CITI Technical Report 01-14

A Practical Distributed Authorization System for GARA

Wiliam A. Adamson and O. Kornievskaia
andros@umich.edu, aglo@umich.edu

Abstract

Although Quality of Service functionality has become a common feature of network hardware, configuration of QoS parameters is done by hand. There is a critical need for an automated network reservation system to provide reliable *last mile* networking for video, audio, and large data transfers. Security of all communications in the process of automating the network configuration is vital. What makes this security problem difficult is the allocation of end-to-end network resources across security realms and administrative domains.

This paper introduces a practical system that shows a design and implementation of GARA services that offer automated network reservation services to users. The contributions of this paper are twofold. First, we provide a fine-grained cross-domain authorization for GARA that leverages existing institutional security and group services, with universal access for users. We identify and discuss issues involved. Second, we eliminate the need for long term PK credentials and associated overheads that are required by other systems. We describe the implementation of an easy and convenient Web interface for making reservation requests.

November 6, 2001

Center for Information Technology Integration
University of Michigan
535 West William Street, 3rd Floor
Ann Arbor, MI 48103-4943

1 Introduction

Reliable high speed end-to-end network services are increasingly important for scientific collaborators, whether separated by large distances or located just across town or campus. Our experience shows that long haul networks demonstrate good performance (thanks to overprovisioning), but the *last mile* – from the edge of the campus network to the desktop – is often a network bottleneck.

Quality of Service functionality is a common feature of network hardware. Recent studies show the viability and utility of these features to control network resources [11]. Currently, QoS configuration of network hardware is done by hand. While several standardization efforts are attempting to produce protocols that enable automated network configuration across administrative domains [12, 23], it is not yet clear which protocol(s) will be embraced.

Our work, sponsored by a multi institutional partnership,[1] addresses the need for an automated network reservation system to provide reliable *last mile* networking for video, audio, and large data transfers for the partner institutions. Reliable end-to-end network service between partner institutions is achieved by reserving network resources within the end-point institution networks, coupled with the demonstrated adequate performance of the overprovisioned interconnecting long haul networks, wherein no network resource reservation is needed.

In automating network configuration, security of all communications is vital. Network hardware is a prime target for malicious hackers, because controlling the routing and resource allocation of a network enables myriad other attacks. What makes this security problem difficult is the cross-domain nature of end-to-end network resource allocation. A user requesting end-to-end network resource allocation between the local domain and a remote domain needs to be authenticated and authorized in both domains before the request can be granted.

Our work is based on the Globus General-purpose Architecture for Reservation and Allocation (GARA) [6, 8, 7, 10]. This is a natural choice because the project partner institutions all run Globus software in either production or pre-production mode. The goal of the GARA architecture is to create a flexible solution that satisfies requirements of different types of resources (networks, CPUs, disks, etc.), while providing a convenient interface for users to create both advance and immediate reservations. GARA uses the Globus Grid Security Infrastructure (GSI) [5] for authentication and authorization. An attractive feature of GSI is that it performs cross-domain authentication, as well as coarse-grained access control.

GSI achieves cross-domain authentication by relying on a Public Key Infrastructure (PKI) and requires users to have long term PK credentials. However, many sites lack a PKI, yet they do have an installed Kerberos [17] base. The University of Michigan is one such site.

In this paper we describe the design and implementation of a GARA system that automates network reservations. The contributions of this paper are twofold. First, we provide a fine-grained cross-domain authorization for GARA that leverages existing security and group services, with universal access for users. Second, we eliminate the need for long term PK credentials, currently required by the system. We also introduce a secure and convenient Web interface for making reservation requests based on Kerberos credentials.

The remainder of this paper is organized as follows. Section 2 describes the GARA architecture. Section 3 describes the KX509 and KCT services and shows how they allow universal access to GARA by enabling a reservation to be made via the Web, obviating the need to install Globus software on workstations. Section 4 presents an architecture for distributed authorization that employs a shared namespace, delegated authorization through secure and trusted channels and a signed authorization payload, and the policy engine used to make the authorization decision. Section 5 is a step by step description of the enhanced GARA system. Section 6 identifies issues that require further research. Section 7 discusses related work. Section 8 concludes by summarizing contributions.

[1]University of Michigan Department of Physics, University of Michigan College of Literature, Science, and the Arts, University of Michigan Center for Information Technology Integration (CITI), University of Michigan Office of the Vice President for Research, Merit, University Corporation for Advanced Internet Development(UCAID), European Organization for Nuclear Research (CERN), Argonne National Laboratory (ANL)

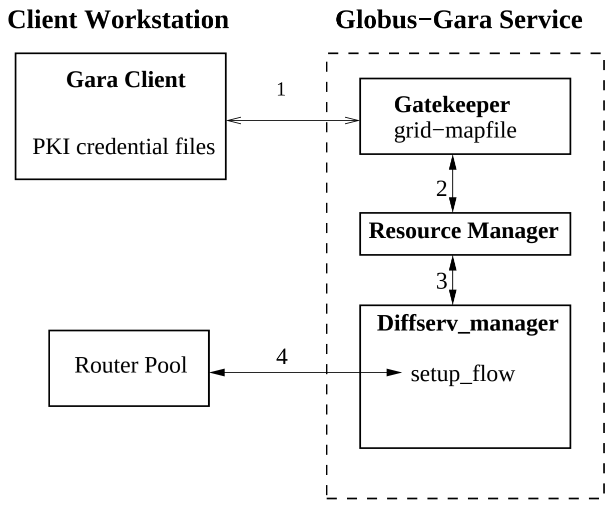

Figure 1: **GARA Architecture.** This figure shows the process of reserving local network resources with the GARA architecture. Long term user PK credentials are stored in the client workstation filesystem. As indicated by the dashed box, a Globus-Gara Service consists of a gatekeeper, a resource manager and a *diffserv_manager* all residing on a single machine. Step 1 uses the GSI protocol. Step 2 is an interprocess communication between root privileged processes. Step 3 uses the Globus Nexus API. Step 4 uses Telnet.

2 Globus GARA Architecture

GARA relies on Globus GSI for security mechanisms. GSI supports cross-domain authentication services. GSI achieves this using a PKI, specifying the use of self-signed Certificate Authority (CA) certificates to join domains. GSI requires users to have PK credentials signed by a CA whose self-signed certificate is available in the local filesystem, as well as a per user entry in a per service file called the *gridmap file* where PK credential Distinguished Names (DN) are mapped to local user names. Figure 1 shows a GARA reservation request step by step. It is assumed that the user has already acquired the long term PK certificate.

1. The user runs *grid_proxy_init*, which generate proxy credentials signed with the user's long term key. The user runs the GARA client program and inputs reservation parameters via a command line interface. A GSI [16] secured connection is established between the GARA client and the gatekeeper. The reservation request parameters are passed in the RSL (Resource Specification Language) form [24]:

```
(reservation-type=network)
(start-time=997212110) (duration=5)
(endpoint-a=141.211.92.130)
(endpoint-b=141.211.92.248)
(bandwidth=5) (protocol=tcp)
```

2. The gatekeeper finds an entry in the *gridmap* file that matches the Distinguished Name field in the proxy certificate used to authenticate the user. The gatekeeper forks a resource manager.

3. The resource manager uses the Nexus API [13, 14] for interprocess communication and passes the RSL to the instance of the *diffserv_manager*, which is running with root privileges.

4. The *diffserv_manager* checks configuration files to locate the routers servicing the reservation source and destination hosts and on the availability of requested network resources on the routers. Having determined resource availability, the *diffserv_manager* runs the *setup_flow* Expect script which Telnet's to the appropriate routers and configures the flow.

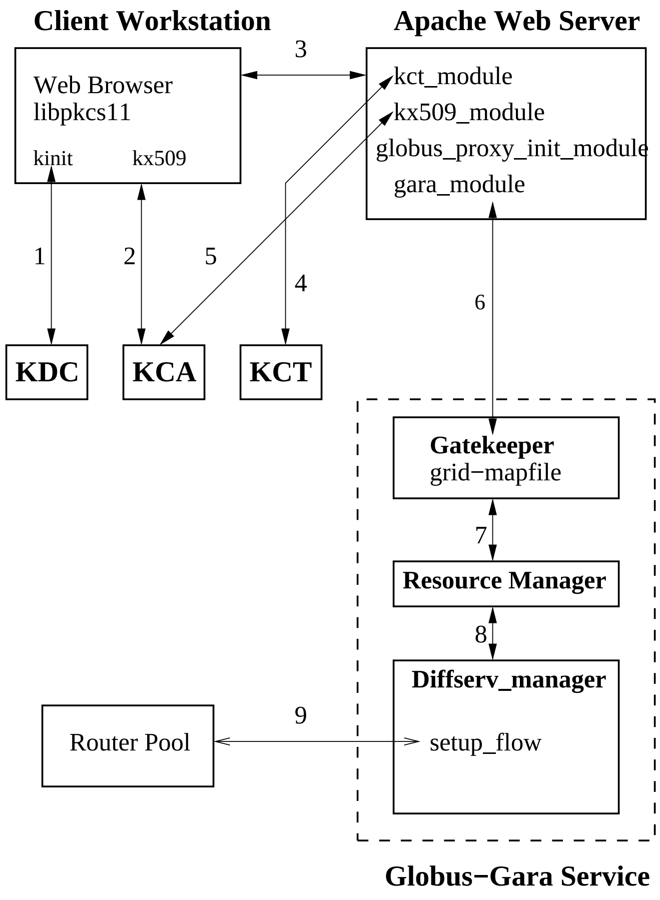

Figure 2: **KX509 GARA Web Interface.** This figure how local network resources are reserved with the GARA Web interface. KX509 junk keys replace long term PK credentials. As indicated by the dashed box, a Globus-Gara Service consisting of a gatekeeper, a resource manager and a *diffserv_manager* all residing on a single machine. Steps 1,2,4,5 use the Kerberos protocol. Step 3 uses HTPPS. Step 6 uses the GSI protocol. Step 7 uses interprocess communication between root privilege processes. Step 8 uses the Globus Nexux API. Step 9 uses Telnet.

3 Web Interfaces to GARA

Many sites, such as the University of Michigan, lack a PKI, but they do have an installed Kerberos base. While Globus provides software to translate from Globus PK credentials into Kerberos credentials or Andrew File System (AFS) tokens at the service, PK user credentials are still required at the client.

The University of Michigan has developed a service that allows users to access Grid resources based on their Kerberos credentials. The KX509 [9, 15] system translates Kerberos V4 or V5 credentials into short-lived PK credentials, or *junk keys*, which in turn can be used by browsers for mutual SSL authentication or by GSI for Globus authentication. KX509 creates a new public/private keypair, and sends the public key to a Kerberized Certificate Authority (KCA) over a Kerberos secured channel. Successful communication with the KCA implies valid Kerberos credentials. The KCA creates and signs an X.509 certificate using the presented public key. The KCA sets a short credential lifetime and returns the junk keys.

Junk keys have several advantages over traditional long-lived PK credentials. Junk keys have a short lifetime, so the problems associated with the need to check the validity of a user's long term public PK certificate (e.g. revocation) are largely obviated. Furthermore, KX509 supports mobile users who obtain new junk keys at each workstation while users with long term PK keys need to somehow access their specific long term private key at each workstation.

In order to make network resource reservations convenient for users, we built a GARA Web interface. GARA client code, which runs GSI, resides on the Web server and uses Globus proxy credentials created on behalf of the user accessing the GARA network reservation form. The Web server runs KX509 on the user's behalf, which creates new junk keys for the user on the Web server. These junk keys are then used to create Globus proxy credentials. Figure 2 details these tasks.

1. User executes *kinit* to acquire Kerberos credentials. This usually occurs at login.
2. User executes *kx509* and acquires junk keys. (This can be placed in a PAM [21] login module).
3. Using a browser, the user makes an HTTPS request for the network resource reservation page. Mutual SSL authentication is required by the Web server, so the junk keys obtained in step 2 are used. The reservation request parameters are sent to the Web server in the RSL form.
4. The Web server *kct_module* makes a Kerberos authenticated request to the Kerberized Credential Translator (KCT) [15] to acquire a service ticket for the KCA service on the user's behalf.
5. The Web server *kx509_module* acquires junk keys on behalf of the user, as in Step 2. This set of user junk keys are stored on the Web server. The Web server *globus_proxy_init module* then uses the newly created junk keys to create Globus proxy credentials.
6. The Web server *gara_module* uses the proxy credentials and sends the reservation request to the gatekeeper using the Globus GSI protocol. (This step is the same as Step 2 described in Section 2 protocol).
7. The gatekeeper finds an entry in the *gridmap* file that matches the Distinguished Name field in the proxy certificate used to authenticate the user. The gatekeeper forks a resource manager. (This step is the same as Step 2 described in Section 2 protocol).
8. The resource manager uses the Nexus API for interprocess communication and passes the RSL the instance of the *diffserv_manager*, which is running with root privileges. (This step is the same as Step 3 described in Section 2 protocol).
9. The *diffserv_manager* checks configuration files to locate the routers servicing the reservation source and destination hosts and on the availability of requested network resources on the routers. Having determined resource availability, the *diffserv_manager* runs the *setup_flow* Expect script which Telnet's to the appropriate routers and configures the flow. (This step is the same as Step 4 described in Section 2 protocol).

4 Distributed Authorization Design

A cross domain distributed authorization scheme is one that allows authorization decisions to be made when the requestor and resources reside in separate domains. Often authorization decisions are made by a policy engine that applies a set of input attributes to a set of policies. These attributes might include user attributes such as group membership or environmental attributes such as time of day. In designing the distributed authorization system for GARA, we must address the following issues:

- *Use of existing services*: Partner institutions have group and security services in production. These services maintain data about users and resources. Using these existing services avoids the additional administrative and computational overhead of creating and managing new services for the same users and resources. A centralized service where all domains add user and resource information has the additional problem of not scaling to the combined number of users and resources currently maintained locally at partner institutions.
- *Shared namespace*: Central to any authorization service design is the formation of an attribute namespace that is understood by policy engines.
- *Signed authorization payload*: An authorization payload contains attributes gathered locally and presented to a remote policy engine. A cryptographic signature of the payload verified by the remote policy engine provides authenticity.
- *Authorization architecture*: Attribute information can come from a variety of sources — from a local service, from the local environment, attached to the resource request, etc. We therefore separate the authorization process into two phases, the *gathering of attributes*, and the *running of the policy engine*. We consider several places in GARA's resource request data flow to perform these tasks.

4.1 Shared Namespace

Frequently, the primary concern in the authorization decision is related to a group membership question: does this user belong to appropriate groups? Within a domain, the statement of group membership is well defined. Both user identity information and a group namespace are available.

A distributed system spanning different administrative domains also requires prior agreement on a shared attribute namespace. A critical part of the distributed authorization service design is architecting the shared namespace.

One solution is the creation of a new central group service where user and resource information from different domains are replicated and managed by domain administrators [26]. The central service maintains a group namespace that is referenced in policy information passed to authorization policy engines. This scheme has the drawback of adding the administrative overhead of maintaining replicated data, and does not scale to hundred of thousands of potential users.

Another solution is to provide a callback address to a local domain group service along with a user name [1]. The callback is used by a remote resource policy engine to gather authorization information from the user's local service. In this scheme, the shared namespace needs to be centrally defined but managed locally by adding the appropriate names to the local service. This scheme avoids the pitfall of data replication at the expense of additional communications.

Our architecture tries to avoid both data replication and, in the common case, extra communication outside the local domain. As in the above example, we propose the formation of a shared namespace (SN_groups), a small number of group names that is shared across domains to be used to determine access to network resources. Each domain can create groups with these names in their existing group service and manage user membership as any local group. Instead of using a callback, we place authorization information in the resource request to the remote network. The GARA in the domain local to the requestor queries the local group membership service and sends the subset of SN_groups in which the requestor is a member to the remote domain GARA, which uses it as input to the remote resource policy engine. This case is illustrated in Figure 6.

4.2 Authorization Architecture

We examine the resource request data flow and note several places where the authorization decision could be implemented. Each option implies certain architecture decisions.

4.2.1 Web Server

The first (proxy) service with which the requestor comes in contact while making a request appears to be a reasonable place to check whether the requestor is authorized to perform the desired action. Prior to initiating any contact with the desired resource, an authorization service could be contacted with the user's identity and resource request information. This authorization service would necessarily need to have a policy for each resource and information on each user. While this architecture would provide a central place for policy, and could exist per local domain and so use existing local group services, it presents extra communications when the resource is not available, or when fine-grained authorization is not required.

4.2.2 Globus Gatekeeper

A limited form of authorization is currently provided by the gatekeeper and enforced with a *gridmap* file. This file contains a mapping between the DN and local username for an authorized users. If no entry is found for the requestor, then the request is denied. A call to check the *gridmap* file can be replaced with a more general authorization check. This option has several disadvantages.

Adding authorization at the gatekeeper has an impact on the gatekeeper's performance. The gatekeeper is expected to handle a lot of requests, thus it pushes the functionality of serving individual requests down to the particular resource manager.

At the gatekeeper, it is still unknown if the resource is available, so as above, the extra communication and work to make an authorization decision could be wasted effort. Furthermore, there is no advantage to having the gatekeeper manage per resource policy information, where one resource may need fine grained access control, and another may need no access control.

4.2.3 Diffserv Manager

The last step before the reservation for the resource is accepted and performed is also the last point where the authorization decision can be performed. This choice enforces authorization at the *diffserv_manager*, and provides a flexible and scalable means for resource services to achieve security. Each service is capable of stating, enforcing, and modifying its policies without depending on the administration of the Globus architecture at large. We perform the authorization decision in the *diffserv_manager*.

4.3 Signed Authorization Payload

The authorization payload is added to the reservation request by the first *diffserv_manager* in the requestor's local domain, after local resources have been successfully reserved (see section 5). The payload is unpacked by successive *diffserv_managers* and is used as input to the policy engine.

As part ot the rest of the reservation request, the payload is protected by the GSI session key based on the Globus proxy credential of the user. The user is allowed to add data to this channel (e.g. the reservation request parameters) and could therefore add false data to the authorization payload. To secure the authorization payload, we require the *diffserv_manager* to sign the authorization payload before adding it to the reservation request. There are several possible ways of signing the payload:

- A *diffserv_manager* can sign the payload with its private key and include the corresponding public key in a certificate in the reply to the Web server. We assume this certificate was issued by the same CA that issued a certificate to the Globus gatekeeper. The signed payload and the certification are included in the request to the remote domain. We sign the authorization payload in this manner.

- A *diffserv_manager* can sign the payload with the gatekeeper's private key. Because the gatekeeper's certificate is known to the remote party, no additional certificates are transmitted and no additional trust relations are assumed. However, if the *diffserv_manager* requires access to the gatekeepers certificate, it must either reside on the same hardware as the gatekeeper and run with gatekeeper's privileges, or it must satisfy all the security requirements imposed on the gatekeeper (e.g., secure storage of the private key).

- A resource manager can sign the payload after receiving the reply from the *diffserv_manager*. Although the resource manager is guaranteed to run on the same hardware as the gatekeeper, it runs with different privileges than the gatekeeper and thus does not have easy access to the private key.

4.4 Policy Engine

After all the requestor's attributes including group membership have been established, the authorization decision can be made by the policy engine. In order to allow for the use of different policy engines, the authorization callout has a generic API that passes information about the requestor and the requested action to the policy engine. We chose KeyNote [2, 3, 4] for our policy engine because of its easy availability.

Applications such as GARA describe policies to KeyNote with a set of attribute-value pairs (called an action condition) creating a *policy namespace*. In Figure 5, the policy namespace consists of the following attributes and their corresponding values: app_domain, operation, type, location, amount, time, and grid_bw. In Section 4.1 we discussed the notion of a shared namespace. An example in the KeyNote policy of SN_groups would be the group named grid_bw. Each Globus site is free to choose any descriptive attribute name to express the security policy. However, if any of the attributes are to be included in the authorization data that is passed to or received from a remote domain, then the attributes need to be included in the shared namespace we described in Section 4.1.

We now describe the basic pseudocode for the KeyNote policy engine call with a group membership action condition.

- *requester*: the requesting principal's identifier.
- *action_description*: the data structure describing an action contains attribute value pairs which are included in the request. For example, "$system\_load \leq 70$" specifies an environment condition stating that the current system load must not exceed 70%.
- *SN_groups*: the shared namespace groups in the action_description, also added as attribute value pairs describing the action. For example, "umich_staff = yes" states the request is a member of the *umich_staff* group.
- *policy*: the data structure describing local policy, typically read from a local file.
- *credentials*: the data structure with any relevant credentials, typically sent along with the request by the requesting principal. Before making use of these credentials, their validity must be confirmed by verifying a signature included in the credential data structure.

Figure 3 shows the main actions of *diffserv_manager*.

```
SN_groups =
  retrieve_group_membership(requestor);

result =
  authorization_decision(requestor,
  action_description, policy, credentials);

if(result == "allowed")
  do the requested action
else
  report action is not allowed
```

Figure 3: Pseudo-code for the authorization mechanism in *diffserv_manager*
.

Figure 4 provides details of the *authorization_decision* function.

Figure 5 shows an example of a KeyNote top level security policy that allows the action if the following conditions hold: an application domain is called *gara* and the requested operation is *reservation* for the resource of type *bandwidth*. Furthermore, if this is a local request then bandwidth for more than 100Mb is not allowed. If the request is from a remote user, then amount greater than 10Mb is not allowed. If the current time is after hours, then no restriction on bandwidth is enforced. The requestor must be a member of *grid_bw* group.

```
session_id = kn_init();

kn_add_assertion(session_id,policy[i]);

kn_add_assertion(session_id,credentials[i]);

for all actions in action_description
  kn_add_action(session_id,
    action_description.attr,
    action_description.value);

result = kn_do_query(session_id);
```

Figure 4: Pseudo-code for the call to the KeyNote Policy Engine

```
keynote-version: 2
local-constants: ADMIN_UM = "x509-base64:MIICrzCC"
authorizer: "POLICY"
licensees: ADMIN_UM
conditions: app_domain == "gara" &&
  operation == "reservation" &&
  type == "bandwidth" &&
  ((location == "local" && @amount <= 100) ||
  (location == "remote" && @amount <= 10) ||
  time == "night") && grid_bw == "yes");
```

Figure 5: Trusted assertion stating KeyNote top level security policy. Note that the value of the key has been truncated.

If the KeyNote policy engine states that the action is not allowed, no reservation is made by the local *diffserv_manager* and an authorization failure is returned to the Web server. As the result, *e2e_gara_module* stops the reservation protocol and replies back to the client with the authorization error. *e2e_gara_module* returns a success value to the client only if both local and remote authorization has succeeded.

5 Putting It All Together

We successfully demonstrated our modifications to GARA by reserving bandwidth for a video application running between the University of Michigan and CERN.

Bandwidth was reserved by filling in a form served by a modified Apache Web server that runs the GARA client. The GARA client communicates with separate GARA services at each endpoint domain, as shown in Figure 6. The GARA services use KeyNote authorization policies configured to require bounded request parameters for bandwidth, time and duration. Group membership is also required.

We demonstrated that if any of the policy parameters were not satisfied, e.g. too much requested bandwidth or incorrect AFS PTS group membership, the reservation fails. If the request parameters are in bounds, and if the user is a member of the correct AFS PTS group(s), the reservation succeeds.

Successful reservation results in configuring the end domain Cisco ingress routers with the appropriate Committed Access Rate (CAR) rate_limit, which marks the packets and polices the flow. The participating routers are statically configure with WRED, Cisco's implementation of the Random Early Detection (RED) class of congestion avoidance algorithms.

What follows is a step by step description of an end-to-end network reservation using the enhanced GARA.

1. The user in the local realm executes *kinit* to acquire Kerberos credentials. This usually occurs at login.
2. The user in the local realm executes *kx509* and acquires junk keys. This can be placed in a

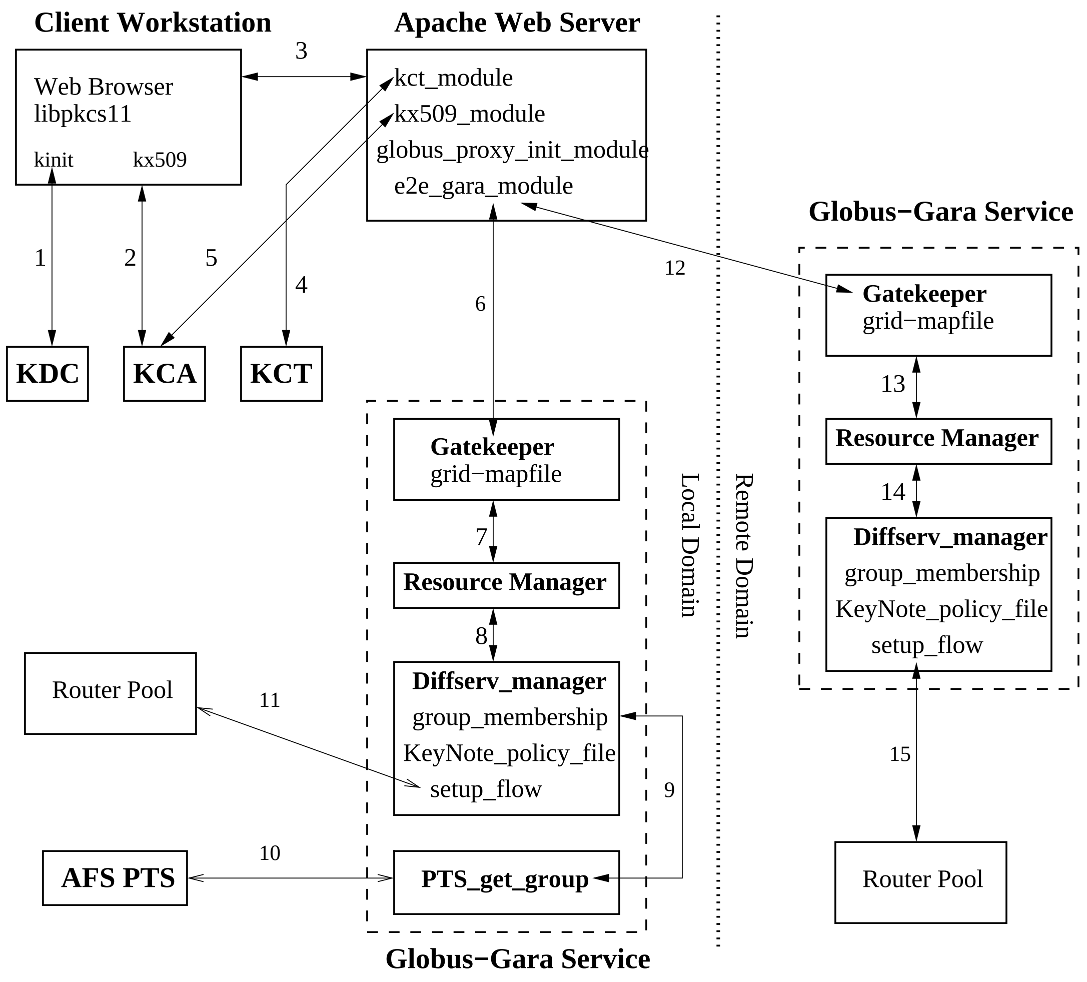

Figure 6: **Network Resource Reservation Data Flow.** KDC is a Kerberos Key Distribution Center. KCA is a Kerberized Key Signer. KCT is a Kerberized Credential Translator. KDC and KCT must share hardware because both require access to the Kerberos database. As indicated by the dashed box, a Globus-Gara Service consisting of a gatekeeper, a resource manager, a *diffserv_manager*, and a group module all running on a single machine. Steps 1,2,4 and 5 use mutually authenticated Kerberos protocol. Step 3 uses mutually authenticated SSL protocol. Steps 6 and 12 use the GSI protocol. Steps 7,9 and 13 are an interprocess communication between root privilege processes. Step 10 uses the RX, the AFS remote procedure call protocol. Steps 11 and 15 use Telnet.

PAM login module [21].

3. Using a browser, the user makes an HTTPS request for the network resource reservation page. Mutual SSL authentication required by the Web server uses the junk keys obtained in Step 2. Network reservation parameters such as source and destination IP address, desired bandwidth, start time, etc. are filled in and sent to the Web server.

4. The Web server *kct_module* makes a Kerberos authenticated request to the Kerberized Credential Translator or KCT [15] to acquire a service ticket for the KCA service on the user's behalf.

5. The Web server *kx509_module* acquires junk keys on behalf of the user as in Step 2. These junk keys are stored on the Web server. The Web server *globus_proxy_init module* then uses the newly created junk keys to create user Globus proxy credentials.

6. The Web server *e2e_gara_module* uses the proxy credentials and sends the reservation request to the local gatekeeper using the Globus GSI protocol.

7. The local GARA gatekeeper finds an entry in the *gridmap* file that matches the Distinguished Name field in the proxy certificate use to authenticate the user. The DN and local id are attached to the RSL (Resource Specification Language):

```
(reservation-type=network)
(start-time=997212110) (duration=5)
(endpoint-a=141.211.92.130)
(endpoint-b=141.211.92.248)
(bandwidth=5) (protocol=tcp)
(client-name=aglo) (client-dn=287f42
c19122ec08ddc679ba259070f9)
```

   The last two attribute value pairs (client-name=client-dn) are the two fields not present in the original GARA RSL reservation request. DN is a considerably long string, se we use a hashed value of the DN instead. The local gatekeeper forks a resource manager.

8. The resource manager uses the Nexus API for interprocess communication and passes all the information on to the instance of the *diffserv_manager*, which is running with root privileges.

9. The *diffserv_manager* checks configuration files to locate the routers servicing the reservation source and destination hosts and on the availability of requested network resources on the routers. Having determined resource availability, The *diffserv_manager* passes the local user name to the group_membership function, which performs one of several actions, depending on configuration and on whether a request is from a local or remote user. As the authorization data in this RSL is null, the request is from a user in the local domain. The group_membership function uses interprocess communication with root privilege to communicate with a get_group module. The UMICH get_group module, PTS_get_group, queries the Andrew File System (AFS) PTS group service. The modular design allows a query to any local group service such as an LDAP service, or a flat file.

10. The PTS_get_group module receives the local name as input and queries the AFS PTS for the list of all SN_groups that include local name. The call is performed over an authenticated channel using *diffserv_manager* identity. Prior to accepting any connections, *diffserv_manager* acquires AFS tokens needed to authenticate with the PTS server. The *diffserv_manager* passes the results of the get_module query, the reservation request information, and environment information to the KeyNote policy engine which reads the policys from the KeyNote policy file and makes and authorization decision.

11. Having determined resource availability and that the user is authorized to make the request, the *diffserv_manager* runs the *setup_flow* Expect script which Telnet's to the appropriate routers and configures the flow. The *diffserv_manager* packages the results of the group_membership function call into a string which is signed and added to the RSL.

12. This step is the same as Step 6 except this time the RSL carries the authorization payload. We use *auth-data* as the attribute name. The value is variable length. In our current implementation, the RSL is limited to 4KB, at least enough to encode information 64 groups (assuming 64 byte names).

13. Same as Step 7.

14. Same as Step 8.

15. This is the same as Step 9, except that since the user is from a remote realm, the auth_data in the RSL is non-null and is used as input to the KeyNote policy engine instead of the results of a group_membership function call.

6 Discussion

Our design enables expression of many policies, including who can request which network resources and when such requests are valid. In the example we presented, the authorization payload is signed by one certificate, the remote GARA *diffserv manager*. More broadly, a site must require the authorization payload to contain assertions from other services. For example, a site might require that users be U.S. citizens, verified by some specific Certificate Authority. Signed assertions can be added to the authorization payload to accommodate such requirements.

We have yet to explore the management of the SN_group namespace and associated policies. We envision a service [2] that lists SN_group names. Each site might also post the authorization information needed to successfully access a resource.

Our desire to provide fully secure communication channels fell short at the router configuration communication. Telnet places passwords on the wire in the clear, so the router configuration step relies on physical security in the form of a protected network between the *diffserv_manager* and the routers. Another security exposure in the design is the setup_flow Expect script, which stores in the local file system the password(s) used to log into the routers command line interface.

Our choice of policy engines was influenced by the availability of working code. The modular design allows for use of other policy engines. Akenti [25], and GAA API [20] were also considered. We acknowledge Akenti's strength over KeyNote in terms of credential management. On the other hand, Akenti imposes a lot of overhead, not required by KeyNote, such as creation of certificates for each of the clients.

[2] Web and/or LDAP

7 Related Work

The Globus MyProxy [19] initiative provides a trusted server to store a user's delegated credentials indexed by a tag and a password. Later, a service can contact a MyProxy server, present a tag and a password and receive corresponding credentials (e.g., certificate or Kerberos ticket) on a client's behalf. Each service requires a different tag and password, forcing users to manage many passwords. This approach requires users to type in their passwords into HTML forms. HTML forms are easily reproduced by a malicious hacker wanting to collect passwords who can obtain a certificate signed by one of the default Certificate Authorities supported by browsers, and run a web server that uses the credential to deliver the spoofed login HTML form. The user can tell if the login form is a spoof by examining the credential - an activity most users don't bother with.

The Grid Portal Architecture [18] is a Web interface to Grid Computing resources that uses MyProxy Services for client authentication.

The Community Access Service (CAS) [26] is a proposed Grid authorization service that the user calls prior to making a request for Grid resources. CAS returns a signed capability to indicate a successful authorization request. The capability is then added to the Grid resource request.

The GARA client is designed to contact each end domain GARA service. Future GARA implementations will have the GARA client contact the first GARA service, which in turn will contact other bandwidth brokers (BB) needed for the end-to-end reservation. The need for such a bandwidth broker to bandwidth broker protocol is discussed by Sander et al. in [22]. The Simple Inter-Domain Bandwidth Broker Specification (SIBBS) [23] is a simple request-response bandwidth broker to bandwidth broker broker protocol being developed by the Internet2 QBone Signaling Design Team. It is anticipated that GARA will be an early development code base for SIBBS.

8 Conclusions

We have demonstrated a scalable fine-grained distributed authorization service for GARA that joins local domain group services via a shared namespace, and asserts group membership by adding a signed authorization payload to existing communications. We eliminate the need for long term PK credentials currently required by the system. We also introduce a secure and convenient Web interface for making reservation requests based on Kerberos credentials.

We showed that authorization succeeds only when the user is a member of the correct groups and the reservation parameters are within bounds, as dictated by the policies present at each endpoint *diffserv_manager*.

9 Acknowledgments

We thank Homer Neal, Sean Mckee, Eric Myers, Brian Athey, Tom Hacker, Victor Wong, Roy Hockett, Bill Doster, and Kevin Coffman from the University of Michigan, as well as Olivier Martin, Paolo Maroni, and Danny Davids from CERN for their help in designing and demonstrating this work. We also acknowledge Ian Foster and Alain Roy from Argonne National Lab and Volker Sander from Forschungszentrum Jlisch for their work on the GARA code base and their support of this work. We thank Peter Honeyman for his criticism and advice.

References

[1] A Digital Library Authentication and Authorization Architecture. `http://www.ucop.edu/irc/cdl/tasw/Authentication/Architecture-3_W95.pdf`.

[2] M. Blaze, J. Feigenbaum, J. Ioannidis, and A. Keromytis. The KeyNote trust managment system version 2, September 1999. RFC 2704.

[3] M. Blaze, J. Feigenbaum, and A. Keromytis. Keynote: Trust management for public-key infrastructure. In *Proceedings Cambridge 1998 Security Protocols International Workshop*, 1998.

[4] M. Blaze, J. Feigenbaum, and M. Strauss. Compliance checking in the PolicyMaker trust management system. In *Proceedings of Second International Conference on Financial Cryptography*, 1998.

[5] R. Butler, D. Engert, I. Foster, C. Kesselman, S. Tuecke, and J. Volmer. A national-scale authentication intrastructure. *IEEE computer*, 33(12):60–66, 2000.

[6] Documentation: A Guide to GARA. `http://www-fp.mcs.anl.gov/qos/qos_papers.htm`.

[7] Documentation: Administrators Guide to GARA. `http://www-fp.mcs.anl.gov/qos/qos_papers.htm`.

[8] Documentation: Programmers Guide to GARA. `http://www-fp.mcs.anl.gov/qos/qos_papers.htm`.

[9] W. Doster, M. Watts, and D. Hyde. The KX.509 protocol. CITI Technical Report 01-2, February 2001.

[10] I. Foster, A. Roy, and V. Sander. A Quality of Service Architecture that Combines Resource Reservation and Application Adaptation. In *Proceeding of the 8th International Workship on Quality of Service (IWQQOS 2000)*, pages 181–188, June 2000.

[11] GEANT. Premium ip overview. `http://www.inernet2.edu/presentations/vimm/20011004-QoSWorkingGroup-Campanella.htm`.

[12] IETF Internet Traffic Engineering Working Group. `http://www.ietf.org/html.charters/tewg-charter.html`.

[13] N. Karonis, C. Kesselman, G. Koenig, and S. Tueke. A secure communication infrastructure for high-performance distributed computing. In *Proceedings of the 6th IEEE Symposium on High-Performance Distributed Computing*, pages 125–136, 1997.

[14] Kesselman and S. Tueke. Managing security in high-performance distributed environment. *Cluster Computing*, pages 95–107, 1998.

[15] O. Kornievskaia, P. Honeyman, B. Doster, and K. Coffman. Kerberized credential translation: A solution to web access control. In *Proceedings of the 10th USENIX Security Symposium*, pages 235–249, August 2001.

[16] J. Linn. Generic security service application program interface, version 2, update 1, October 2000. RFS2743.

[17] C. Neuman and T. Ts'o. Kerberos: an authentication service for computer networks. *IEEE Communications*, 32(9):33–38, September 1994.

[18] Grid Computing Portal. `http://hotpage.npaci.edu/cgi-bin/hotpage_top.cgi`.

[19] MyProxy project. `http://dast.nlanr.net/Projects/MyProxy`.

[20] T. Ryutov and C. Neuman. Representation and evaluation of security policies for distributed system services. In *Proceedings of the DISCEX*, January 2000.

[21] V. Samar and R. Schemers. Unified login with pluggable authentication modules (PAM), October 1995. OSF Request For Comments 86.0.

[22] V. Sander, W. A. Adamson, I. Foster, and A. Roy. End-to-end provision of policy information for network qos. In *Proceedings of the 10th Symposium on High Performance Distributed Computing*. IEEE, August 2001.

[23] SIBBS. The simple inter-domain bandwidth broker specification. `http://qbone.internet2.edu/bb/`.

[24] The Globus Resource Specification Language RSL v1.0. `http://www.globus.org/gram/rsl_spec1.html`.

[25] M. Thompson, W. Johnson, S. Mudumbai, G. Hoo, K. Jackson, and A. Essiari. Certificate based access control for widely distributed resources. In *Proceedings of the 8th USENIX Security Symposium*, August 1999.

[26] S. Tuecke. Security and CAS futures. `http://www.globus.org/about/events/retreat01/presentations/retreat01tueckeSecurity_and_CAS_Aug01.ppt`.