P5.4.5
Design and implementation of dynamic security components

Workpackage:	5	Grid Dynamics	
Author(s):	Thomas Leonard Mark McArdle Mike Surridge		IT Innovation
	Mehran Ahsant		KTH
Authorized by	Mike Surridge		IT Innovation
Doc Ref:	P5.4.5		
Reviewer	Stefano Beco		DATAMAT
Reviewer	Peer Hasselmeyer		NEC
Dissemination Level	Public: equivalent to PU in DoW		

Date	Author	Comments	Version	Status
2006-02-17	Thomas Leonard	Initial Outline	0.1	Draft
2006-02-26	Mehran Ahsant	Completing the STS contributions	0.2	Draft
2006-03-06	Mike Surridge	High level review of content and readability, added comments and some changes	0.2 ms	Draft
2006-03-06	Mark McArdle Thomas Leonard	Implemented changes from MS	0.3	Draft
2006-03-07	Mark McArdle Thomas Leonard	Many changes including introduction, questions, WS-Addressing, account service implementation and conclusions.	0.4	Draft
2006-03-08	Thomas Leonard	Integrated changes from Mehran	0.5	Draft
2006-03-15	TAL, MM	Updated for Peer's Comments	1.0	Draft

Executive Summary

This document describes project outputs P5.4.5a and P5.4.5b from NextGRID WP5, comprising dynamic security core components of NextGRID for dynamic policy and dynamic token services respectively.

Task 5.4 is concerned with federated security and trust. Basic Grid security is based on long-established mechanisms drawing from a wealth of off-the-shelf technology and standards. Work in NextGRID during the first year therefore focused on “higher-level” security federation issues, including dynamic distributed authorisation and authorisation policy, trust and accountability and their relationship with higher-level Web Service specifications such as WS-Trust and WS-Federation.

In this period, the focus has been in two main areas: development of a NextGRID security reference implementation, and validation of the NextGRID dynamic security component model. These components will enable continued exploration of virtual organisations and distributed enterprise models in relation to the bipartite SLA-based architectural principles of NextGRID.

The work is comprised of two parts: an implementation of a WS-Trust dynamic security token service, led by KTH, and an implementation of a dynamic security policy framework, led by IT Innovation.

The component model and the reference implementation have been evaluated using an experiment on an accounting scenario based on an application from NextGRID partner KINO. The experiment involved the development of two accounting services; one to issue tokens to users within a client organisation, and one to validate the tokens at each service provider. The tokens themselves are signed SAML assertions, and the policy for accepting them is set dynamically at the remote service by the client organisation when a federation between them is established.

Having the client organisation specify the policy for validation of the tokens it issues removes the need to pre-agree role ontologies between organisations. Issuing SAML tokens in a client domain and validating them in the service provider domain allows for simple and robust access control management, improving on earlier implementations where the access control lists are maintained at multiple remote sites. The experiment shows that the NextGRID architecture enables rapid bipartite federation using the “fast VO” model.

Table of Contents

1 Introduction

NextGRID WP5 is concerned with dynamic aspects of Grid systems: how Grid services and clients interact to enable agile federation of resources, services and users between disparate domains. Its goal is to investigate these issues using experimental software.

Task 5.4 is concerned with federated security and trust. Basic Grid security is based on long-established mechanisms drawing from a wealth of off-the-shelf technology and standards. Work in NextGRID during the first year therefore focused on "higher-level" security federation issues, including dynamic distributed authorisation and authorisation policy, trust and accountability and their relationship with higher-level Web Service specifications such as WS-Trust and WS-Federation.

An initial experiment at M06 showed that design patterns using WS-Trust and WS-Federation in conjunction with a dynamic authorisation policy facility would enable implementation of a wide range of dynamic trust and security federation models. Subsequent work has focused on accountability, manageability and standards compliant reference components to implement these patterns.

In this period, the focus has been in two main areas: reference implementation and validation of NextGRID dynamic security components, and continued exploration of virtual organisations and distributed enterprise models in relation to the bipartite SLA-based architectural principles of NextGRID.

Related work is ongoing in WP4, which is developing service endpoint security mechanisms, including access policy decision and enforcement points and policy-driven encryption.

This report describes the software components implemented for experiments [1] to explore issues relating to token issuance, exchange and validation in dynamic grids. This work is comprised of two parts:

- Implementation of a WS-Trust dynamic security token service as part of the NextGRID dynamic security core, led by KTH.
- Implementation of a dynamic security policy framework as part of the NextGRID dynamic security core, led by IT Innovation.

The experiment is based around an accounting scenario (described in the next section) and has included the development of two accounting services (client and service-side). These accounting services will also be used in M24 experiments focussing on how they can be used to provide interoperable accounting mechanisms in different infrastructures.

2 Objectives

The objective of this experiment was to evaluate and validate the NextGRID dynamic security architecture and components developed in Task 5.4. To provide a focus for the work, we adopted a simple test scenario from a NextGRID end user (KINO's Digital Media Use Cases). This is similar to the scenario used in the initial experiment at M06, but incorporates new capabilities provided by the NextGRID security components.

Figure 1 shows a model of the NextGRID security components used to enable cross-domain federations. The following sections describe how each of these components is used.

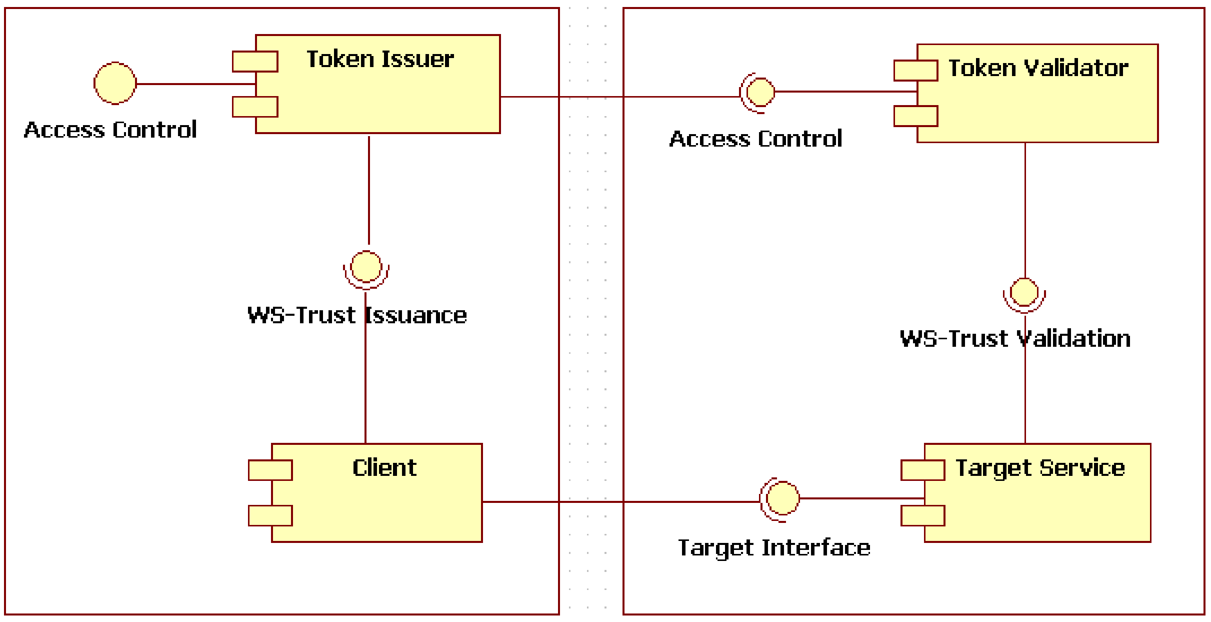


Figure 1 NextGRID security architecture component model

NextGRID leverages SOAP Message Security for authenticating and encrypting SOAP messages. The X.509 token profile is the default security token format, which works well in static setups, i.e., where the participating services are known beforehand and trust relationships are in place. For dynamic setups (where new service nodes in untrusted domains can appear), other token profiles may be more suitable.

2.1 Client Scenario

In the course of their business, KINO has a need to perform high-definition 3D digital video rendering calculations, taking virtual 3D scenes and characters and generating high-quality video sequences from them. This is computationally expensive, and not needed every day, so it is desirable to use externally-provided (i.e. Grid) computational resources. How can KINO quickly establish a business relationship with a new Grid supplier, and how can they control access to it by users within their own organisation?

We assume that the client organisation already uses its own local authentication system, based on Kerberos. When users log in to their desktop machines with their password they get a Kerberos Ticket Granting Ticket. They should not have to provide any further authentication to access Grid resources; authentication across domains can be federated automatically by the NextGRID security systems.

The Use Case diagram below shows the actors involved. In order to submit a job, the animator at KINO needs there to be an approved KINO trade account open at the service provider, and they need a security token proving to the service provider that they are permitted to use it.

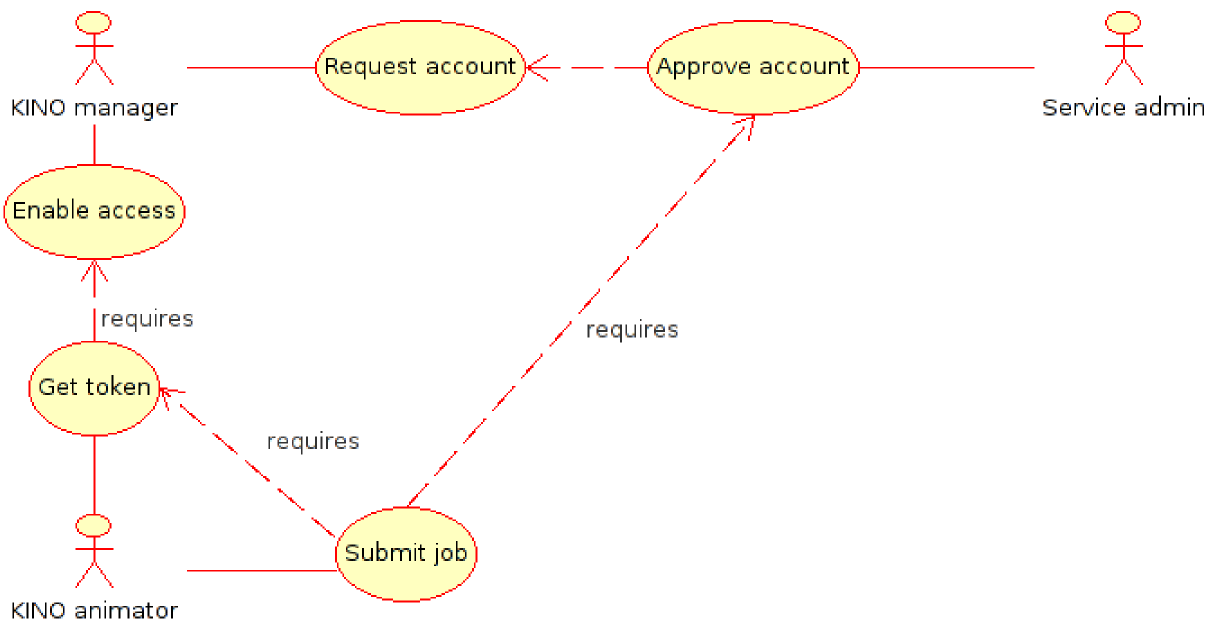


Figure 2. KINO Use Cases

The manager at KINO requests a trade account at the service provider once they have decided to do business with them, and waits for it to be approved

Internally, KINO runs a client accounting service (corresponding to the generic token issuer component in Figure 1) which records which members of staff are permitted to spend money on KINO's various trade accounts. The manager enables access to the new trade account not by setting the access control policy of the remote trade account directly (as in the P5.4.1 experiment), but by altering the policy of this local service. Users invoke an operation on this service to get the security tokens they require to charge to the remote account. This allows the animators to use remote services in order to complete jobs quicker than they could internally, and the budget holder can monitor their activities and regulate access to security tokens through the client accounting service, to prevent them spending too much money on each project.

2.2 Service Provider Scenario

The service provider has a number of services that it wants to allow other companies to use and be billed for the time spent using them. In this scenario the service provider has a number of 3D video rendering services.

The service provider sets up an account service for clients to request trade accounts, which can then be used by users in the client organisation to run jobs. This corresponds to the token validator in Figure 1. When the service provider receives a request for a new trade account from the client organisation the service administrator checks that the credit details are valid and approves or declines the account.

The service provider wants to make it as quick and easy as possible for new clients to sign up. At the same time, the service provider must ensure the security of their services and protect their existing clients. A weakness of the P5.4.1-5.4.2 experiment was that setting up a new account required the service provider to trust the client's certificate authority globally, which requires long and careful checks of new clients. The new security components developed in task 5.4 remove this limitation.

2.3 NextGRID Architectural Questions Addressed

The NextGRID Architecture Straw Man document [13] lists a number of architectural questions which NextGRID needs to address. Since then, additional questions have been raised; these are tracked in the P1.4.x series of documents. The questions addressed by this experiment are:

Addressing

- 5.1) Does WS-Addressing address security considerations of all parties (for e.g. intermediaries) involved in the SOAP processing chain?

Authentication

- 33) What other authentication mechanisms (e.g. username-password) should be supported by the NextGRID architecture?
- 34) What mechanisms should be used for identity federation/trust federation (federated token issuance and verification)? How is this related to emerging standards capable of dealing with identity such as WS-Trust, WS-Federation, etc?
- 35.1) Is a digital signature over the message body only sufficient?

Authorisation

- 36.5) What is the model for dynamic authorization? Are there base specs that we can work from or use here (WS-Policy?)
- 37) How do dynamic authorisation models relate to dynamic resource management and virtual organisation management?
- 38) Do we need a process role vocabulary coming out of NextGRID?

Trust

- 53.2) How should trust be represented and analysed in (business) processes; can the methodology used by GRIA be developed to address this?
- 54) How can trust be managed in a dynamic business context, including VO lifecycles, and delegation mechanisms?

3 Starting Points

3.1 GRIA

GRIA is a Web Service grid middleware created by the University of Southampton and NTUA in the GRIA project, based on components developed by them in GRIA and also in the EC GEMSS and UK e-Science Comb-e-Chem projects. The GRIA middleware was tested using two industrial applications, one of which was KINO's high-definition video rendering application.

GRIA has the following key characteristics:

- it uses secure "off the shelf" web services technology;
- it is designed for business users: supporting B2B functions and easy-to-use APIs;
- it can easily support legacy applications: all the applications used in the GRIA project are legacy applications;
- it is ready today: the current release is GRIA 4.3.1, available via the GRIA website [2].

Unlike more “traditional” grids, GRIA was designed from the outset to support commercial service provision between businesses, by supporting conventional B2B procurement processes. The security infrastructure of GRIA is designed to support and enforce these processes, so that nobody can use GRIA services without first agreeing to pay the service provider. The procedure for using GRIA services was summarised in Section 2.2 of NextGRID Project Output P5.4.1/P5.4.2 [3].

The GRIA middleware was a convenient starting point for these experiments because (a) it already has a dynamic authorisation mechanism, and (b) applications needed for KINO’s scenario are already available as GRIA services from the original GRIA project.

The wholly distributed management and policy enforcement model used in the current GRIA release makes it easy to deploy and operate, and provides resilience against security breaches in other sites. However, it also makes it hard for service consumers to manage access and monitor usage across large numbers of service providers. GRIA also uses a conventional PKI authentication model, which prevents rapid formation of relationships with new service providers, even though the decentralised authorisation mechanisms would allow this. The work described in this report shows how a next generation security architecture can overcome these shortcomings.

3.2 KX.509

The P5.4.1-P5.4.2 experiment used the KX.509 protocol to map trust (actually identity tokens) between Kerberos [4] and X.509 [5]. The KX.509 protocol was developed at University of Michigan [6] and provides a mechanism for obtaining X.509 identity certificates based on a Kerberos domain log-in identity. There are two principal components:

1. The KCA is a Kerberised service, running inside a Kerberos domain, which provides the functionality of an X.509 certification authority.
2. KX509 is a standalone client program that acquires a short-lived X.509 certificate from the KCA for an authenticated Kerberos user.

The protocol between the client and the KCA is called KX.509. The normal mode of operation is for the KCA to sign short-lived X.509 certificates for applicants that it can authenticate through Kerberos. The lifetime of the certificate is equal to the lifetime of the Kerberos ticket used for the Kerberos authentication, while the distinguished name (DN) is based on an LDAP catalogue of users and organisational units.

Although KX.509 does not support the standards proposed in the NextGRID Straw Man architecture, it provided a convenient starting point for the experiment.

3.3 Apache WSS4J

Apache WSS4J is an implementation of the OASIS Web Services Security (WS-Security) from the OASIS Web Services Security TC. WSS4J is a Java library that can be used to sign and verify SOAP Messages with WS-Security information. WSS4J will use Apache Axis and Apache XML-Security projects and is interoperable with JAX-RPC based server/clients and .NET server/clients.

3.4 Apache Axis

Axis is an implementation of the SOAP protocol. It shields the developer from the details of dealing with SOAP and WSDL. This experiment uses Axis within the web services (each deployed as a Tomcat webapp), and also on the client side to invoke the services.

3.5 JAAS

The Java^{TM} Authentication and Authorization Service (JAAS) is used for authentication of users to diverse security domains. JAAS authentication is performed in a pluggable fashion. This permits applications to remain independent from underlying authentication technologies. Then new or updated technologies can be plugged in without requiring modifications to the application itself. An implementation for a particular authentication technology to be used is determined at runtime. The implementation is specified in a login configuration file which should be configured as we describe later in this document. For this experiment, we use JAAS authentication technology for Kerberos technology.

3.6 Standards and Specifications

3.6.1 WS-Security

WS-Security [7] is a set of SOAP extensions to provide message-level integrity, confidentiality and authentication. WS-Security enables collaboration between other Web service security standards and protocols. It does not dictate one specific security technology and allows organizations to use heterogeneous security models and encryption technologies for multiple security tokens, multiple trust domains, multiple signature formats, and multiple encryption technologies. Message digests of selected parts of the message (for example, the SOAP body) can be given in a security SOAP header block, and signed using various methods.

3.6.2 WS-Trust

WS-Trust [8] defines a protocol by which web services in different trust domains can exchange security tokens for use in the WS-Security header of SOAP messages. Clients use the WS-Trust protocols to obtain security tokens from Security Token Services. WS-Trust is highly relevant to the question of how to obtain an X.509 certificate for accessing a web service based on a Kerberos-authenticated identity – indeed this is a scenario commonly used to illustrate how WS-Trust works.

The main limitation of WS-Trust is that it doesn’t actually provide any mechanisms to manage trust, only token exchange between entities that already trust each other. Normally, services can trust each other only when they are in a single security domain. If services are in different security domains then some other method must be used so that one service can verify tokens issued by the other. The only solution provided by WS-Trust if this is not the case is to have an additional security token service within the service's realm, but this only moves the problem from the web service to the new token service.

3.6.3 WS-Federation

WS-Federation [9] describes how to use WS-Trust, WS-Security and WS-Policy together to provide federation between security domains. However, WS-Federation does not define any standard way to establish this trust relationship dynamically. According to the specification:

"The following topics are outside the scope of this document:

- *Definition of message security or trust establishment/verification protocols..."*

The specification defines a number of variations on the exchange of tokens between domains. For example, the user can send the web service a message signed by one token service, leaving the web service to call its own token service to convert the token into one it can understand and respect. Alternatively, the user can call the remote token service to get a token the web service can understand. WS-Federation can also be used to implement a "delegation" model, in which the user provides a security token to their delegate using the mechanism described in WS-Trust. The delegate then includes this token in any messages they send to the service, allowing the service to understand for whom the delegate is working.

Thus, trust relationships must already exist between the WS-Trust token services in a WS-Federation exchange. Although these two specifications describe the message exchanges needed, they do not solve the problem of dynamic trust and security federation.

3.6.4 SAML

SAML[10,11] (the Security Assertion Markup Language) is a specification from OASIS based on assertions. SAML assertions include authentication information (such as that the user logged in at a particular time) and attributes, such as asserting that a user has a particular role. Although SAML has its own protocol which overlaps with WS-Trust, it is still possible to use SAML purely to represent tokens, and there is a WS-Security profile for doing this [12].

4 Experimental Setup

4.1 Overview

Figure 3 shows the services used to evaluate the security components:

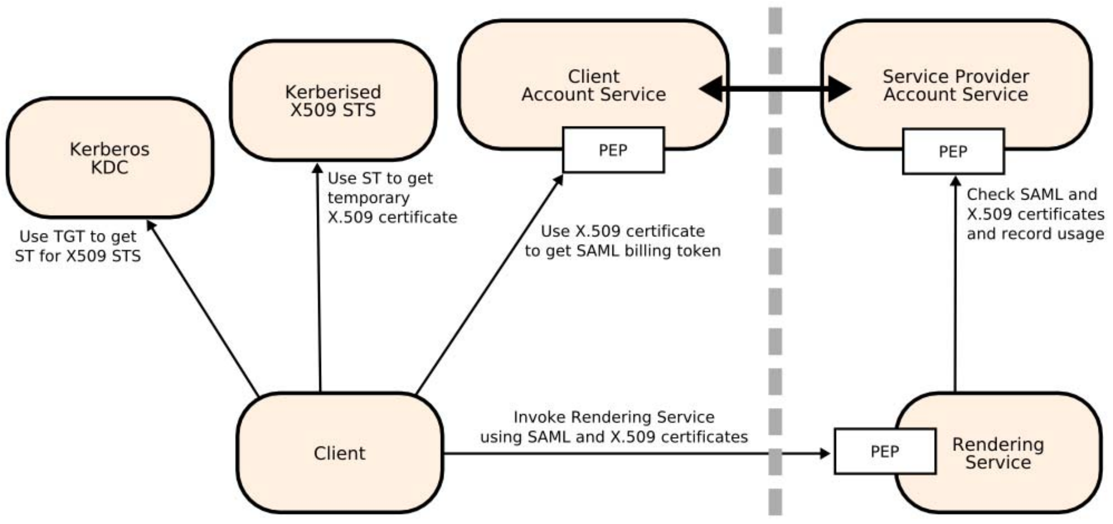

Kerberos KDC
Kerberised X509 STS
Client Account Service
PEP
Service Provider Account Service
PEP
Use TGT to get ST for X509 STS
Use ST to get temporary X.509 certificate
Use X.509 certificate to get SAML billing token
Check SAML and X.509 certificates and record usage
Client
Invoke Rendering Service using SAML and X.509 certificates
PEP
Rendering Service

Figure 3. System Architecture

The client organisation (KINO) runs three services for use by their own employees:

- The Kerberos KDC (Key Distribution Center) is used by the animator when logging in. The animator uses their password to get a Kerberos Ticket Granting Ticket (TGT). This ticket is presented to the KDC to get a Service Ticket (ST) for each Kerberised service in the Kerberos domain that the client needs to use.
- The Kerberised STS is a WS-Trust Security Token Service. It authenticates an animator using their Service Ticket (ST) and generates signed short-lived X.509 certificates for them.
- The client account service authenticates the animator using the X.509 certificate and generates a SAML assertion asserting that they may use the remote trade account.

A Security Token Service (STS) is a Web Service that issues or validates security tokens. This is achieved either by converting a security token that a user currently has to one which is suitable for accessing a desired service, or by creating a new security token such as a SAML attribute assertion. All of the client services above are security token services, through which the client application user can get a SAML assertion required to use the remote rendering service. Currently, only the Kerberised STS uses the standard WS-Trust interface.

There are two services run by the service provider:

- The service provider's account service keeps track of money owed by customers, by allowing other services to record charges on a customer's trading account. The billing service must send a SAML token with each charge request, obtained from its client, which must match a policy agreed with the customer for the trading account.
- The rendering service takes a scene description from a client and renders video from it. Before doing so it checks the SAML token the client provided by using it to record a charge on the customer's trade account using the service provider's account service. Only if the token is sufficient will the charge be accepted, whereupon the rendering service can start rendering the video..

The first of these is also a security token service, since it validates the SAML tokens. The two account services also support a distributed management processes needed to control access to the rendering application service. These services (along with the rendering service) are all protected using the dynamic authorisation system, which enables and enforces conformance to these management processes, thus allowing trust between the manager and the service provider to be used as a basis for dynamically federated security. The operation of the Policy Enforcement Point (PEP) component is described in detail in section 5.2.3. The dynamic security token services together with the dynamic authorisation systems form the core components of the NextGRID dynamic security architecture.

To enable individual animators to use the remote trade account the manager could update the remote account's access policy to give specific users access (as in P5.4.1-P5.4.2). However, this creates a management problem as potentially long access control lists must be maintained at multiple remote sites. Instead, KINO updates the remote policy once to specify a single SAML assertion, signed by their own domain (actually their client accounting service), which anybody from that domain could use to charge services to the account. Access to signed SAML assertions is then managed locally via the client accounting service, thus allowing the manager to regulate access to services at an arbitrary number of remote sites via a single policy.

It would be possible to replace the two account services with more generic "federation services" which simply issued tokens and permitted policy updates. However, combining the token issuing with the accounting functionality makes the system more manageable. For example, a manager can ask their client accounting service to aggregate statements from all their trade accounts. If the token issuance and accounting functions were separate, it would be necessary to keep the list of remote trade accounts synchronised between the two.

The following sections describe each of the components in more detail.

4.2 Client Kerberos Systems

If the user holds a Kerberos ticket asserting their identity, but a service the user wishes to invoke needs an X.509 certificate, the Kerberos ticket can be presented to an STS, and the STS issues an equivalent X.509 certificate asserting the same identity of the requester.

The two account services and the rendering service are all protected using the PBAC system. When a manager or an animator at KINO needs to invoke one of these services they must get an X.509 certificate. As illustrated in Figure 4, this certificate can be requested from the client token service by the GRIA client for each authenticated Kerberos user (an animator or manager).

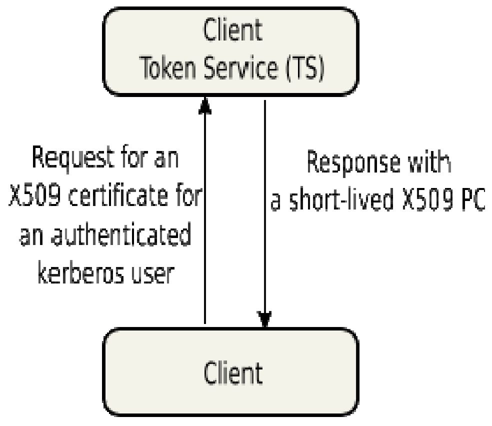


Figure 4 Client Kerberos System Architecture

The details of issuing tokens are described in the implementation description in section 5.4.

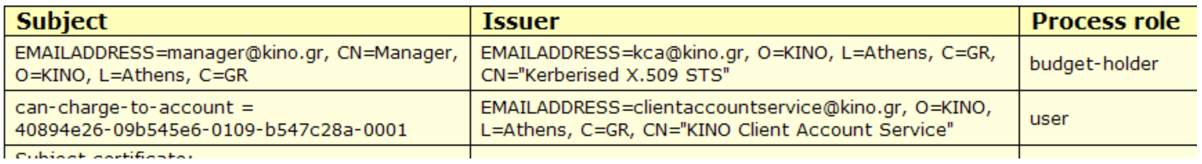
4.3 Client Account Service

The KINO manager wants to quickly set up a relationship with a new supplier of rendering services and effectively manage this and control access to it within the organisation. The SOAP message request to open the remote trade account is signed using the manager's X.509 certificate, obtained using the local Kerberos systems, as described above. The manager tells the service provider's account service to trust KINO's Kerberised STS to identify budget-holders. This trust is only within the context of this one trade account.

When opening a new trade account the manager sends their own X.509 distinguished name and the public key of the client Kerberised STS. This allows the service provider to recognise the manager when they later try to use the account. The manager also grants their local client account service the "budget-holder" role so that it can act on their behalf. Again, this is done by specifying the X.509 distinguished name of the client account service and the Kerberised STS's public key.

The KINO managers typically create new local *project* accounts for each project KINO is involved with using the local client account service. They use the access control interface of the local account service to add animators to different projects. An animator can be working on multiple projects.

The manager "peers" these project accounts with open trade accounts at each service provider. This is a many-to-many mapping; several client project accounts may use the same service provider trade account, and each client project account may use trade accounts at several different service providers. Establishing the peering relationship involves the client account service contacting the service provider's account service and updating its policy to allow any user access if authorised by the client account service (by checking for an attribute chosen by the client account service).



Subject	Issuer	Process role
EMAILADDRESS=manager@kino.gr, CN=Manager, O=KINO, L=Athens, C=GR	EMAILADDRESS=kca@kino.gr, O=KINO, L=Athens, C=GR, CN="Kerberised X.509 STS"	budget-holder
can-charge-to-account = 40894e26-09b545e6-0109-b547c28a-0001	EMAILADDRESS=clientaccountservice@kino.gr, O=KINO, L=Athens, C=GR, CN="KINO Client Account Service"	user
Subject certificate:		

Figure 5. Access Control Rules

In Figure 5 we can see the access rules as they appear on the service provider's administration page. The Subject column gives an attribute or identity that someone must have to get the role shown in the Process role column. The Issuer is a party trusted to assert the attribute or identity. Access control rules are explained in more detail in section 5.2.2.

The rule for the "user" process role grants the "user" role to anyone who has the attribute shown in the subject column ("can-charge-to-account"). The attribute is chosen by the client account service and does not have any special meaning to the service provider's account service. In this case the attribute's value is the resource context identifier of the client account. The attribute must be asserted by the owner of the issuer certificate (i.e., the client account service).

The rule for the “budget-holder” process role requires the subject to have the manager’s X.509 Distinguished Name as asserted by the client’s Kerberised X.509 STS.

When an animator wants to bill a service to a project which they are working on, they must produce an appropriate token that will be recognised by their manager’s trading account with the service provider. This billing token must prove they are allowed to use the project account by asserting the attribute previously specified when the project account was peered with the trading account. They get the token from their local client account service, which they can do if and only if the access control policy set by the manager for the project account allows it.

The billing tokens issued by the local account service take the form of a SAML assertion that asserts:

> *“The subject identified by <some X.509 certificate> has the attribute ‘can-charge-to-account”* – signed by *“the client account service”*.

See section 5.4 for more details about the format of the billing tokens.

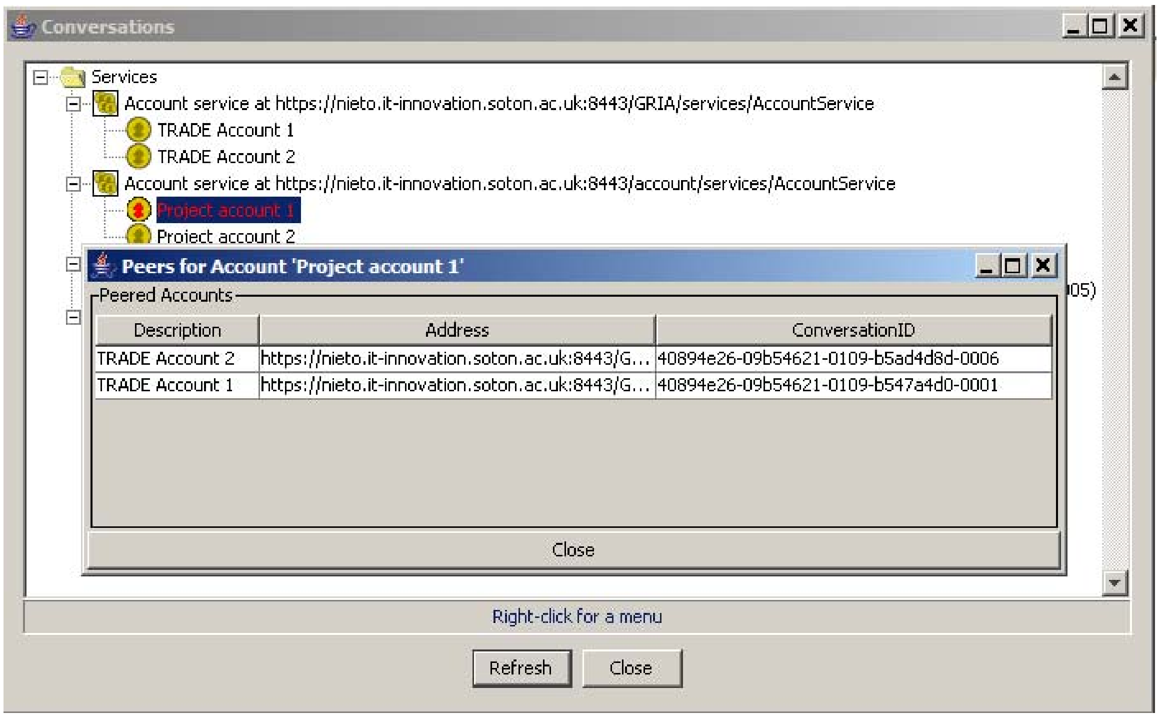
Finally, the client account service can retrieve the list of charges from each trade account it is peered with, and get an overall summary of the money being spent on each project across the consumer organisation. This allows the manager to see which animators are spending what amount of money with each service provider, and how much money is being spent on each project.

When a new animator joins, the manager need only update the internal client account service and add the new user to the project. The manager no longer needs to inform each service provider of the change. So long as the new animator can get access to the security token they can bill the client account.

4.4 Client Application

The client used by both the animator and the manager is a slightly modified version of the GRIA client. The client has been modified to contact the local account service to gain billing tokens and to pass them to the remote services. The client interface has also been extended to provide operations related to peering of accounts.

The KINO manager can manage the peered accounts of each client account via the customised GRIA client application, as shown in Figure 6.


Conversations
Services
Account service at https://nieto.it-innovation.soton.ac.uk:8443/GRIA/services/AccountService
TRADE Account 1
TRADE Account 2
Account service at https://nieto.it-innovation.soton.ac.uk:8443/account/services/AccountService
Project account 1
Project account 2
Peers for Account 'Project account 1'
Peered Accounts
Close
Right-click for a menu
Refresh
Close

Description	Address	ConversationID
TRADE Account 2	https://nieto.it-innovation.soton.ac.uk:8443/G...	40894e26-09b54621-0109-b5ad4d8d-0006
TRADE Account 1	https://nieto.it-innovation.soton.ac.uk:8443/G...	40894e26-09b54621-0109-b547a4d0-0001

Figure 6. Client Application – Peered Accounts

Each animator has a short lived X509 Certificate they acquire from the Client Kerberos System. When the animator tries to submit a rendering job to the remote job service, the client performs these steps:

1. The client software queries the remote job service for its policy, and discovers that an account at the service provider is needed.
2. The client sends a request to the local KINO account service (authenticating using its short-lived X.509 certificate), listing these account services and gets back a WS-Addressing Endpoint Reference for a trade account, and a SAML assertion stating that the animator has the required attribute to charge to it.
3. The client sends the rendering job to the rendering service, quoting the trade account endpoint reference and including the SAML assertion.
4. The job runs, and the trade account is billed for the usage.

The animator will have a different local project account for each project they are working on. Before submitting any jobs, the animator selects the appropriate project account, as shown in Figure 7.

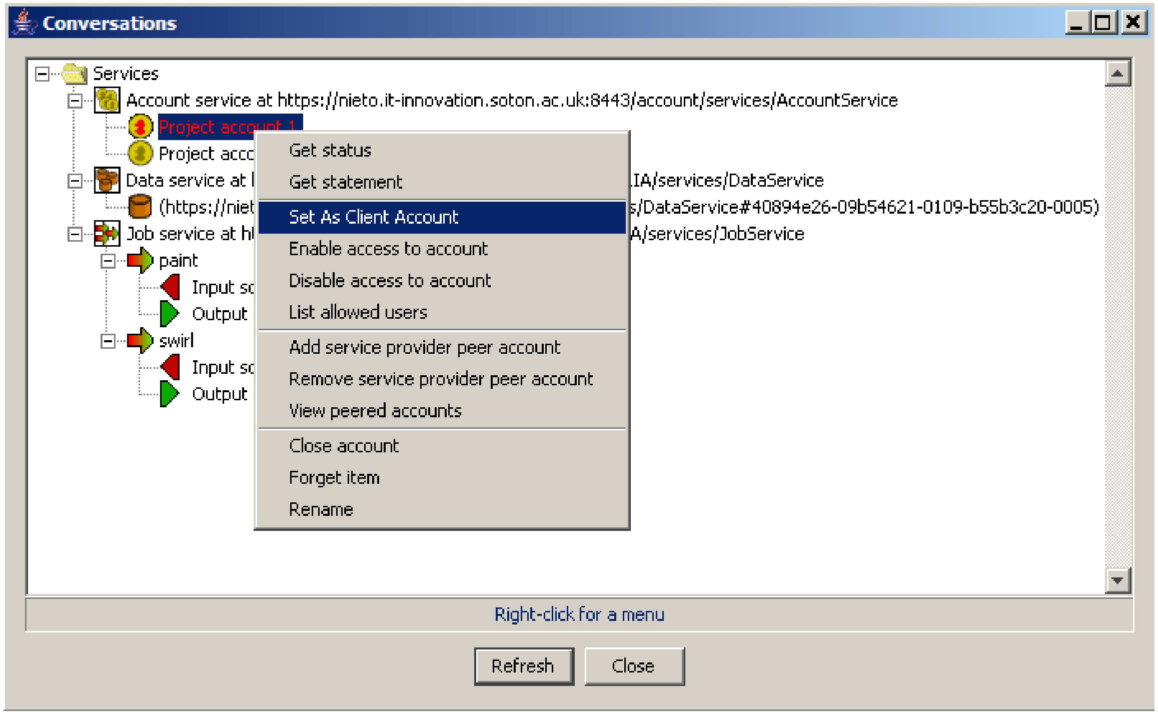


Figure 7. Client Application – Default Account

4.5 Service Provider Account Service

When a new trade account is requested, the client's budget holder provides details of the client organisation and the method of payment, as in GRIA 4, and a security policy for identifying budget holders in future.

The security policy argument was added as a NextGRID upgrade of the original GRIA account service to allow this experiment, and takes advantage of the more flexible NextGRID security components. Typically this initial policy will contain only a rule stating that the KINO manager can act as a budget holder. This rule contains the X.509 distinguished name of the KINO manager and the public key of the client's Kerberised STS which issued it. This allows for the use of short-lived X.509 certificates; the manager will get a new key-pair and certificate each day when logging in, but as long as the issuer (STS) key remains the same the manager will still have access to the account.

The service provider need not trust the client's internal X.509 STS globally, as the security software developed for this scenario now supports per-rule roots of trust. That is, the Kerberised STS is trusted only within the context of this single policy fragment.

The service provider checks the method of payment and approves the trade account. The manager can now update the trade account's policy to grant others the budget holder role too. In this scenario, they will grant their client side account service this role so that it can act on their behalf (to set up peering and to aggregate statements from multiple accounts).

The service provider requires another policy to be associated with the new account so that it knows who should be able to spend money using it. The manager at KINO gets their client account service to send a policy stating that anyone bearing a particular attribute ("*can-charge-to-account=<client account>*") can use the account, and then controls the issuing policy for this attribute locally.

Table 1 shows what the access control rules for a trade account would look like in GRIA 4. Table 2 shows the simpler rules allowed by the new security components. Note that in the GRIA 4 case, the service provider must know and trust the issuer globally, whereas in the NextGRID scenario each rule includes the authority trusted to make the required assertion.

Process Role	Security Policy
Budget Holder	Manager X.509 Distinguished Names (Subject and Issuer)
User	Animator 1 X.509 Distinguished Names (Subject and Issuer)
User	Animator 2 X.509 Distinguished Names (Subject and Issuer)
…	…
User	Animator N X.509 Distinguished Names (Subject and Issuer)

Table 1 Example GRIA 4 security policy

Process Role	Security Policy
Budget Holder	X.509 Distinguished Name, Client Kerberised STS X.509 certificate, including public key.
User	Has attribute can-charge-to-account = <client project account> Client account service X.509 certificate, including public key.

Table 2 Example NextGRID security policy

In GRIA 4 the policy requires a separate rule for each animator (1 to N) who is permitted to use the account, and this list would have to be maintained separately by the budget holder at each service provider where they wanted to use services. With the NextGRID version, there is one rule to cover all permitted animators, allowing access to be regulated via a single point of control for a token service on the client side (actually the client account service). See section 5.5 for information about how the policy is enforced.

4.6 Service Provider Rendering Service

The rendering service renders a scene description sent by the client into a video, if the client can prove that they are authorised to pay for it. The rendering service verifies that the account specified by the client is at an account service that it trusts and then checks that the user is authorised to use it.

It performs this check using a SOAP operation on the service provider's account service, passing in the X.509 certificate of the client (used to sign the client's SOAP message) and any SAML tokens provided by the client, which the rendering service does not need to understand.

The account service checks that the subject of the SAML assertion is the subject of the certificate and that the owner of the X.509 certificate has the "user" role within the account.

If these checks pass, the rendering service renders the film and bills the usage to the account. It may also charge the user for uploading and downloading any large data files.

5 Security Components

The NextGRID component model, shown in Figure 8, contains a Policy Enforcement Point (PEP) within the service, a Policy Decision Point (PDP), and a token validation service. In our experiment, the token validation service is the service provider's account service, which evaluates the SAML token provided by the client.

The Kerberos components developed for NextGRID by KTH follow the standard pattern for a local WS-Trust service, exchanging one token type (Kerberos tickets) for another (X.509 certificates).

5.1 The Account Services

A service provider account service was developed for the GRIA project [2]. This provided features for requesting and approving accounts, recording client's debts and generating statements. It was protected using the PBAC system, allowing clients to update the access control policies for accounts to grant and deny users access to them.

For NextGRID the account service was modified so that it could also be deployed within a client organisation and used to manage access to other accounts rather than having charges recorded against it directly. This involved the following changes:

- Each client account maintains a list of trade accounts with which it is peered. When adding a peer to a client account, the account service contacts the remote service and updates the trade account's policy to permit anyone with a particular attribute to charge actions to the account.
- An operation allowing users to obtain an assertion stating that they have the required attribute was added. The user's software queries the target service (the rendering service in our example) to get a list of account services trusted by the target service. This list is passed to the client account service so that it can select the appropriate peer account.
- Access to the token issuance operation is protected using PBAC, which enforces the policy set by the client account's manager.
- Statement aggregation allows a client account's manager to collect together charges from all the peered trade accounts for a particular project account. The combined statement shows the charges to the project by each animator.

Only one change to the service was required to support its use at the service provider:

- The operation for checking whether a user should be permitted to use the account was extended to allow a SAML assertion to be passed in. The account service passes the assertion to PBAC for checking.

5.2 Dynamic Policy Components

The PEP and PDP components used in the experiment are based on the PBAC system developed for the ARTEMIS project by IT Innovation.

The original PBAC system was modified to provide the initial NextGRID dynamic Policy Decision Point (PDP) and Policy Enforcement Point (PEP) components. In the NextGRID component model, the PEP is considered to be embedded in the target service (see Figure 8).

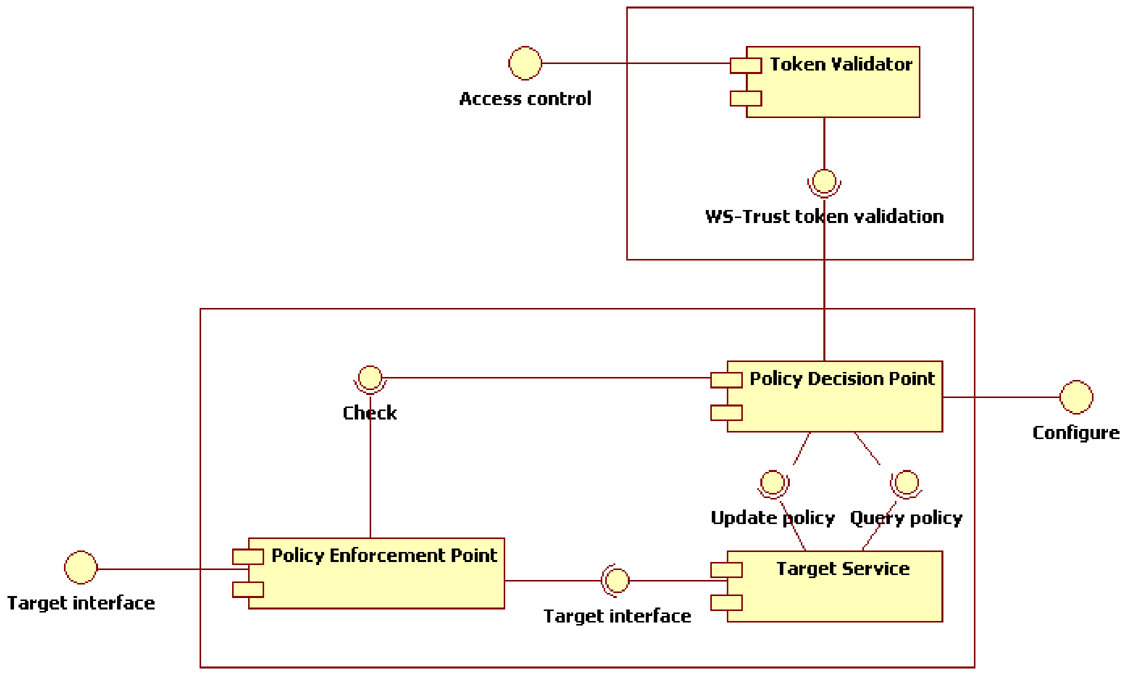


Figure 8. NextGRID security component model

The following sections detail how these components were modified to support the NextGRID security architecture. To test these new features, the GRIA client and services have also been modified to use the new components and to use WS-Addressing Endpoint References and headers to identify contextualised resources.

5.2.1 Policy Decision Point

A new dynamic authorisation Policy Decision Point component (the PBAC PDP) has been developed by IT Innovation as part of the ARTEMIS project:

- The PDP will grant or deny access to a protected resource based on the set of process roles the client has. Whether a particular process role is permitted to perform the operation given the current state of the resource depends on the service policy (set by the service administrator using an XML configuration file).
- Whether a user has a particular process role depends on the resource's dynamic policy, which is updated by the service.
- Updates to the dynamic policy are performed by the service, typically at the request of a user or the service administrator. Access to all service operations, including those which update the policy, is controlled by the PDP.

The PDP has been extended to provide greater support for the highly dynamic federation scenarios required by NextGRID, which follow the pattern shown in Figure 8:

- The PDP provides support for new methods of federation using SAML assertions. This has been tested using an accounting scenario, as described above.
- Roots of trust are now per-policy fragment, reducing the amount of due diligence required from service providers when establishing new federations. For example, credit-worthiness must still be checked when approving a new account, but it is not necessary to establish a high level of trust in the client's CA, because the CA now only controls access to that single account. This allows for better support of "fast VOs".

5.2.2 Access Control Rules

Each PBAC rule has four parts:

- The distinguished name of the subject to whom the rule applies.
- The distinguished name of the issuer of the subject's certificate.
- Whether matching the rule grants or denies the role.
- Which process role is to be granted (or denied).

These policy rules have been extended in two ways for NextGRID:

- Each rule includes the X.509 certificate of a party trusted to assert that the user has some identity or attribute, instead of containing only the issuer's Distinguished Name (DN). The public key within the certificate is used to validate any certificate or assertion provided by the client cryptographically.
- A rule can now specify a SAML attribute (name, value) pair that the user must have. Previously, a rule could only give the distinguished name of the subject.

5.2.3 Policy Enforcement Point

The PBAC Policy Enforcement Point (PEP) asks the PDP for a decision based on the resource context, the requested SOAP operation and the user's security tokens, all of which it extracts from the SOAP message. The PEP also uses the PDP to enforce transactional semantics between authorisation decisions and subsequent policy updates.

For NextGRID, the standard Axis security handler (WSS4J) has been modified so that it no longer checks roots of trust itself, but simply passes them to the PEP for verification. The PEP has been modified to pass the security tokens themselves to the PDP, rather than just passing the distinguished names within the X.509 certificate.

The PEP currently only extracts X.509 tokens, not SAML tokens, from the WS-Security header. PEP SAML support was not necessary for this experiment as no SOAP operation requires a SAML token to be presented before being invoked:

- The rendering service allows anyone to invoke the operation to start a new job. Checking of the provided tokens is done within the operation via the attempt to bill for the requested job via the service provider's account service.
- The PEP for the account service only checks that the rendering service is permitted to invoke the operation to check a token; the tokens themselves are checked inside the account service's check operation.

In future, it is likely that the mechanisms used here to submit new jobs will also be used to control access to operations which are accessed directly. For example, access to the operation to generate an account's statement may be granted to anyone with the "KINO manager" attribute. In this case, the PEP will need to extract the SAML tokens and pass them to the PDP when the operation is called.

5.3 WS-Addressing Profile

The WS-Addressing[14] specification defines an EndpointReferenceType (EPR) as containing:

- A URL for the endpoint of a service.
- A number of reference parameters.
- A number of meta-data elements.

It suggests that when accessing the resource named in the EPR, the service should promote each reference parameter to be a top-level SOAP header element. There are obvious risks here, which we detail in Appendix A along with our guidelines for the safe use of WS-Addressing. We applied these general guidelines to this experiment as follows:

- Each service has a white-list of account service endpoints which it can use (typically there will only be one). Any other endpoint address will be rejected.
- The only reference parameter that is forwarded is the resource context identifier, which cannot cause confusion since the account service knows that this is a reference parameter and will treat it appropriately.
- ReplyTo and FaultTo elements are ignored.

5.4 Kerberos – X509 Security Token Service

A Security Token Service (STS) is a Web Service that issues and validates security tokens as defined by the WS-Trust specification.

This is achieved either by converting a security token that a user currently has with the one which is needed to access a desired service or by creating a new security token such as a SAML attribute assertion. For example, if the user holds a Kerberos ticket asserting their identity, but the target service needs an X.509 certificate, the Kerberos ticket can be presented to an STS, and the service issues an equivalent X.509 certificate asserting the same identity of requester.

The development of the token service for this phase of the experiment was specifically focused on Kerberos-PKI interoperability, converting identity tokens only. However, the token service design is architecturally open and able to handle attributes other than identity and other token formats such as SAML attribute assertions. The STS implementation is based on a Kerberized Certification Authority (KCA), introduced in the PM06 P5.4.1 experiments which issues short-lived user certificates based on the user's Kerberos identity.

The KCA has its own certificate signing key, and a long-lived, self-signed CA certificate, which is not widely known. A relying party must trust the KCA's own certificate in order to verify user certificates issued by it. Thus, the KCA does not directly address the problem of establishing trust between domains, but it does provide a good starting point for experiments involving identity mapping and trust federation between domains including a translation between different authentication mechanisms.

An STS should also be able to validate security tokens, enabling Grid entities to perform cross-organizational interactions when they:

- Do not trust to the issuer of security tokens directly (lack of direct trust relationship between different organizations).
- Use different security technologies and different attribute definitions.
- Use the same security technologies but with different attribute definitions.

In our development process, we encountered several limitations in both the proposed standards (WS-Trust specification) and its trivial WSS4J implementation. The current implementation of the token service components has been done based on some extensions to the WS-Trust specification and some modification and extensions to the WSS4J implementation of WS-Trust.

In the following, we describe the general design of the STS for both issuing security tokens in different formats and for issuing attribute assertions.

5.4.1 Design Diagram

Figure 9 illustrates how a STS can be used to obtain an attribute assertion or an identity security token for a Kerberized user. The ultimate goal of this approach would be using the WS-Trust specification for obtaining security tokens at any level of interoperability, even at the local Kerberos layer, as the Kerberos KDC conceptually implements what the WS-Trust specification calls a Security Token Service: It generates security tokens (e.g., Kerberos TGT) in exchange for other tokens (e.g., a user-name and password). This implies that a fully WS-Trust compliant model can be achieved by leveraging the web service security specification and the mechanisms described by the WS-trust specification even at local Kerberos interoperability level. However, in this experiment in order to simplify the implementation we use the traditional Kerberos mechanism for obtaining Kerberos tickets instead of using the WS-Trust specification.

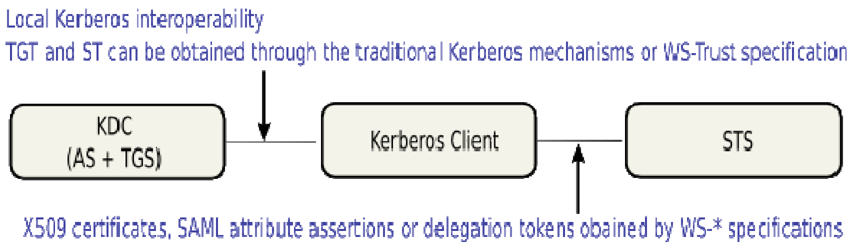

Local Kerberos interoperability
TGT and ST can be obtained through the traditional Kerberos mechanisms or WS-Trust specification
KDC
(AS + TGS)
Kerberos Client
STS
X509 certificates, SAML attribute assertions or delegation tokens obained by WS-* specifications

Figure 9 General Kerberos-PKI interoperability protocol

The following section describes in detail the functionality of a STS for issuing and validating identity tokens in different formats.

5.4.2 Security Token Issuance

In the case that a security token is not in a format or syntax understandable by a recipient, the STS should be able to convert it into a comprehensible security token for the recipient (for example, the STS converts a security token in form of a Kerberos ticket into an X.509 certificate). One important consideration is that the recipient should be able to build a chain-of-trust from its own trust anchors (e.g. its X.509 Certificate Authority, a local Kerberos KDC, or a SAML Authority) to the issuer or signer of a token. Then the signature of STS should be verifiable for the ultimate recipient of security token. It implies that the recipient and the STS need an established trust relationship directly or through the intermediaries.

In Figure 10 we illustrate a use case which leverages a STS for issuing security tokens in a different format. We particularly consider a Kerberos-based requester who needs to communicate with a PKI-based server. For this identity token mapping to take part the requester needs to identify a security token service that it trusts for converting his security token (e.g., the one which is closely tied to his home organization’s KDC).

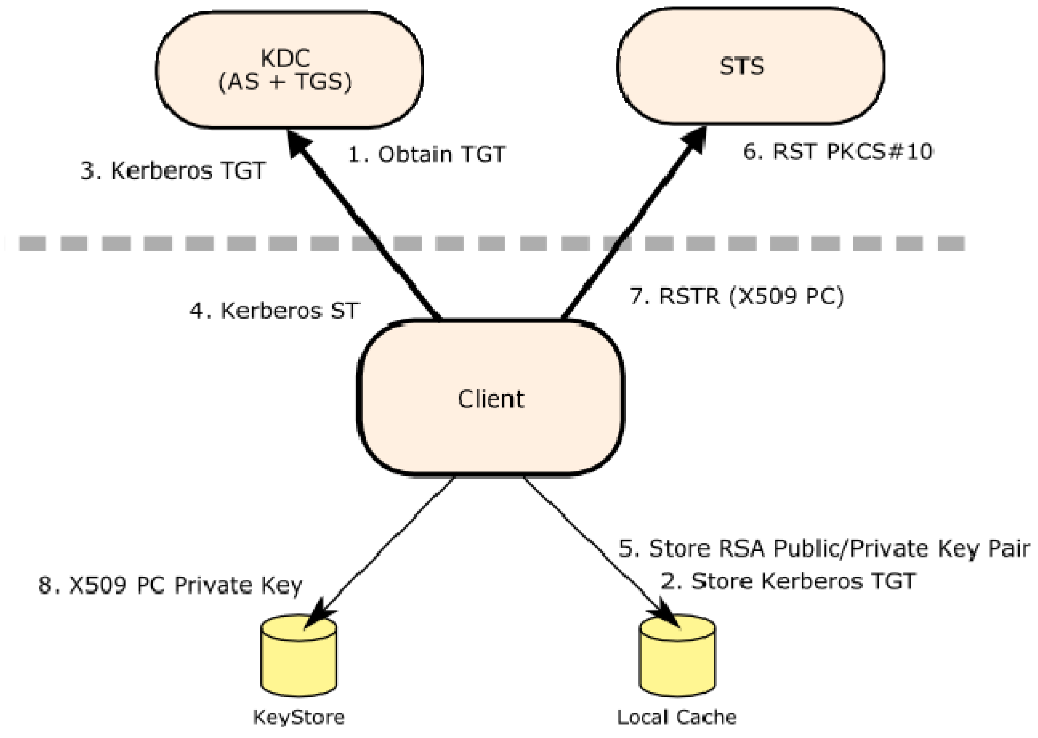


Figure 10 Client Kerberos System Architecture

The requester then communicates with the STS as follows:

1. The requester, through the "Kerberos login" obtains a TGT.
2. The TGT is stored in the local cache of user's home directory. This should be done using the traditional Kerberos mechanisms.
3. The client fetches the TGT from the cache and sends that to the Ticket Granting Service part of KDC to obtain a Service Token (ST) for further communication and authentication with the STS. This ST token is obtained from the KDC using the traditional Kerberos mechanisms.
4. The ST allows authentication to the STS and can be used as a proof-of-possession token which contains the session key encrypted for the client. Now using the shared secret in ST, all the messages between the requester and the STS can be signed and encrypted as described in the WS-Security specification.
5. The client generates an RSA public/private key-pair and stores it in the local cache on the local machine.
6. The client obtains the public half of generated key and generates a PKCS#10 certificate request. It further generates a request message (RequestSecurityToken message) for an X509 security token using the mechanisms based on the WS-Trust specification and sends the message to the STS. The client attaches the generated PKCS#10 as the "claims" part of this request according to WS-Trust specification. The client encrypts the message and provides a signature on the request using the session key obtained from the ST. The STS receives the request message, uses the ST session key to verifying the integrity of the incoming request and decrypting the message. STS detaches the claims provided by the requester from the request message and generates a short-lived X.509 proxy certificate from the PKCS#10 certificate request.
7. The STS then generates a response message (RequestSecurityTokenResponse) and attaches the short-lived X509 certificate to it, as described by the WS-Trust

specification. The STS also uses its private key to sign and encrypt the RSTR message. It finally sends the response to the client.

8. The client receives the message from the STS, verifies the signature, decrypts the message and stores the X509 proxy certificate along with its associated private key to a configured Keystore on client side.

5.4.3 SOAP Messaging

The SOAP messaging for issuing and validating tokens between clients and the Kerberos STS is fully compliant with the WS-Trust specification. The detailed description can be obtained from the "issuance binding" and "validation binding" of the WS-Trust specification released February 2005 [8].

In brief, the general approach described by the WS-Trust specification for issuance and validation of security tokens is based on a simple request/response message pair between the requester and a STS. The requester makes a "RequestSecurityToken" message which includes a <RequestType> element with a proper content (Issue/Validate) and a <TokenType> element to specify the type of requested security token ("wsse:Kerberosv5ST", SAML or "wsse:X509V3"). The requestor sends this request to the STS. In response, the STS sends back a "RequestSecurityTokenResponse" SOAP message that contains the requested security token or the results of the validation process (Valid/Invalid). Each request message should also include all security tokens and claims that can be used as proof-of-possession for the requester.

5.4.4 Identity Mapping Token Service Implementation

The STS implementation is built on several open standards including SOAP, WS-Security and WS-Trust. In general, the STS will be implemented to perform the above functionalities by exchanging a set of secure request/response SOAP messages as described by the WS-Trust and WS-Security specification.

The STS components are a set of Java API's which can be leveraged for implementing a stand-alone service to be invoked by clients for issuing and validating security tokens. These API's can also be used as ready-to-use libraries for developers as well. In the following we describe some of the key components of STS component:

- STS Manager: The STS manager is a set of Java API's for handling SOAP messages. This facilitates generating and handling request/response messages that clients send to the STS for when issuing and validating security tokens.
- Proxy certificate generator: The Proxy certificate generator is a set of Java methods, which facilitate generation of X509 proxy certificates. Proxy generation has support for generating limited and restricted proxy certificates from a certificate request or a key pair.
- X509 certificate manager: The X509 certificate manager provides a set of Java libraries for generating, validating and signing X509 certificates. It also provides a set of functions to generate user certificate requests, and key pairs in different formats such as PEM encoded format.
- Configuration manager: The configuration manager provides an API to support customized configuration of parameters for issuing and validating security tokens. It provides configuration parameters through property files for both sides of the interaction.

5.5 Cryptographic Attribute Assertions

The tokens issued by the client account service take the form of signed SAML assertions. In this section, we describe the parts of each token and the rules used to validate them.

An example of a SAML token issued by the client account service can be found in Appendix B: Example SAML token.

5.5.1 SAML Attribute Support in Dynamic Policies

When a trade account is added to a project account's list of peers, the project account service contacts the trade account service and adds a match rule saying that anyone with a given attribute may use the account. The full match rule contains these fields:

- The attribute name ("can-charge-to-account").
- The attribute value (the resource ID of the client's project account).
- The process role affected ("user").
- Whether the role is to be granted or denied on a successful match.
- The X.509 certificate of the issuer of the assertions (the client account service).

The attribute is specified by the client and could be anything. For example, the client might not wish the service provider to see their internal account identifier. In that cause, they could use some randomly generated string instead.

Notice that the rule has only one trusted authority (the client account service). It would be possible to specify that both the account service and the client's Kerberised STS are to be trusted. The client account service could issue assertions of the form:

Subject "name" can-charge-to-account "account"

By trusting the Kerberised STS to associate the public key with a name, and the account service to associate the name with the attribute, the service provider's account service could deduce that the user should be trusted. However, for this experiment we decided to keep the match rules simple by having only one issuer.

Having only one authority means that the user's public key must appear in the SAML assertion, otherwise there is no way to know whether the sender of the message really is the subject. However, having the public key alone is not sufficient, since the account service would only be able to record that "someone authorised to use this account" was spending money. This does not provide the client manager with sufficient information if someone is abusing the account.

Therefore, the SAML assertions issued by the client account service use the client's full X.509 certificate as the subject:

Subject <X.509 certificate> can-charge-to-account "account".

The account service is asserting that the holder of the public key is permitted to use the account *and* that they have the given distinguished name. The SAML assertion is signed using the client account service's key.

5.5.2 Consistency of SAML Assertions with X.509 Certificates

There is a further problem, which is that the client software and the rendering service shouldn't have to understand SAML tokens. Only the issuer and the validator of the tokens should have to understand them at all. Therefore, the account service also checks that the certificate in the SAML assertion is identical to the one used to sign the SOAP message (which is forwarded to it by the rendering service). This allows the rendering service to process only X.509 tokens, which it already understands, while still ensuring that the audit trail written by the rendering service is using the correct name.

Without this check, a user authorised to spend money on the account could get a SAML assertion for themselves and then generate an X.509 certificate claiming they were someone else. The job service would correctly deduce that they could use the account from the SAML token, but the use would appear to be from someone else.

Note that as at the service provider's account service, the access control rules on the client account service must also ensure that not only do only authorised users gain access, but that they do so with a valid X.509 certificate (so that it can be included in the assertion).

For example, if the manager set the policy to issue billing tokens to any user at all (and relied on the fact that only users within the organisation could reach the server), then users would be able to have any name appear on the final account statement as the user performing the work. However, any reasonable access control policy for billing token issuance would include the Kerberised STS issuer certificate, so this isn't generally a problem.

5.5.3 Sending Tokens via SOAP

The SAML tokens are digitally signed, but when serialising the XML extra whitespace can easily get added, invalidating the signature. For this experiment we avoided the problem by sending the whole SAML document as an encoded string argument. In future, we hope to be able to send it in the WS-Security header in a way that still prevents it from being accidentally modified.

5.5.4 Token Lifetime

Both the SAML tokens issued by the client account service and the X.509 tokens issued by the Kerberised STS have expiry dates set just a few hours in the future. The expiry date of the X.509 token is checked by the security handler of the rendering service (WSS4J), while the expiry date of the SAML assertion is checked by the OpenSAML library used by the PDP.

The Kerberised STS and client account service certificates are used as issuers in the PDP's match rules and so should normally be long-lived. When these services generate a new key-pair additional match rules containing the new certificates must be uploaded. After a while, the old match rules can be removed. The PDP does not currently check for expired issuer certificates in match rules.

It is a general principle of digital security systems that the strength of security depends on the lifetime of a token or a trust anchor. For high security, tokens and/or revocation lists should have short lifetimes, so that an intruder that manages to compromise a token or issuing system will only have a short time in which to work. The price for this is a performance overhead, as users have to get new tokens every time they want to use a secure service, and the service may have to check frequently with the token issuer to validate tokens or obtain revocation lists.

At this stage, the lifetime of tokens is based on the underlying Kerberos tokens. However, in one of the NextGRID WP7 applications (Option Pricing led by NEC), an experiment is planned to test the trade-off between security and performance for low-latency services. This is likely to involve adjusting the token lifetime, so tokens can be pre-fetched and cached, rather than weakening security in more brutal ways by reducing key-lengths, for example.

6 Conclusions and Recommendations

In this report we have described the NextGRID security components that were developed in the third six months of the project in Task 5.4 to explore issues relating to token issuance, exchange and validation in dynamic grids. We have performed an experiment to validate these components, based on a simple test scenario from a NextGRID end user (KINO's Digital Media Use Cases).

6.1 Dynamic Policy Components

IT Innovation has extended the PBAC security system to support the NextGRID security model. This involved adding support for issuing and validating SAML assertions and for per-policy trust anchors.

To test these new dynamic policy features, the GRIA client and services have been modified to use the new components and to use WS-Addressing Endpoint References and headers to identify contextualised resources. A new client-side accounting service has been developed which issues SAML assertions to users asserting that they can use an account. These assertions are checked by the PDP in the service provider's accounting service, using a matching policy uploaded dynamically by the client organisation. Access to the client accounting service is also protected using the new security components, including access to the token issuing operation.

We have shown how these components improve the manageability of federations within the grid, by permitting authorisations to be managed centrally within each client organisation.

The two accounting services developed in this work will also be used in M24 experiments focussing on how they can be used to provide interoperable accounting mechanisms in different infrastructures.

Support for "fast VOs" has been improved by specifying the full X.509 certificate of a trusted issuer in each policy rule, including the issuer's public key. This removes the need for service providers to trust new certificate authorities globally, allowing new federations to be established more quickly by reducing the amount of due diligence required from service providers when establishing new federations. For example, credit-worthiness must still be checked when approving a new account, but it is not necessary to establish a high level of trust in the client's CA, because the CA now only controls access to that single account.

Work on profiling WS-Addressing to avoid certain security problems has led to a simple initial policy of only promoting white-listed reference properties to SOAP headers and only using trusted addresses.

6.2 Trust Federation and Mapping

Development of standardized versions of the security components from the PM06 experiment has been the main focus of KTH's work in this period. The result has been a Security Token Service (STS) developed for issuing and validating security tokens, which is mostly achieved along the lines of WP4 / WP5 joint security experiment plan, and standardization of PM06 security components.

The STS components are a set of Java APIs which can be leveraged to implement a stand-alone service to be invoked by clients for issuing and validating security tokens. These APIs can also be used as ready-to-use libraries for developers as well. Development of the STS has aimed at providing two main functionalities: Issuance and Validation of security tokens in a cross organisational interaction based on the WS-Trust specification.

In our current development the "issuance" is mainly achieved by converting a security token that a user currently has (e.g. a Kerberos ticket) to one which is needed to access a desired service (e.g. an X.509 certificate) in a standard fashion proposed by the WS-Trust specification.

For this implementation we have followed our previous approach in Trust federation and identity mapping with the focus on Kerberos-PKI identity mapping mechanisms. The STS implementation is based on a Kerberised Certification Authority (KCA) and the kx509 protocol, introduced in the PM06 P5.4.1-P5.4.2 experiments. However the implementation in this phase is fully compliant with the specification proposed by WS-Trust for issuing security tokens.

It should be noted that although for this phase we only provide an interoperable identity mapping mechanism from Kerberos identity into X509 certificates in a standard fashion, the design and implementation of STS, are architecturally open to handle other token formats such as SAML and even attributes rather than only identities.

6.3 Answers to Architectural Questions

Based on the results of this experiment, we can provide the following answers to the questions posed in section 2.3.

Addressing

- 5.1) Does WS-Addressing address security considerations of all parties (for e.g. intermediaries) involved in the SOAP processing chain?
    - WS-Addressing alone does not provide the required level of security. It is necessary to profile the specification so that only certain addresses and reference parameters are passed on to third parties.

Authentication

- 33) What other authentication mechanisms (e.g. username-password) should be supported by the NextGRID architecture?
    - We need to support existing local authentication mechanisms such as Kerberos. This can be done using WS-Trust Security Token Services.

- 34) What mechanisms should be used for identity federation/trust federation (federated token issuance and verification)? How is this related to emerging standards capable of dealing with identity such as WS-Trust, WS-Federation, etc?

  - This report describes a federation mechanism which uses services in both client and service provider organisations to federate identity and trust. While WS-Trust and WS-Federation can be used for token issuance and validation, additional interfaces are still required to update the issuance and validation policies. In the current experiment this was dealt with by using scenario-specific (accounting) management interfaces. Future experiments with WP4 will continue to investigate how this can be done in a generic and standard way (for example, by profiling or extending WS-Trust).

- 35.1) Is a digital signature over the message body only sufficient?

  - No. Any headers which affect the interpretation of the message (such as a WS-Addressing resource context identifier header) must be signed together with the body, by a single signature. Otherwise, an attacker can modify an invocation to target a different resource of their choosing. For example, the attacker could destroy any resource owned by the client when the client tries to destroy some particular resource.

Authorisation

- 36.5) What is the model for dynamic authorization? Are their base specs that we can work from or use here (WS-Policy?)

  - Services can provide a SOAP interface to allow dynamic updates to the service's policy. These interfaces should themselves be protected by the policy. WS-Policy has not been investigated by this experiment.

- 37) How do dynamic authorisation models relate to dynamic resource management and virtual organisation management?

  - Authorisations can be made in the context of individual resources (granting access to a particular job, for example), or they can be done in some larger context, such as an account which represents a federation between two organisations. This experiment used the PBAC system for access control; a follow-up experiment involving WP4 as well will see how well this fits with WS-Policy/Generic AAA. [15]

- 38) Do we need a process role vocabulary coming out of NextGRID?

  - No. The client can specify the mapping from their internal roles to process roles at the service provider. For example, a policy may state that anyone with the attribute "animator at KINO" has the process role "user" in the context of a particular remote account. The service provider does not need to understand the original meaning of "animator at KINO" to enforce this policy.

Trust

- 53.2) How should trust be represented and analysed in (business) processes; can the methodology used by GRIA be developed to address this?

- In this experiment, credit worthiness is the normal basis for a service provider to trust a client. The business models developed by GRIA have been successfully extended to support this scenario.

- 54) How can trust be managed in a dynamic business context, including VO lifecycles, and delegation mechanisms?

  - Trust is established using traditional business processes, based on existing mechanisms such as credit checks. Once a root of trust is established it is linked to cryptographic key material allowing delegation operations to be performed without further intervention by the service provider. During the lifetime of the bipartite relationship, standard business accounting practices such as regular billing, auditing and the enforcement of credit limits are used to manage trust. VOs could be established quickly using similar mechanisms, and this will be explored further in a later experiment.

7 Appendix A: WS-Addressing Profile

WS-Addressing provides a standard way to encode context identifiers. However, WS-Addressing also allows a sender to induce a recipient to transmit a (signed) message to a third party, and to include SOAP headers of the sender's choosing under the signature of the intermediary. This is unacceptable in an industrial Grid environment, and it is proposed that NextGRID should recognise WS-Addressing specifications for constructing and addressing messages only for a white-list of addresses and context identifier elements.

7.1 *Description*

WS-Addressing was originally designed as a way to convey connection state or context in SOAP messages, emulating the contextualisation mechanism provided by HTTP headers in conjunction with stored cookies. This makes it possible to contextualise message exchanges in a similar way, independently of the transport used.

To support this, WS-Addressing provides a schema for an "endpoint reference" (EPR), which can be used to specify where the response should be sent, and the context headers that should be attached (specified within the endpoint reference through "reference parameters"). The endpoint reference can thus convey a fully qualified and contextualised message destination, and is starting to be used not only to specify the headers needed in a reply, but also to specify onward routing destinations and headers. In this case, the reference parameters are supposed to be treated as opaque by the initial recipient, who is obliged to simply transcribe any reference parameters into the SOAP header of the forwarded message.

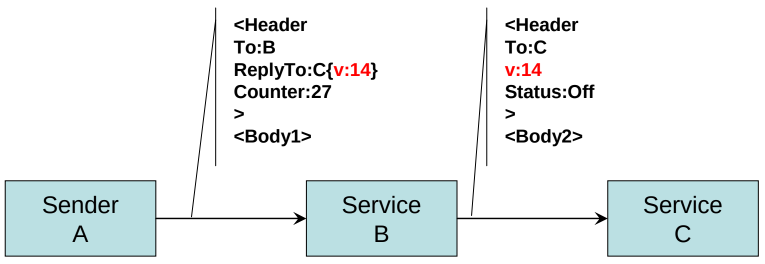


Figure 11. WS-Addressing using ReplyTo with reference parameters

Thus in Figure 11 the ReplyTo header in the message from A means B should insert the header “<v>14</v>” into its response (abbreviated to “v:14” in the diagram), and send this response to C (not A). Note that B can insert their own headers (e.g. “<Status>Off</Status>”) if desired, but C cannot tell that this header was chosen by B while the other was not.

Several contributors objected to this behaviour, pointing out that this mechanism can force a recipient of a SOAP message to send a message to an arbitrary destination with arbitrary headers. See for example the objection submitted by Anish Karmarkar and others on 19 May 2005, which describes a number of unwanted consequences of opaque header generation in the SOAP binding of WS-Addressing reference parameters. Suppose C in Figure 11 was a service provided by your bank, B was one of your trusted suppliers and A is some other client of B. If A sent a message to B including reference parameters “<Account>Your Account</Account><Debit>£150<Debit/>”, would you want B to construct the WS-Addressing mandated response, sign it, and send it to your bank (Figure 12)?

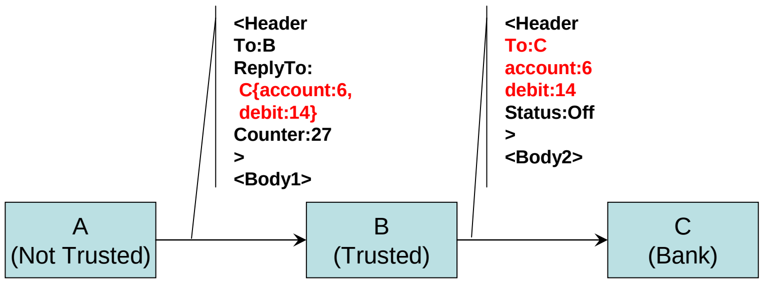


Figure 12 Malicious use of WS-Addressing

Several solutions were proposed, but the one adopted in the Candidate Recommendation of Aug’05 simply mandated that a WS-Addressing type attribute be used to indicate header elements generated from an EPR, and that the “must understand” attribute never be set for such headers. The specification also allows a service to refuse to process an EPR fully if it is not satisfied that to do so would be safe. For example, the specification suggests that this might be determined by authenticating the issuer of the EPR, and processing it only if it comes from a trusted source.

Finally, note that WS-Addressing only forces services to construct a response from an EPR when it is included in a WS-Addressing ReplyTo or FaultTo header element. EPR can also be conveyed to or from a service by other means, in which case the recipient may (but is not required) to use them for contextualised addressing of responses or subsequent messages as though they were sent in the WS-Addressing headers. In practice, this is how WS-Addressing is most often used in Grids. Of course, it doesn’t matter how the EPR is transmitted, the problems described here arise whenever an EPR is used to generate context headers for an outgoing signed SOAP message.

7.2 Issues

Our analysis has shown that the solution from the August 2005 specification is not satisfactory. Firstly, allowing recipients to ignore WS-Addressing obligations at will means the specification cannot guarantee interoperability. Furthermore, the above attack using reference parameters can also be committed using Address elements in an EPR. For example, the address for C in Figure 12 could be set (by A) to “https://yourbank.com/debit?amount=14”, causing B to send a message that achieves the same effect. A may even be able to gain access to protected information by specifying C=“http://yourbank.com”, causing B to send a message to C without transport-layer security.

Unfortunately, signing over messages and headers does not really solve the problem. If A signs its message to B, this would allow B to tell it came from A, including the ReplyTo header. B may then decide to go ahead and construct a response as specified by A's ReplyTo header, converting reference parameters into full SOAP header elements and sending the resulting message to C. Only B can then bind this header to the message by signing over it, so C can only tell that the response came from B. C cannot tell if B understood the header elements, nor who originally specified them. Thus B's signature can no longer be taken to mean B intended the meaning implied by all parts of its message, but only that B constructed the message to C, including some headers specified by a third party whose identity cannot be verified by C.

If C wanted to know which headers B really did understand, B could insert more attributes, but this assumes that C recognises the attributes. The attacker can simply specify the address of a service which does know about this new convention.

B could insert a second message-level signature covering only the headers it inserted, but this would conflict with the WS-Interoperability Basic Security Profile, which says there should only be one signature to identify the sender of a message. (Multiple signatures would in any case create a semantic interoperability problem, as applications would have to decide the meaning of each signature). B could of course apply a single signature covering only the headers it originated or understands. However, this would leave the remaining "opaque" headers from A open to tampering en-route between B and C.

7.3 Profiling Options

Given this, there seems to be only two "reasonable" policies to take regarding the use of WS-Addressing:

- Recipients (B in Figure 12) can constrain where a response generated from an EPR can be sent (e.g. only back to the original sender) so that those who trust them (e.g. C) cannot be induced to act on messages specified by a possibly less trusted third party (e.g. A).

- Recipients can constrain which reference parameters or HTTP arguments from an EPR they are willing to handle, e.g. by using a blacklist of "unsafe" opaque element names, or a whitelist of "understood" element and argument names.

It is also possible to apply both policies at the same time, constraining both the destination and the header content of any response messages dictated by WS-Addressing.

The second policy would allow A to specify some information in the message from B to C, but B would be able to filter out elements B thought unsafe (using a blacklist) or that B did not understand (using a white list). Clearly, a blacklist provides greater interoperability, as services and clients can use any elements not on the blacklist. However, a blacklist is less secure as it depends on the service operator identifying and blacklisting ALL unsafe elements, or else weakening the guarantees implied by their signature over responses. With a white list, B could sign over the whole response message and signify by this that they understand and vouch for its entire content. It may be significant that in HTTP a white list is used restricting HTTP context headers to a small number of cookie types.

Clearly, if an EPR is sent by some other means (not in a WS-Addressing ReplyTo or FaultTo header), the recipient is not obliged by WS-Addressing to do anything with it. However, the above policies can also be used if such an EPR were used to generate new messages in the conventional fashion.

7.4 Recommendations

It is recommended that where NextGRID adopts WS-Addressing, using EPRs to encapsulate resource context identifiers, we should avoid using the ReplyTo and FaultTo headers, and impose a white list defining EPR reference parameters that are understood and can be elevated to full headers in messages addressed using EPR.

Addresses containing query strings (“scheme://host/resource?query”) should be rejected. Ideally, only white-listed addresses should be permitted. For example, when starting a rendering job the client specifies the address of the service provider’s account service with an EPR, and this address is checked against the service’s white-list of trusted account services.

However, this is not possible for all uses of EPRs. When the client runs the job they must also give an EPR with the address of the job’s input data. The input data can be at a data service not known to or controlled by the rendering service’s organisation, and therefore its address cannot be checked against a white-list.

8 Appendix B: Example SAML token

This appendix shows the format of the SAML tokens issued by the client account service. The assertion contains a condition (the expiry date), an attribute statement, and a signature over the assertion.

The attribute statement contains:

- The subject’s X.509 certificate, containing the public key used to check that the sender of a SOAP message really is the subject of the assertion, and the distinguished name of the subject (which appears on the account statement and in the service logs).
- The attribute which the account service is asserting the user has (authorisation to charge to this account).
- The “NameIdentifier” is not used; it exists only to work around a minor bug in the OpenSAML library we use.

```
<?xml version="1.0" encoding="UTF-8"?>
<Assertion
       AssertionID="c814e3493913f3267c36f21b695a212a"
       IssueInstant="2006-03-08T13:11:29.760Z"
       Issuer="https://banyuls.it-innovation.soton.ac.uk/account/services/AccountService"
       MajorVersion="1" MinorVersion="1"
       xmlns="urn:oasis:names:tc:SAML:1.0:assertion"
       xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion"
       xmlns:samlp="urn:oasis:names:tc:SAML:1.0:protocol"
       xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
       xmlns:xsd="http://www.w3.org/2001/XMLSchema"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

  <Conditions NotOnOrAfter="2006-03-08T15:58:09.760Z"/>

  <AttributeStatement>
    <Subject>
      <NameIdentifier NameQualifier="-">-</NameIdentifier>
      <SubjectConfirmation>
        <ConfirmationMethod>urn:oasis:names:tc:SAML:2.0:cm:holder-of-key</ConfirmationMethod>
        <ds:KeyInfo>
          <ds:X509Data>
            <ds:X509Certificate>
              MIIDLjCCApegAwIBAgIBAjANBgkqhkiG9w0BAQQFADBnMQ8wDQYDVQQDEwZXZWFrQ0ExCzAJBgNV
              […]
              bd1pndURPexeiqPpLTSrOr2YgTBAnd76j4z7ZGxtC0QdQNq/NTi94Gv53LppCU9ELs0qGir1M256
              eTQtxlrlGf4t29e3mV3ZYlz0aXg=
            </ds:X509Certificate>
          </ds:X509Data>
        </ds:KeyInfo>
      </SubjectConfirmation>
    </Subject>

    <Attribute
       AttributeNamespace="http://www.it-innovation.soton.ac.uk/2006/grid/account"
       AttributeName="can-charge-to-account"
       xmlns:xsd="http://www.w3.org/2001/XMLSchema"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
      <AttributeValue xsi:type="xsd:string">40894e38-09b14cc1-0109-5…a-0001</AttributeValue>
    </Attribute>
  </AttributeStatement>

  <ds:Signature>
    <ds:SignedInfo>
      <ds:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
      <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#dsa-sha1"/>
      <ds:Reference URI="#c814e3493913f3267c36f21b695a212a">
        <ds:Transforms>
          <ds:Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
          <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#">
            <ec:InclusiveNamespaces
                PrefixList="code ds kind rw saml samlp typens #default"
                xmlns:ec="http://www.w3.org/2001/10/xml-exc-c14n#"/>
          </ds:Transform>
        </ds:Transforms>
        <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
        <ds:DigestValue>lXa6YviQGR1oyZZVhW3yMszQw1E=</ds:DigestValue>
      </ds:Reference>
    </ds:SignedInfo>
    <ds:SignatureValue>PmhRcebZmw/dBs85XapX…VRiisK0X5/ipR1zDZFW0g==</ds:SignatureValue>
  </ds:Signature>
</Assertion>
```

9 References

For more information, see:

1. M. Ahsant, J. Claessens, T. Leonard, M. Surridge and F. Wan, NextGRID Dynamic Security Experiment Plan, Appendix to T4.6, T4.7, and T5.4 outputs.
2. See the GRIA project website at http://www.gria.org.
3. M. Ahsant, S. Hafeez, A. Krishna, T. Leonard, M. Surridge and S. Tsasakou, NextGRID Project Output P5.4.1/P5.4.2: Dynamic Trust Federation and Access Control v1.1, 04 July 2005.
4. Heimdal Kerberos 5 Derrick J Brashear, Ken Hornstein, Johan Ihren, et al, http://www.pdc.kth.se/heimdal/heimdal.html
5. R. Housley, RFC 3280: Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile, April 2002.
6. The KX.509 protocol William Doster, Marcus Wyde and Dan Hyde, February 2001. http://www.citi.umich.edu/projects/kerb_pki/
7. Web Services Security (WS-Security) v1.0, Chris Kaler, April 2002. http://www-106.ibm.com/developerworks/webservices/library/ws-secure/
8. Web Services Trust Language (WS-Trust) v1.1, Steve Anderson, Jeff Bohren, et al, May 2004. http://www-106.ibm.com/developerworks/library/specification/ws-trust/
9. Web Services Federation Language (WS-Federation), Chris Kaler and Anthony Nadalin, July 2003. http://www-106.ibm.com/developerworks/webservices/library/ws-fed/
10. Executive Overview of the Security Assertions Markup Language (SAML) v2.0, Paul Madsen, OASIS, Working Draft 02, November 2004. http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security

11. Debunking SAML myths and misunderstandings, Frank Cohen, IBM developerWorks, 08 July 2003. http://www-106.ibm.com/developerworks/xml/library/x-samlmyth.html?Open&ca=daw-se-news

12. WSS Message Security with SAML tokens profile Chris Kaler, et al, 1st December 2004. http://docs.oasis-open.org/wss/oasis-wss-saml-token-profile-1.0.pdf.

13. The NextGRID Architecture Straw Man. Available from NextGRID WP1.

14. Web Services Addressing (WS-Addressing) 1.0, W3C Candidate Recommendation 17 August 2005. http://www.w3.org/TR/2005/CR-ws-addr-core-20050817/

15. Web Services Policy Framework (WS-Policy), Siddharth Bajaj, Don Box, et al, September 2004. http://www-128.ibm.com/developerworks/library/specification/ws-polfram/