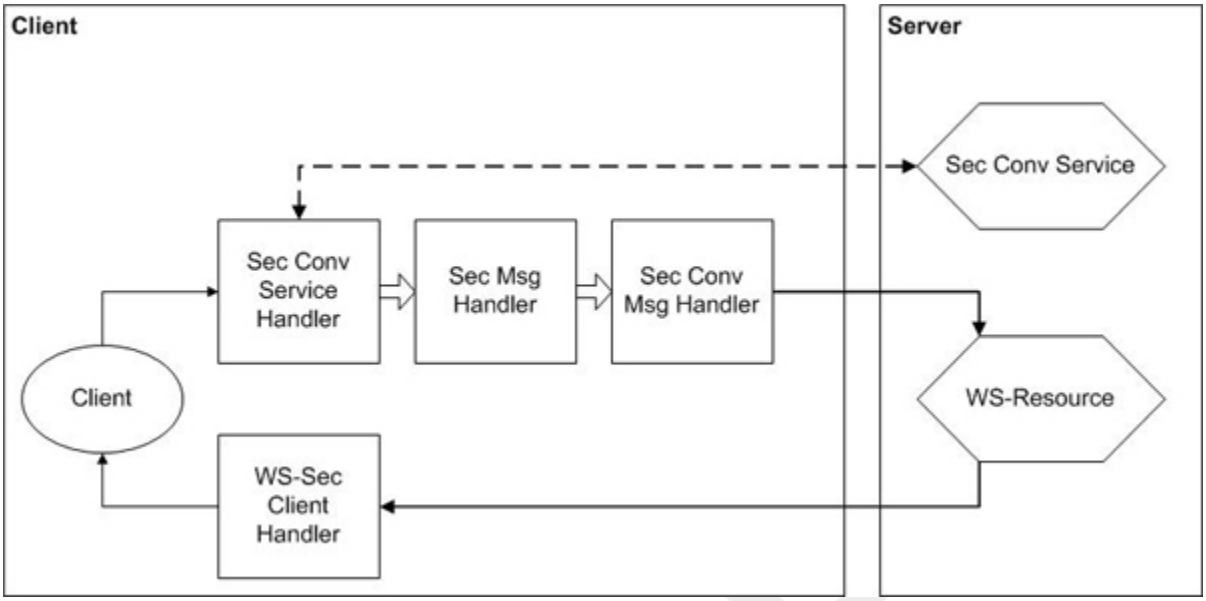
The operations that are actually performed depend on the message properties associated with the message by the inbound handlers, i.e. outbound messages will have the same security attributes as inbound messages. That being said, a service has the option of modifying the message properties, if so desired. These handlers are identical to the client-side handlers described in the following section.

1.2.3. 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 must 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.

DRAFT

Figure 5.2. JAX-RPC handlers involved in security related message processing on a server.



1.2.4. 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:

1. 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.

2. 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.

3. The third outbound handler [fixme - is there a name?] 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 the GSI Secure Message operation, it will also authorize the remote side in a similar fashion to the outbound secure conversation service handler.

2. Authorization architecture

2.1. Server-side authorization

The Java WS Authorization framework leverages the generic GT Java Authorization Framework. The framework 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 framework enables attribute-based authorization. PIPs can be used to collect attributes about resource/operations/subjects and used in the decision making process. While the toolkit provides some implementations of PIPs/PDPs, the framework is pluggable and custom mechanisms can be written and configured.

An authorization engine consists of PIPs, PDPs and a combining algorithm. The configured 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 architecture of Generic Java Authorization Engine is described in detail in this document[1]. It also describes interfaces and writing custom PDPs/PIPs.

Any PDP has to implement the interface `org.globus.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.security.authorization.ChainConfig` 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.security.authorization.PIP` with the functionality to collect attributes from the invocation context that are relevant to making the authorization decision.

2.1.1. Authorization Policy Configuration

A chain of PDPs and PIPs, with relevant configuration information, can be configured at resource, service or container level. These chain use Permit Override with Delegation as default combining algorithm. Additionally an administrative policy can be configured at the container level. The administrative chain uses First Applicable combining algorithm by default. Note that comining algorithms can be configured to over-ride the deafult. The following describes the precedence in which configured policy is used:

1. If container level administrative policy is specified, it is evaluated.
    a. If (1) returns a deny, the request is denied.
    b. If (1) returns a permit, step (2) is done.
2. If resource level policy is specified, it is evaluated.
    a. If (2) returns a deny, the request is denied.
    b. If (2) returns a permit, the request is permitted.
3. If (2) is not specified and service level policy is specified, it is evaluated.

[1] ../gtJavaAuthzEngine.pdf

   a. If (3) returns a deny, the request is denied.

   b. If (3) returns a permit, the request is permitted.

4. If (3) is not specified and container level policy configuration is specified, it is evalauted.

   a. If (4) returns a deny, the request is denied.

   b. If (4) returns a permit, the request is permitted.

2.1.2. Authorization Handler Steps

1. Invoke Container PIP to collect attributes inherent to the framework. The PIP creates an instance of RequestEntities class to use as parameter with PIPs. It also creates an instance of ResourceChainConfig class to push the current message context as a parameter to ContianerPIP.

2. Evaluate the administrator authorization engine, if one is configured

   a. If bootstrap overwrite is configured, then only BootstrapPIPs in administrator engine is invoked. Else the X509 Bootstrap PIP is invoked prior to any other Bootstrap PIPs configured.

   b. The authorization engine is run and if a deny decision is returned, the operation is denied. If a permit decision is returned, the operation is permitted. If a not applicable or indeterminate is returned, further authorization engines are evaluated.

3. Evaluate the authorization engine configured in the resource, service, container, in that order depending on which is configured

   a. If bootstrap overwrite is configured, then only BootstrapPIPs in administrator engine is invoked. Else the X509 Bootstrap PIP is invoked prior to any other Bootstrap PIPs configured.

   b. If any decision other than a Permit is returned, the operation is denied. If a permit is returned the operation is allowed.

4. If no authorization engine was configured, then default authorization engine is created, which checks whether the caller has same credentials as service (self authorization)

2.2. 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. This is done as a part of the web services client handler. The other two authentication schemes supported, GSI Secure Conversation and GSI Transport, authorize the server during the handshake, prior to the operation invocation.

The Transport Level Security protocol allows for authorization (an expected DN comparison) during the handshake. This is disabled by default in the toolkit, unless delegation of credential is requested. If no delegation is requested, the configured authorization mechanism is invoked afer the handshake is complete, prior to the operation invocation. If delegation is requested, authorization (expected DN comparison) is done during key exchange as a part of the protocol.

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.

Chapter 6. APIs

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

1. Authorization Programming Model

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. Alternatively the security settings can be done using programmatic security descriptor constructs. 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.

2. Authentication/message protection Programming Model

The authentication 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 Configuring client authentication and message/transport security. More information about the security descriptor can be found in Java WS A&A Security Descriptor Framework.

3. API

- Generic Java Authorization Engine API
  - org.globus.security.authorization.PDP
  - org.globus.security.authorization.PIP
  - org.globus.security.authorization.ChainConfig
  - org.globus.security.authorization.Interceptor
- Stable API
  - org.globus.wsrf.security.Constants
  - org.globus.wsrf.security.SecureResource
  - org.globus.wsrf.security.SecurityManager
  - org.globus.wsrf.security.authorization.Constants
  - org.globus.wsrf.security.impl.authorization.Authorization
- Less stable API

[1] http://www.globus.org/api/javadoc-4.2.0/globus_java_ws_core

- org.globus.wsrf.impl.security.descriptor.ClientSecurityDescriptor
- org.globus.wsrf.impl.security.descriptor.ServiceSecurityDescriptor
- org.globus.wsrf.impl.security.descriptor.ResourceSecurityDescriptor

DRAFT

Chapter 7. Services and WSDL

1. Secure Conversation Service

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[1] and WS-SecureConversation[2] specifications.

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'/>
```

[1] http://www.ibm.com/developerworks/library/ws-trust/
[2] http://www-106.ibm.com/developerworks/library/ws-secon/

```
   </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 schema, the second `RequestSecurityTokenResponse` element refers to the final message in the exchange.

1.3. Resource properties

This service has no associated resource properties.

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.

1.5. WSDL and Schema Definitions

- WS-Trust WSDL[3]

[3] http://www-106.ibm.com/developerworks/library/specification/ws-trust/ws-trust.wsdl

- WS-Trust XSD[4]
- WS-SecureConversation XSD[5]
- Secure Conversation WSDL [fixme - link]

2. SAML Authorization Callout

The authorization framework as such does not have a WSDL interface. On the other hand one of the authorization schemes in the toolkit, a callout to an authorization service compliant with the specification published by the OGSA Authorization Working Group (OGSA-AuthZ)[6] requires a WSDL interface for the service. The callout makes a query on the configured authorization service, which returns an authorization decision.

2.1. Protocol overview

The authorization service takes a query as input and returns an authorization decision. The Security Assertion Markup Language (SAML)[7] 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.

2.2. 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
  - InvalidResourcePropertyQNameFault
- `SetResourceProperties`: Sets the value for resource properties. This operation throws the following faults:
  - ResourceUnknownFault
  - InvalidSetResourcePropertiesRequestContentFault
  - UnableToModifyResourcePropertyFault
  - InvalidResourcePropertyQNameFault
  - SetResourcePropertyRequestFailedFault
- `QueryResourceProperties`: Used for the querying of resource properties using a query expression. This operation throws the following faults:
  - ResourceUnknownFault
  - InvalidResourcePropertyQNameFault

[4] http://www-106.ibm.com/developerworks/library/specification/ws-trust/ws-trust.xsd
[5] http://www-106.ibm.com/developerworks/library/specification/ws-secon/ws-secureconversation.xsd
[6] https://forge.gridforum.org/projects/ogsa-authz
[7] http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security

- UnknownQueryExpressionDialectFault
- InvalidQueryExpressionFault
- QueryEvaluationErrorFault

2.3. 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.

2.4. Faults

- ResourceUnknownFault[8]
- InvalidSetResourcePropertiesRequestContentFault[9]
- UnableToModifyResourcePropertyFault[10]
- InvalidResourcePropertyQNameFault[11]
- SetResourcePropertyRequestFailedFault[12]
- UnknownQueryExpressionDialectFault[13]
- InvalidQueryExpressionFault[14]
- QueryEvaluationErrorFault[15]

2.5. WSDL and Schema Definition

- OGSA-AuthZ Authorization Service WSDL [16]

[8] http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceProperties-1.2-draft-04.pdf
[9] http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceProperties-1.2-draft-04.pdf
[10] http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceProperties-1.2-draft-04.pdf
[11] http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceProperties-1.2-draft-04.pdf
[12] http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceProperties-1.2-draft-04.pdf
[13] http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceProperties-1.2-draft-04.pdf
[14] http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceProperties-1.2-draft-04.pdf
[15] http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceProperties-1.2-draft-04.pdf
[16] http://viewcvs.globus.org/viewcvs.cgi/wsrf/schema/core/security/authorization/authz_port_type.wsdl?rev=1.9&only_with_tag=globus_4_2_0&content-type=text/vnd.viewcvs-markup

Chapter 8. Framework-level Protocols

1. WS-Security

The framework implements the Web Services Security: SOAP Message Security[1], Web Services Security: Username Token Profile[2] and Web Services Security: X.509 Token Profile[3] specifications.

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[4]).

[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-username-token-profile-1.0.pdf
[3] http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0.pdf
[4] ftp://ftp.rfc-editor.org/in-notes/rfc3820.txt

Chapter 9. Configuring client authentication and message/transport security

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 of the two approaches:

- Setting a 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,
               SelfAuthorization.getInstance());
```

- Setting properties using a client descriptor:

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

The descriptor file is described in detail in Chapter 1, Security Descriptors Introduction.

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();
    }
```

DRAFT

2. Syntax of the interface

DRAFT

Table 9.1. Client side security properties

Number	Task	Stub Configuration	Des Co tio
1.	Allows for configuration of credentials for authentication.	Property: `org.globus.axis.gsi.GSIConstants.GSI_CREDENTIALS` Value equals the Instance of `org.ietf.jgss.GSSCredential`.	Sec tio "Co ing tial
2.	Allows for configuring client-side authorization.	Property: `org.globus.wsrf.security.Constants.AUTHORIZATION` Value equals the Instance of `org.globus.wsrf.security.authorization.Authorization` If GSI Secure Transport or GSI Secure Conversation is used, the value should be an instance of `org.globus.gsi.gssapi.auth.Authorization`. But this translation is done automatically by the toolkit.	Ref Sec tio "Co ing atio ani

3.	Enable GSI Secure Conversation with specified message protection level.	1. Property:
		`org.globus.wsrf.security.Constants.GSI_SEC_CONV`
		Values equal one of the following:
		• Constants.ENCRYPTION
		• Constants.SIGNATURE
		2. Property:
		`org.globus.wsrf.security.Constants.GSI_SEC_CONV_SECREPLY_UNNECESSARY`
		If the value is set to Boolean.TRUE, the GSI Secure conversation protection is not required in the reply message. By default, if the request was secured with GSI Secure Conversation, the response is also required to have the same protection.
		3. Property:
		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.
			Ref tio "Co ing cur ver

DRAFT

4.	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.	Ref tion "Co ing cur ver
5.	Enables GSI Secure Transport with some protection level.	Property: `org.globus.gsi.GSIConstants.GSI_TRANSPORT` Values equal one of the following: • `Constants.ENCRYPTION` • `Constants.SIGNATURE`	Ref tion "Co ing cur por
6.	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.	Ref tion "Co ing cur ver and tion "Co ing cur por

DRAFT

7.	Enable GSI Secure Message with specified message protection level.	1. Property:
		`org.globus.wsrf.security.Constants.GSI_SEC_MSG`
		Values equal one of the following:
		• `Constants.ENCRYPTION`
		• `Constants.SIGNATURE`
		2. Property:
		`org.globus.wsrf.security.Constants.GSI_SEC_MSG_SECREPLY_UNNECESSARY`
		If the value is set to Boolean.TRUE, the GSI Secure Message protection is not required in the reply message. By default, if the request was secured with GSI Secure Message, the response is also required to have the same protection.
		3. Property:
		`org.globus.wsrf.security.Constants.GSI_SEC_MSG_SINGLECERT`
		If the value is set to Boolean.TRUE, only a single certificate is used for the GSI Secure Message request. By default, the whole certificate chain is sent.
		4. Property:
		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.
			Ref
			tion
			"Co
			ing
			cur
			sag

8.	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.
			Ref
			tion
			"Co
			ing
			nam
			wo

DRAFT

9.	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);`
			Ref tion "Co ing cur sag

10.	Sets the trusted certificates location.	Property: `org.globus.wsrf.security.TRUSTED_CERTIFICATES` Value should be a comma-separated list of directories and file names.	Ref tion "Co ing cre "
11.	Sets the SAML Authorization Assertion to embed in SOAP Header.	Property: `org.globus.wsrf.impl.security.authentication.Constants.SAML_AUTHZ_ASSERTION` Value should be an instance of org.opensaml.SAMLAssertion.	Ca co usi des

DRAFT

Chapter 10. Authorization domain-level interface

1. Interface introduction

Configuration on the server side is done using Chapter 1, Security Descriptors Introduction. Make sure you have read about security descriptors (in the aforementioned link) before continuing with this chapter. Custom authorization mechanisms can be written and used as a part of the GT framework. The next section describes the steps involved.

On the client side, in addition to the security descriptor, properties on the stub can be set to configure security properties. Properties and values are described in detail in the next section.

2. Syntax of the interface

2.1. Configuring client-side authorization on the stub

The property to use depends on the authentication scheme:

- **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.getInstance());
```

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 the value for the appropriate property on the stub.

Note

Security descriptors cannot be used to configure custom authorization schemes on the 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[1]

- **For all authentication schemes other than those in previous step** 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

       // 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:

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

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

Various authorization scheme implementations in package `org.globus.wsrf.impl.security.authorization` would serve as examples. View CVS Link[2].

2.3. Writing a custom server-side authorization mechanism

The server side authorization framework can be configured to use custom authorization interceptors, bootstrap PIP, PIP and PDP. Detailed information on writing custom PDPs can be found in GT Java Authorization Framework[3]. Also, the section Section 1, "Migrating Java WS Authorization Framework from GT 4.0" describes migrating from older PDP/PIP implementations.

For example, a custom PDP must implement the interface `org.globus.security.authorization.PDP`.

Example:

```
package org.foobar;

import ....;

public class FooPDP implements PDP
{
    private Principal authorizedIdentity;


    public Decision canAccess(RequestEntity requestEntity,
                              NonRequestEntity nonRequestEntity)
        throws AuthorizationException {

        // process and return decision
    }

    public Decision canAdminister(RequestEntity requestEntity,
                                  NonRequestEntity nonRequestEntity)
        throws AuthorizationException {

        // process and return decision
    }

}
```

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/>
```

[2] http://viewcvs.globus.org/viewcvs.cgi/wsrf/java/core/source/src/org/globus/wsrf/impl/security/authorization/?pathrev=HEAD
[3] ../gtJavaAuthzEngine.pdf

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

DRAFT

Chapter 11. Configuring

Java WS A&A is configured using security descriptors. The following describes configuration settings specific for authorization and authentication. You can read the entire Java WS A&A Security Descriptor documentation here.

- Configuring authorization
- Configuring authentication/message protection

1. Configuring authorization

1.1. Configuration overview

Security descriptors are mechanisms used to configure authorization mechanism and policy. The authorization on the server side can be configured at the container, service or resource level.

On the client side, authorization can be configured using security descriptors or as a property on the stub. This configuration can be done on a per invocation granularity

1.2. Server side authorization

The server side authorization can be configured at the container, service or resource level using

- Security descriptors using files. Refer to Section 1, “Configuring Using Files”
- Security descriptors programmatically. Refer to Section 2, “Configuring Programmatically ”

To write and configure a server-side custom authorization mechanism refer to Section 2.3, “Writing a custom server-side authorization mechanism”.

1.3. Client side authorization

The client side authorization can be configured for each invocation.

- Security descriptors using files. Refer to Section 1, “Configuring Using Files”, specifically Section 1.2.2, “Configuring authorization mechanism ”.
- Security descriptors programmatically. Refer to Section 2, “Configuring Programmatically ”
- Properties on the Stub. Refer to Section 2.1, “Configuring client-side authorization on the stub”

To write and configure custom authorization mechanism refer to Section 2.2, “Writing custom client-side authorization scheme”.

If no authorization mechanism has been specified, HostOrSelf authorization is used. In this scheme host authorization is tried first, if it fails, self authorization is attempted

2. Configuring authentication/message protection

2.1. Configuration overview

Configuration of service-side security settings can be achieved by using container or service security descriptor. Some of the security configuration, like the credential to use and trusted certificates location, can also be configured using CoG properties or rely on default location. **The preferred way is to provide these settings in a security descriptor.**

The next section provides details on the relevant properties. An overview of the syntax of security descriptors can be found in Java WS A&A Security Descriptor Framework. Available CoG security properties can be found in Chapter 2, Configuring

2.2. Syntax of the interface

The following properties are relevant to authentication and message/transport security:

Table 11.1. Configuring server side authentication and message/transport security

Number	Task	Descriptor Configuration	Alternate Configuration
1	Credentials	Container or service descriptor configuration	• X509_USER_CERT or CoG Configuration: User certificate configuration • X509_USER_KEY or CoG Configuration: User key configuration • X509_USER_PROXY or CoG Configuration: User proxy configuration If no explicit configuration is found, the default proxy is read from `/tmp/x509_up_<uid>`.
2	Trusted Certificates	Container security descriptor configuration	CoG Configuration
3	Limited proxy policy configuration	Container or service descriptor configuration	None.
4	Replay Attack Window	Container or service descriptor configuration	None.
5	Replay Attack Filter	Container or service descriptor configuration	None.
6	Replay timer interval	Container descriptor configuration	None.
7	Context timer interval	Container descriptor configuration	None.

Chapter 12. Environment variable interface

1. Environmental variables for WS Authentication & Authorization (Java)

Refer to Chapter 2, Configuring for environment variables. Note that the above environment variables do *not* supersede any settings provided in security descriptors.

DRAFT

Chapter 13. PDP Reference

1. Introduction

[fixme - what are PDPs?]

If you have a PDP you'd like to contribute to the Globus Toolkit, use the following template:

- PDP template[1]

Note

The above files are in DocBook XML format. Simply click the link, save to your hard drive, edit the file in a text or xml editor and email to ?. Don't worry about getting the tags right, it's enough to enter the information where it makes sense and we'll clean up the tags where necessary.

2. Access Control List PDP

2.1. Class name

`org.globus.wsrf.impl.security.authorizationew.AccessControlListPDP`

2.2. Overview

The PDP uses configured access contol lists to ascertain if a subject can access an operation.

2.3. Configuration

`accessConfigFile` Property to configure the file containing policy for accessing resource. If not configured, file *access-acl.conf* is used by default.

`adminConfigFile` Property to configure file containing policy for administrating the resource. If not configured, file *admin-acl.conf* is used by default.

The PDP looks for the files in the following order:

1. Current directory (.)
2. /etc
3. /etc/grid-security
4. $GLOBUS_LOCATION

The configured files should have the following format:

```
subjectName=serviceName#method1,method2;testService2#method1,method2
```

[1] ../Java_WS_Security_PDP_Template.xml

- Each subject DN should be a seperate line. Equal to signs (=) and spaces should be escaped with backslash (\)
- An equal to must separate the subject DN and the service/operation name description.
- List of service URL and method names separated by semi-colon (;)
- Each service and method description should container service URL followed by hash sign (#) and list of methods the subject DN is allowed to access/admin.
- Method names for each service separated by comma (,).

2.4. Decision Table

Bad configuration	INDETERMINATE
No configuration	INDETERMINATE
No policy for subject	DENY
Anonymous Access	DENY
No policy for subject to access service/operation	DENY
Policy exists for subject to access service/operation	PERMIT

2.5. Related interceptors

No interceptors are related other than the default bootstrap interceptor (X509 Bootstrap PIP), which is required to be used with this PDP.

3. GridMapAuthorization

3.1. Class name

`org.globus.wsrf.impl.security.authorization.GridMapAuthorization`

3.2. Overview

This scheme checks the presence of a user's DN in a configured gridmap file. The grid map file contains a list of mappings from a user's DN to set of local user names that the user is mapped to. The DN and the list of comma-separated mappings are separated by a space. Each mapping is a separate new line by itself.

If the user is present in the configured gridmap file, the mappings are populated in the peer subject object as principals and the user is allowed access to the resource. Absence of DN in gridmap file yields a deny.

3.3. Configuration

The gridmap file location or an instance of GridMap object needs to be configured as a parameter to this PDP. The parameter names are:

`gridmap-file` Property to configure the location of the gridmap file. (From CoG add details on how this file is read in, relative to current directory ?)]

`gridmapObject` Property to configure an instance of GridMap class.

The PDP looks for the gridmap file in the following order:

1. GridMap object property in the configuration.
2. GridMap file property in the configuration.
3. GridMap object in the container default properties.
4. GridMap file property in the container default properties.

A default grid map file can be configured as described in Section 1, “Configuring Default GridMap Files”.

3.4. Decision Table

No peer subject	INDETERMINATE
Bad/No gridmap configuration	INDETERMINATE
Failed girdmap refresh	INDETERMINATE
Anonymous Access	DENY
No gridmap entry	DENY
Gridmap entry present	PERMIT

3.5. Related interceptors

No interceptors are related other than the default bootstrap interceptor (X509 Bootstrap PIP), which is required to be used with this PDP.

4. Host Authorization

4.1. Class name

`org.globus.wsrf.impl.security.authorization.HostAuthorization`

4.2. Overview

PDP performs host-based authorization of the client, i.e expects the client to be running with host credential.

4.3. Configuration

`url` Property that should be configured with the URL of the client.

`service` Property that can be configured with the service, if service credentials are used rather than regular host credentials. By default the value is set to *host*.

4.4. Decision Table

No peer subject	INDETERMINATE
Bad configuration	INDETERMINATE
url property not configured	INDETERMINATE

Anonymous Access	DENY
Peer DN does not match expected DN	DENY
Peer DN matches expected DN	PERMIT

4.5. Related interceptors

No interceptors are related other than the default bootstrap interceptor (X509 Bootstrap PIP), which is required to be used with this PDP.

5. IdentityAuthorization

5.1. Class name

`org.globus.wsrf.impl.security.authorization.IdentityAuthorization`

5.2. Overview

Compares the peer subject with a specific configured subject DN.

5.3. Configuration

`identity` Property that should be configured with the expected peer DN.

5.4. Decision Table

No peer subject	DENY
No local subject	INDETERMINATE
Peer subject does not match configured (expected) subject DN	DENY
Peer subject matches the configured(expected) subject DN	PERMIT

5.5. Related interceptors

No interceptors are related other than the default bootstrap interceptor (X509 Bootstrap PIP), which is required to be used with this PDP.

6. No Authorization

6.1. Class name

`org.globus.wsrf.impl.security.authorization.NoAuthorization`

6.2. Overview

This PDP always returns a permit.

6.3. Configuration

None.

6.4. Decision Table

Always returns a permit	-

6.5. Related interceptors

No interceptors are related

7. ResourceProperties Authorization

7.1. Class name

`org.globus.wsrf.impl.security.authorization.ResourcePropertiesPDP`

7.2. Overview

The PDP enforces a parameter based authorization policy on GetResourceProperty, GetMultipleResourceProperties and SetResourceProperties. QueryResourceProperties is not protected by this PDP and to prevent malicious access of RPs through that method, access to that method must be protected using other schemes. GetMultipleResourceProperies access is allowed only if policy allows user access to all the RPs qeuried.

7.3. Configuration

`get-rp-pdp-config` Property that should be configured with a file containing authorization policy for access to GetResourcePropery and GetMultipleResourceProperties method.

`set-rp-pdp-config` Property that should be configured with a file containing authorization policy for access to SetResourceProperty method.

The format of the configuration files should be as follows:

```
subjectDN=qName1;qName2;qName3
```

All equal to signs and space in Subject DN should be escaped using backslash.

For example, if the set resource properties policy is defined as follows:

```
/C\=US/O\=Globus\ Alliance/OU\=User/CN\=101497d3dcd.3dcd5aef=\
{http://www.globus.org/tests/security}booleanVal;{http://www.globus.org\
/tests/security}intVal1
```

[Above should be a single line without any spaces. Spaces provided here for document formatting] This implies that subject DN "/C=US/O=Globus Alliance/OU=User/CN=101497d3dcd.3dcd5aef" can set value of the RPs {http://www.globus.org/tests/security}booleanVal and {http://www.globus.org/tests/security}intVal1

7.4. Decision Table

Policy for set RP and get RP not configured	Initialize Exception
Erroneous configuration file	INDETERMINATE
Missing parameter value attribute	INDETERMINATE
No policy for user to access requested resource property.	DENY
Policy found for user to access requested resource property..	PERMIT

7.5. Related interceptors

- Default bootstrap interceptor (X509 Bootstrap PIP), is required to use this PDP.
- Parameter PIP (Parameter PIP is required to use with this PDP to be able to collect information about the requested resource property. The ParameterPIP needs to be configured with the following :

  ```
  servicePath getMultipleResourceProperties{http://docs.oasis-open.org\
  /wsrf/2004/06/wsrf-WS-ResourceProperties-1.2-draft-01.xsd}\
  getMultipleResourceProperties

  servicePath getResourceProperty {http://docs.oasis-open.org/wsrf/2004/06/wsrf\
  -WS-ResourceProperties-1.2-draft-01.xsd}getResourceProperty

  servicePath setResourceProperties {http://docs.oasis-open.org/wsrf/2004/06/wsrf\
  -WS-ResourceProperties-1.2-draft-01.xsd}SetResourceProperties
  ```

  [Above should be a single line without any spaces. Spaces provided here for document formatting]

  The servicePath needs to be replaced with the service endpoint that required resource property access to be authorized based on parameters.

8. SAML Authorization Callout

8.1. Class name

`org.globus.wsrf.impl.security.authorization.SAMLAuthorizationCallout`

8.2. Overview

This PDP can be used to talk to any authorization service that implements the OGSA AuthZ[2] interface. The steps involved include:

1. Secure call is made to the authorization service
2. Secure response is expected
3. Identity of soap message siganture or tls connection is established
4. If response is signed, identity of response signature is established.

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

5. Issuer identity is determined from authz service is 4 or 5, in that order
6. If there are any errors constructing request,contacting authz service or parsing response, an indeterminate decision is issued by container
7. Response signature is verified.
8. Every assertion in the response is verified to be issued by issuer identity (point 5)
9. If any of the statement is not a permit on the particular subject, resource, action, a deny is returned by issuer identity. Otherwise a permit is returned.
10. In the future, a delegation step from the issuer identity (in 5) to some other issuers could be set as policy on SAMLAuthzCallout to establish a chain.

8.3. Configuration

`authzService` Property that points to the authorization service to contact. The value should be a URL.

`securityMechanism` Property that indicates the security mechanism to use to contact authorization service. Should be either "msg" for GSI Secure Message or "conv" for GSI Secure Conversation. If authorization service URL container "https" as protocol, GSI Secure Transport is used to contact the authorization service.

`protectionLevel` Property that indicates the protection level to use to contact the authorization service. Should be either "sig" for signature and "enc" for encryption.

`authzServiceCertificate` Property that points to the authorization service public certificate file. This property is required only if "securityMechanism" is GSI Secure Message and "protection-Level" is encryption.

`authzServiceIdentity` Property that has the expected identity of the authorization service. This is used for authorizing the call to authorization service.

`samlAuthzSimpleDecision` Property indicates if SimpleAuthorizationDecisionStatement as defined in OGSA AuthZ specification is being requested from the authorization service.

8.4. Decision Table

Error constructing local SAML data types.	INDETERMINATE
Error converting EPR to string	INDETERMINATE
Error signing SAML Request	INDETERMINATE
Error accessing configured authorization service	INDETERMINATE
Null response from configured authorization service	INDETERMINATE
SAML exception from configured authorization service	INDETERMINATE
SAML Response signed by identity different from expected identity ofconfigured authorization service	INDETERMINATE
If decision returned from service is anything other than permit.	DENY
If decision returned from service is permit.	PERMIT

8.5. Related interceptors

No interceptors are related other than the default bootstrap interceptor (X509 Bootstrap PIP), which is required to be used with this PDP.

9. SAML Authorization Assertion PDP

9.1. Class name

`org.globus.wsrf.impl.security.authorization.SAMLAuthzAssertionPDP`

9.2. Overview

This PDP parses and enforces any SAML Authorization Decision Statements that are a part of the requester's attribute bag. Typically SAMLAuthzAssertionPIP is used in tandem with this PDP to collect those attributes.

The PDP expects the service name as SAML Resource and the operation name as action.

9.3. Configuration

No configuration is required.

9.4. Decision Table

No requester attributes	NOT_APPLICABLE
No resource attributes	INDETERMINATE
No action attributes	INDETERMINATE
If atleast one of the SAML Decision statement is permit for access to said resource and action	PERMIT
If none of the SAML Decision statement is permit for access to said resource and action	DENY

9.5. Related interceptors

- Default bootstrap interceptor (X509 Bootstrap PIP), is required to use this PDP.
- SAML Authorization Assertion PIP (SAMLAuthzAssertPIP is required to use with this PDP to be able to collect SAML Authorization Assertions that are sent in as a part of the request.

10. Self Authorization

10.1. Class name

`org.globus.wsrf.impl.security.authorization.SelfAuthorization`

10.2. Overview

This PDP does self authorization and expects the peer subject to match the local subject, which is the current JAAS Subject associated with the service/resource. The current JAAS subject is determined by the value of the run-as element in the service security descriptor (see Configuring run-as mode).

10.3. Configuration

No configuration is required

10.4. Decision Table

No peer subject	DENY
No local subject	INDETERMINATE
Peer subject matches local subject	PERMIT
Peer subject does not match local subject	DENY

10.5. Related interceptors

No interceptors are related other than the default bootstrap interceptor (X509 Bootstrap PIP), which is required to be used with this PDP.

11. Username Authorization

11.1. Class name

`org.globus.wsrf.impl.security.authorization.UsernameAuthorization`

11.2. Overview

Uses Java Login module to authorize based on user name and password used on the method call. The username and password are passed to the Login module using javax.security.auth.callback.NameCallback and javax.security.auth.callback.PasswordCallback.

Username authorization as part of the authorization framework for a service involves setting up a Configuration class that reads in the Login Module to be using. It is also possible to specify the login modules to be used by altering some parameters in the java.security file located at $JAVA_HOME/jre/lib/security which has been discussed under the general configuration section. Steps for username authorization using Login Modules configured via custom Configuration classes is described under the custom configuration section.

11.3. Configuration

Login modules can be setup for use by our application in two different ways. One, by adding the login module to the list of login modules used by the JVM. This method has been explained under the JVM Configuration. Second, by creating a custom configuration class to dynamically pick up the login module.

11.3.1. JVM Configuration

Some login module and configuration file needs to be configured (example: jre/lib/ext/jaasmod.jar and jre/lib/security/java.security). A sample configuration entry:

```
login.configuration.provider=com.sun.security.auth.login.ConfigFile
login.config.url.1=file:${java.home}/lib/security/jaas.config
```

Sample jaas.config file:

```
Login {
 com.sun.security.auth.module.NTLoginModule required;
};
```

11.3.2. Custom Configuration

A configuration class is used to setup a specified login module for executing authorization. The class details are provided to the Toolkit through the security descriptor for the service that enforces username auhorization. In order for the username authorization handler to use a custom Login Configuration, a parameter with name "login-config" and a value containing the custom Configuration classname has to be specified along with the PDP in the security descriptor. The PDP in this case is "Username Authorization". The relevant snippet from the service security descriptor is shown below:

```
    <authzChain>
 <pdps>
        <interceptor name="prefix:org.globus.wsrf.impl.security.authorization.UsernameAuth
   <parameter>
      <param:nameValueParam>
      <param:parameter name="login-config" value="org.globus.wsrf.impl.security.authorizat
      </param:nameValueParam>
   </parameter>
 </interceptor>
 </pdps>
    </authzChain>
```

Internal Details

The usernameAuthorization PDP loads the Configuration specified in the parameter "login-config" and sets that as the current configuration.(These are internal details which may be disregarded safely for the purposes of writing up a custom configuration) `org.globus.wsrf.impl.security.authorization.UsernameAuthorization`

```
String className = (String) this.chainConfig.getProperty(this.prefix, "login-config
Class config = ClassLoaderUtils.forName(className);
Object loginConfig = config.newInstance();
if(loginConfig instanceof Configuration ){
  Configuration.setConfiguration((Configuration) loginConfig);
}
```

The custom Configuration class has to be a subclass of `javax.security.auth.login.Configuration`. An example custom Configuration class is available in the unit tests directory at `org.globus.wsrf.impl.security.authorization.LoginConfiguration`. The login module that has to be used is specified using a login.config file that the configuration class uses. The configuration class loads the appropriate properties from this file. A sample entry for this file is:

```
Login {
   org.globus.wsrf.impl.security.authorization.UsernameLoginModule required;
};
```

More details regarding Login Configuration files can be found here[3]. The next section focuses on the setup and usage of login modules.

11.4. Login Modules

The login modules can be used to handle the authorization of users. The details regarding writing login modules is beyond the scope of this document and can be found at in the Login Module Developers' guide[4]

The unit tests have an example of a login module `org.globus.impl.security.authorization.UsernameLoginModule`. The username and password to authorize the client have to be passed via the client security descriptor similar to the one listed below. If the password type is not provided then it defaults to PasswordText. In that case, the passwords are sent unencrypted over the wire.

```
<clientSecurityConfig xmlns="http://www.globus.org/security/descriptor/client">
  <GSISecureTransport>
   <integrity/>
  </GSISecureTransport>

  <usernameType>
   <username value="globus"/>
     <passwordType>
         <password value="unittesting"/>
         <type value="http://docs.oasis open.org/wss/2004/01/oasis-200401-wss-usernam
      </passwordType>
  </usernameType>

  <authz value="none"/>

   </clientSecurityConfig>
```

This login module handles the authorization of usernames and passwords supplied either as Text (unencrypted format) or as a Digest (encrypted format.) by comparing them to the username and password stored on the server in a file. The format of the parameters in the password file used by our example Login Module is illustrated in the following example. `~/tests/unit/etc/Test-authz-pwd`

```
name=globus
```

[3] http://java.sun.com/j2se/1.4.2/docs/guide/security/jaas/JAASRefGuide.html#AppendixB
[4] http://java.sun.com/j2se/1.4.2/docs/guide/security/jaas/JAASLMDevGuide.html#Intro

```
password=unittesting
usernames=globus
```

In summary, Login Modules can be used to implement username authorization. This can be done by either changing the java environment to invoke your login module or by programmatically setting the authorization framework to use your login module. The unit tests provide a good example of how this can be done programmatically which in most cases would prove beneficial.

11.5. Decision Table

If no error is returned from Login module	PERMIT

11.6. Related interceptors

None.

DRAFT

Chapter 14. PIP Reference

1. Introduction

[fixme - what are PIPs]

If you have a PIP you'd like to contribute to the Globus Toolkit, use the following template:

- PIP template[1]

Note

The above files are in DocBook XML format. Simply save the link to your hard drive, edit the file in a text or xml editor and email to ?. Don't worry about getting the tags exactly right, it's enough to enter the information where it makes sense and we'll clean up the tags where necessary.

2. Container PIP

2.1. Class name

`org.globus.wsrf.impl.security.authorization.ContainerPIP`

2.2. Overview

This implements the BootstrapPIP interface [PIP-glossary] and is used with in the toolkit to initialize the request entities. It collects information about the service and operation invoked. It is always invoked prior to any authorization processing.

2.3. Configuration

No configuration is required.

2.4. Attributes Collected

This PIP collects three attributes described in the following tables:

[1] ../Java_WS_Security_PIP_Template.xml

Table 14.1. Attribute I

Description of attribute	Message Context associated with the thread
Identity attribute	Identity attribute
Attribute ID	Constants.MSG_CTX_ATTRIBUTE_URI
Datatype	Constants.MSG_CTX_DATATYPE_URI
Issuer	null. The issuer is null since the message context is required to construct the container entity, which is the default issuer for attributes collected in the container.
Validity from	Current time
Validity to	Infinity

Table 14.2. Attribute II

Description of attribute	URL of the service invoked.
Identity attribute	Identity attribute
Attribute ID	Constants.SERVICE_ATTRIBUTE_ID_URI
Datatype	Constants.STRING_DATATYPE_URI
Issuer	Container
Validity from	Current time
Validity to	Infinity

Table 14.3. Attribute III

Description of attribute	Name of the operation invoked.
Identity attribute	Identity attribute
Attribute ID	Constants.OPERATION_ATTRIBUTE_ID_URI
Datatype	Constants.STRING_DATATYPE_URI
Issuer	Container
Validity from	Current time
Validity to	Infinity

This PIP also sets up the container issuer entity, which is used as the default issuer for attributes collected in the container. The entity has the following attributes:

Table 14.4. Attribute I

Description of attribute	Container id
Identity attribute	Identity attribute
Attribute ID	Constants.CONTAINER_ATTRIBUTE_URI
Datatype	Constants.STRING_DATATYPE_URI
Issuer	null
Validity from	Current time
Validity to	Infinity

DRAFT

Table 14.5. Attribute II

Description of attribute	Java Principals from container credential, only if credentials are configured.
Identity attribute	Identity attribute
Attribute ID	Constants.PRINCIPAL_ATTRIBUTE_ID
Datatype	Constants.PRINCIPAL_DATATYPE_URI
Issuer	null
Validity from	Current time
Validity to	Infinity

Table 14.6. Attribute III

Description of attribute	Java Subject from container credential, only if credentials are configured.
Identity attribute	Identity attribute
Attribute ID	Constants.SUBJECT_ATTRIBUTE_ID
Datatype	Constants.SUBJECT_DATATYPE_URI
Issuer	null
Validity from	Current time
Validity to	Infinity

The container entity is created with the same attributes as above with the above entity as the issuer.

2.5. Related interceptors

None.

3. X509Bootstrap

3.1. Class name

`org.globus.wsrf.impl.security.authorization.X509BootstrapPIP`

3.2. Overview

This implements the BootstrapPIP interface [PIP-glossary] which is used when X509 Certificates are used during authentication scheme. It collects peer entities' attributes obtained from the certificates presented by the peer.

3.3. Configuration

No configuration is required.

3.4. Attributes Collected

This PIP collects two attributes described in the following tables:

Table 14.7. Attribute I

Description of attribute	Peer's Subject object
Identity attribute	Identity attribute
Attribute ID	Constants.SUBJECT_ATTRIBUTE_ID
Datatype	Constants.SUBJECT_DATATYPE_URI
Issuer	Container Issuer Entity
Validity from	Current time
Validity to	Infinity

Table 14.8. Attribute II

Description of attribute	Peer's principals
Identity attribute	Identity attribute
Attribute ID	Constants.PRINCIPAL_ATTRIBUTE_ID
Datatype	Constants.PRINCIPAL_DATATYPE_URI
Issuer	Container Issuer Entity
Validity from	Current time
Validity to	Infinity

3.5. Related interceptors

If X509 Certificates are used for authentication, this bootstrap is used by the Authorization Framework by default.

4. SAML Authorization Assertion PIP

4.1. Class name

`org.globus.wsrf.impl.security.authorization.SAMLAuthzAssertionPIP`

4.2. Overview

The PIP extracts SAML Authorization Assertion from the request and adds it to the bag of attributes. The message context and the proxy certificate are checked to see if SAML Authorization Assertions are presents.

If the subject DN in the decision statement matches with the requestor's then the attribute is merged with the requestor's bag of attributes.

4.3. Configuration

No configuration information is required.

4.4. Attributes Collected

This PIP collects attributes described in the following tables:

Table 14.9. Attribute I

Description of attribute	Subject DN from the subject in SAML Authorization Decision Statement (one attribute per statement in assertion)
Identity attribute	Identity attribute
Attribute ID	Constants.PRINCIPAL_ATTRIBUTE_ID
Datatype	Constants.PRINCIPAL_DATATYPE_URI
Issuer	Container Issuer Entity
Validity from	Extracted from assertion
Validity to	Extracted from assertion

Table 14.10. Attribute II

Description of attribute	SAML Authoirzation Decision Statement (one attribute per statement in assertion)
Identity attribute	Non-Identity attribute
Attribute ID	Constants.SAML_AUTHZ_DECISION_ATTRIBUTE_ID
Datatype	Constants.SAML_AUTHZ_DECISION_DATA_TYPE
Issuer	Container Issuer Entity
Validity from	Extracted from assertion
Validity to	Extracted from assertion

4.5. Related interceptors

This PIP can be used in tandem with SAMLAuthzAssertionPDP.

5. Parameter PIP

5.1. Class name

`org.globus.wsrf.impl.security.authorization.ParameterPIP`

5.2. Overview

This PIP extracts configured parameter element from the SOAPMessage. The parameter is added as an action attributes in the associated RequestAttribute.

5.3. Configuration

`parameterConfig` Property pointing to configuration file with information about the service, method and parameter to extract as attributes. If configured file name is not absolute, an attempt is made to find the file as provided, if not an attempt is made to locate it relative to GLOBUS_LOCATION and if that fails, an attempt it made to locate it relative to current directory.

The configuration file is read and stored as SOAPParameter. This class is used to store a specific parameter element path for a given operation for a said service. `servicePath operationName ParameterPath`

The parameter path is a list of QNames, where each QName is QName of a child element of previous QName element. The parameterPath is a string with string representation of each QName, in the order it needs to be looked into with semicolon (;) as delimiter. For example, *{http://temp.ns}element1;{http://temp.ns}nextElem2;{http://temp.ns}nextElem3* would represent the parameter {http://temp.ns}nextElem3. The SOAPBody element here is {http://temp.ns}element1, with nextElem2 as its child and nextElem3 as its child.

5.4. Attributes Collected

This PIP collects two attributes described in the following tables:

Table 14.11. Attribute I

Description of attribute	Configured parameter if it occurs in that operation. The value is an object of type org.w3c.dom.Node and represents the parameter of the operation.
Identity attribute	Identity attribute
Attribute ID	Parameter path as described in previous section.
Datatype	Constants.PARAMETER_PATH_DATA_TYPE
Issuer	Container Issuer Entity
Validity from	Current time
Validity to	Infinity

5.5. Related interceptors

This PIP can be used in tandem with ResourcePropertiesPDP to parameter-based authorization for resource property access.

DRAFT

Chapter 15. Debugging

Because Java WS A&A is built on top of Java WS Core, developer debugging is the same as described in Chapter 10, Debugging.

For information about system administrator logs, see Chapter 6, Debugging.

Java WS Core also provides an API for CEDPs-compliant logging as described in Section 1.2, "Configuring system administration logs".

1. Debugging authorization

Log output from the authorization framework is a useful tool for debugging issues. Because the Authorization Framework is built on top of Java WS Core, developer debugging is the same as described in Chapter 10, Debugging.

1.1. Development Logging in Java WS Core

The following information applies to Java WS Core and those services built on it.

Logging in the Java WS Core is based on the Jakarta Commons Logging[1] API. Commons Logging provides a consistent interface for instrumenting source code while at the same time allowing the user to plug-in a different logging implementation. Currently we use Log4j[2] as a logging implementation. Log4j uses a separate configuration file to configure itself. Please see Log4j documentation for details on the configuration file format[3].

1.1.1. Configuring server side developer logs

Server side logging can be configured in `$GLOBUS_LOCATION/container-log4j.properties`, when the container is stand alone container. For tomcat level logging, refer to Logging for Tomcat[4], . The logger `log4j.appender.A1` is used for developer logging and by default writes output to the system output. By default it is set for all warnings in the Globus Toolkit package to be displayed.

Additional logging can be enabled for a package by adding a new line to the configuration file. Example:

```
#for debug level logging from org.globus.package.FooClass
log4j.category.org.globus.package.name.FooClass=DEBUG
#for warnings from org.some.warn.package
log4j.category.org.some.warn.package=WARN
```

1.1.2. Configuring client side developer logs

Client side logging can be configured in `$GLOBUS_LOCATION/log4j.properties`. The logger `log4j.appender.A1` is used for developer logging and by default writes output to the system output. By default it is set for all warnings in the Globus Toolkit package to be displayed.

[1] http://jakarta.apache.org/commons/logging/
[2] http://logging.apache.org/log4j/
[3] http://logging.apache.org/log4j/docs/api/org/apache/log4j/PropertyConfigurator.html#doConfigure(java.lang.String, org.apache.log4j.spi.LoggerRepository)
[4] http://tomcat.apache.org/tomcat-5.5-doc/logging.html

1.2. Enabling verbose logging

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[5] 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
```

[5] http://www.globus.org/cog/java/

Chapter 16. Troubleshooting

For a list of common errors in GT, see Error Codes.

DRAFT

1. Error Messages For Java WS A&A

DRAFT

Table 16.1. Java WS A&A Errors

Error Code	Definition	Possible Solutions
`[JWSSEC-248] Secure container requires valid credentials`	This error occurs when globus-start-container is run without any valid credentials. Either a proxy certificate or service/host certificate needs to be configured for the container to start up.	1. If you are not looking to start up a container that uses GSI Secure Transport, which is used by the container by default, use `globus-start-container -nosec`. You will be able to use insecure clients and services. However, this also implies that if you have not configured individual services with credentials, you will not be able to securely access the service. 2. If you are running a personal container, generate a proxy certificate with grid-proxy-init. If the proxy certificate is not in the default location, configure the container security descriptor as described in Configuring Container Security Descriptor. 3. If you want to use host certificates, configure the container security descriptor as described Configuring Credentials.
`Failed to start container: Container failed to initialize [Caused by: [JWSSEC-250] Failed to load certificate/key file]`	This error occurs if the file path to the container certificate and key configured are invalid.	1. The path to the container certificate and key are configured in `$GLOBUS_LOCATION/etc/globus_wsrf_core/global_security_descriptor.xml`. This file is loaded as described [here - fixme link]. Ensure that the path is correct.
`Failed to start container: Container failed to initialize [Caused by: [JWSSEC-249] Failed to load proxy file]`	This error occurs if container proxy file configured is invalid.	1. The path to the container proxy certificates are configured in `$GLOBUS_LOCATION/etc/globus_wsrf_core/global_security_descriptor.xml`. This file is loaded as described [here - fixme link]. Ensure that the path is correct.
`Failed to start container: Container failed to initialize [Caused by: [JWSSEC-245] Error parsing file: "etc/globus_wsrf_core/ global_security_descriptor.xml" [Caused by: ...]`	This error occurs if the container security descriptor configured is invalid.	1. The container security descriptor should conform to the Container Security Descriptor Schema.[1] 2. Refer to the "Caused by: " section for details on the specific element that is not correct.

[1] http://www.globus.org/toolkit/docs/4.2.0/security/container_security_descriptor.xsd

Error Code	Definition	Possible Solutions
`[JGLOBUS-77] Unknown CA`	This error occurs if the CA certificate for the credentials being used is not installed correctly.	1. If this issue occurs on the server side, the container is not configured with CA certificates. The container looks for trusted certificates in the default location as described Java CoG Toolkit FAQ[2] 2. On the server side, the trusted certificates can be configured as described in Trusted Certificates 3. On the client side, trusted certificates can be configured as described in Configuring Trusted Credentials

[2] http://www.globus.org/cog/distribution/1.2/FAQ.TXT

Chapter 17. Related Documentation

See Section 9, “Associated Standards” for a list of associated standards.

DRAFT

Glossary

E

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

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 this.

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 place. GSI uses proxy certificates for single sign on and delegation of rights to other entities.

For more information about types of proxy certificates and their compatibility in different versions of GT, see http://dev.globus.org/wiki/Security/ProxyCertTypes.

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

T

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

DRAFT