A Practical Distributed Authorization System for GARA

William A. Adamson and Olga Kornievskaia

Center for Information Technology Integration
University of Michigan, Ann Arbor, USA
{andros,aglo}@citi.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 Globus General-purpose Architecture for Reservation and Allocation (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 public key 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.

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 over provisioning), 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. 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, 19], it is not yet clear which protocol(s) will be embraced.

Our work 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 over provisioned 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. Requesting end-to-end network resource allocation between the local domain and a remote domain, a user 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 by relying on a Public Key Infrastructure (PKI) and requiring users to have long term public key (PK) credentials.

GSI provides coarse-grained access control. A flat file, called the *gridmap* file, stores mappings from PK credentials (Distinguished Names, (DN)) to local user names. A user is allowed access to Globus services if there is an entry corresponding to this user in the *gridmap* file. This all-or-nothing access control policy is extremely limiting. Authorization decisions in QoS are based on many parameters such as the amount of available bandwidth, time of day, system load, and others. We propose to control resource usage with a policy engine and expressive security policies.

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 Kerberized Credential Authority (KCA) and Kerberized Credential Translation (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 3 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

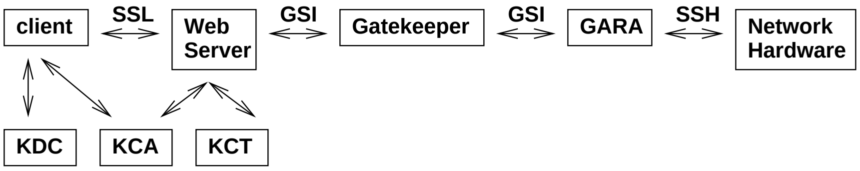

Fig. 1. KX509 GARA Web Interface. This figure shows how local network resources are reserved with the GARA Web interface. KX509 junk keys replace long term PK credentials. Communications with the KDC, KCA, and KCT are Kerberos protected.

to make the authorization decision. Section 4 briefly describes implementation of the system. Related work is presented in Section 5. Finally, Section 6 concludes.

2 GARA Web Interface

Many sites, such as the University of Michigan, lack a PKI, but they do have an installed Kerberos [14] base. The University of Michigan has developed a service that allows users to access Grid resources based on their Kerberos credentials. The KX509 [9, 13] system translates Kerberos 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.

Junk keys have several advantages over traditional long-lived PK credentials. They have short lifetimes, so the revocation problem [1] is largely obviated. While, in a traditional PKI, long term credentials put the ease of user mobility in question, KX509 users can obtain new junk keys at each workstation.

KX.509 creates a new public/private keypair and sends the public key to a Kerberized Certificate Authority (KCA) over a Kerberos secured channel. Using the presented public key, the KCA creates and signs a short term X.509 identity certificate.

In order to make network resource reservations convenient for users, we built a GARA Web Interface. A user makes a reservation by filling out a GARA network reservation Web form. All requests are SSL protected and require mutual authentication. As opposed to a traditional password-based user authentication, we use short-lived user certificates, priorly acquired with KX.509. After the Web server authenticates the user, it contacts a Kerberized Credential Translation (KCT) [13] server, presents appropriate credentials, and requests Kerberos credentials on the user's behalf. Next, the Web server runs KX509 on the user's behalf, which creates a new junk key for the user on the Web server. This junk key is then used to create Globus proxy credentials. GARA client code resides on the Web server and uses Globus proxy credentials. Figure 1 gives an overview of the GARA Web Interface.

3 Distributed Authorization Design

In a cross domain distributed authorization scheme, authorization decisions are made even if the requestor and resources reside in separate domains. Often authorization decisions are made by a policy engine that applies policy rules to a set of input attributes. These attributes might include user attributes such as group membership or environmental attributes such as time of day. Attribute information can come from a variety of sources: local services, environment, configurations, or attached to the resource request. We separate the authorization process into two phases: gathering of attributes and running of the policy engine.

In designing the distributed authorization system, we must address the location where the authorization decision takes place. We discuss how the use of shared namespace and delegated credentials are the key to creating a practical authorization scheme. We also believe in utilizing existing local authorization services to require as little replication of information as possible.

Location of authorization decision: The question that needs to be answered is: where is the best place in GARA to make the authorization decision? Three possible locations exist: Web server, gatekeeper, and resource manager.

Prior to initiating any contact with the desired resource, the Web server can contact an authorization service and provide user's identity and resource request information. Having such an authorization service would perforce need to have a policy for each resource and information about each user. However, this choice presents extra communications when the resource is not available, or when fine-grained authorization is not required.

The gatekeeper is expected to handle numerous requests, so performing the authorization decision at the gatekeeper could have an impact on the gatekeeper's performance. 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. We conclude that adding authorization at the gatekeeper would be counter productive.

The best place to enforce authorization in the GARA architecture is at the resource manager where each service is capable of stating, enforcing, and modifying its policies without depending on the administration of the Globus architecture at large.

Shared namespace. Central to any authorization service design is the formation of an attribute namespace that is understood by policy engines. Frequently, the primary concern in the authorization decision is related to a group membership question: does this user belong to appropriate groups? Consequently, a security policy would enforce the restricted membership for specific actions. Within a domain, the statement of group membership is well defined. Both user identity information and a group namespace are available locally.

A shared group namespace, presented to policy engines in multiple domains and used to control access to resources in multiple domains, is defined by a number of groups with common names across domains. In its existing group service, each domain creates groups with these names and manages user membership as any local group. Other attributes presented to the distributed policy engines such as the amount of requested bandwidth or start-time of the request are already encapsulated in a shared namespace in that they are coded as name,value pairs in the request.

Signed authorization payload. At the remote service, we do not add a callback to the local group service to determine group membership, instead, authorization information is added to the existing resource request.

The local GARA resource manager queries the local group membership service for the subset of shared namespace groups in which the requestor is a member, and passes the group list along with the request parameters to its policy engine to make an authorization decision. If the request succeeds, the local GARA resource manager creates an *authorization payload* consisting of the requestor's distinguished name and the group list. To secure the authorization payload, we require the local GARA resource manager to sign the authorization payload before adding it to the reservation reply returned to the GARA client running on the Web server. The reservation request is forwarded to the remote GARA who validates the signature on the received authorization information before passing the group list as input to its policy engine. Thus we piggy-back the group membership information on the existing reservation request communication.

Policy Engine. After all the requestor's attributes such as group membership and request parameters have been established, the fine-grained authorization decision can be made. In general, policy engines accept attribute-value pairs as input, compare the input attributes to a set of policy rules, and return a pass/fail response. The granularity of the authorization decision is embodied in the complexity of the policy rules that must be satisfied by the input attribute-value pairs. To allow different policy engines, the authorization callout has a generic API that passes information about the requestor and the action to the policy engine. We chose KeyNote [2–4] for our policy engine because of its flexibility and easy availability.

4 Implementation

4.1 KeyNote Policy Engine

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 4, the policy namespace consists of the following attributes and their corresponding values: app_domain, operation, type, location, amount, time, and grid_bw. To express a security policy, each Globus site is free to choose any descriptive attribute name. 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 previously.

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

- *requester*: the requesting principal's identifier.

```
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
```

Fig. 2. Pseudo-code for the authorization mechanism in *diffserv_manager*.

```
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);
```

Fig. 3. 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");
```

Fig. 4. Trusted assertion stating KeyNote top level security policy. Note that the value of the key has been truncated.

- *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, "grid_bw = yes" states the request is a member of the *grid_bw* 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 2 shows the main actions of *diffserv_manager*.

Figure 3 provides details of the *authorization_decision* function.

Figure 4 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.

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, the reservation protocol returns an authorization error back to the client. A success value is returned to the client only if both local and remote authorization have succeeded.

4.2 Reservation Flow

We successfully demonstrated our modifications to GARA by reserving bandwidth for a video application running between the University of Michigan and CERN[1]. Bandwidth is reserved by filling in a Web form served by a modified Apache Web server that runs the GARA client. The GARA client communicates with separate GARA services at each end-point domain, as shown in Figure 5. 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 are not satisfied, e.g. too much requested bandwidth or incorrect AFS PTS group membership, the reservation fails.

A successful reservation results in configuration of the end domain Cisco ingress routers, that marks the packets and polices the flow, with the appropriate Committed Access Rate (CAR) rate_limit. The participating routers are statically configured 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, also illustrated in Figure 5.

1. User (locally) executes *kinit* and acquires Kerberos credentials.
2. User (locally) executes *kx509* and acquires junk keys.
3. Using a browser, a user makes an https request for the network resource reservation page. The junk key, obtained in Step 2, is used for mutual SSL authentication. Network reservation parameters such as source and destination IP address, desired bandwidth, start time are entered into a form and sent to the Web server.
4. The Web server *kct_module* makes a Kerberos authenticated request to the KCT and acquires a service ticket for the KCA service on the user's behalf.

[1] European Organization for Nuclear Research

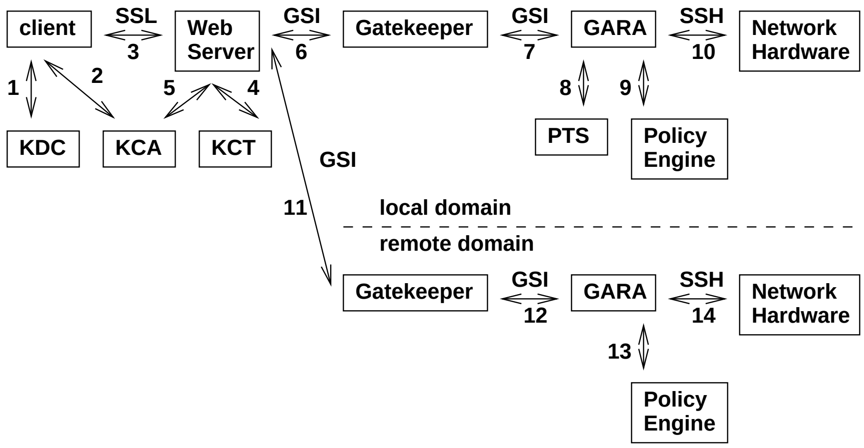

Fig. 5. Network Resource Reservation Data Flow. KDC is a Kerberos Key Distribution Center. KCA is a Kerberized Key Signer. KCT is a Kerberized Credential Translator. KDC and KCT must share hardware because both require access to the Kerberos database. PTS is AFS Protection Server.

5. The Web server *kx509_module* acquires and caches junk keys on behalf of the user as in Step 2. Then, the Web server *globus_proxy_init module* uses the newly created keys to create user's Globus proxy certificate.
6. The Web server *gara_module* constructs a reservation request to the local gatekeeper using the Globus GSSAPI_SSLEAY protocol and the proxy certificate. The local GARA gatekeeper looks for an entry in the *gridmap* file that matches the corresponding distinguished name – a field in the Globus proxy certificate (from Step 5). The DN and local id are attached to the RSL (Resource Specification Language) string.
7. Using the Nexus API for interprocess communication, the local gatekeeper forwards the RSL to the resource manager (*diffserv_manager*).
8. The local id is passed to the group_membership function, which performs one of several actions, depending on configuration and on whether the request is from a local or remote user. If the authorization data in this RSL is null, the request is from a user in the local realm. In our settings, group membership function queries a Protection Server (PTS) – a part of AFS. The call can be easily replaced by a query to any local group service such as an LDAP service or a flat file. The call is performed over an authenticated channel using the *diffserv_manager*'s identity. Prior to accepting any connections, *diffserv_manager* acquires AFS tokens needed to authenticate with the PTS server.
9. The group membership information acquired in the previous step, the reservation parameters, and a policy file are passed to the KeyNote policy engine that makes an authorization decision. If the request does not satisfy the cur-

rent security policy, an error is returned back to the client and the rest of the steps are not executed.
10. The setup_flow Expect script uses SSH to communicate with the appropriate network hardware and configures the network reservation.
11. 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).
12. Same as Step 7.
13. Same as Step 9.
14. Same as Step 10.

This design lets us express 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 may 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.

5 Related Work

The Globus MyProxy [16] initiative provides a trusted server to store 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 who collects passwords. He can obtain a certificate, signed by one of the default Certificate Authorities supported by browsers, and run a Web server, providing a spoofed login HTML form. However, by examining the credential, an activity most users do not bother doing, the user can tell the login form is spoofed.

The Grid Portal Architecture [11] is a Web interface to Grid Computing resources that uses MyProxy Services for client authentication. The GARA Web interface differs from the Grid Portal in several important ways. Access in our scheme is via done an https stream and requires mutual SSL authentication, which in turn requires a user certificate, thus obviating the need for users to type passwords in HTML forms, as it is done in the Grid Portal.

The Community Access Service (CAS) [15] 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. In the future, GARA client will contact the first GARA service, which in turn will contact other bandwidth brokers (BB), needed for the end-to-end reservation. Sander *et al.* discusses the bandwidth broker to bandwidth broker protocol in [18]. The Simple Inter-Domain Bandwidth Broker Specification (SIBBS) [19] 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.

Our choice of policy engines was influenced by the availability of working code. The modular design allows for use of other policy engines. Akenti [20], and GAA API [17] 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.

6 Conclusions

We have demonstrated a scalable distributed authorization service that joins local domain group services via a shared namespace, and asserts group membership by adding a signed authorization payload to existing communications.

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 end-point.

We have focused our distributed authorization service design on the ability to use existing local domain group services, firm in the belief that coalescing and maintaining replicas of user and group information does not scale and is an unnecessary administrative burden.

References

1. Andre Arnes. *Public Key Certificate Revocation Schemes.* PhD thesis, Norwegian University of Science and Technology, Kingson, Ontario, Canada, February 2000.
2. M. Blaze, J. Feigenbaum, J. Ioannidis, and A. Keromytis. The KeyNote trust management system version 2. RFC 2704, September 1999.
3. M. Blaze, J. Feigenbaum, and A. Keromytis. Keynote: Trust management for public-key infrastructure. In *Proceedings Cambridge 1998 Security Protocols International Workshop*, April 1998.
4. M. Blaze, J. Feigenbaum, and M. Strauss. Compliance checking in the PolicyMaker trust management system. In *Proceedings of Financial Cryptography*, February 1998.
5. R. Butler, D. Engert, I. Foster, C. Kesselman, S. Tuecke, and J. Volmer. A national-scale authentication infrastructure. *IEEE computer*, 33(12):60–66, December 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. Technical Report 01-2, Center for Information Technology Integration, University of Michigan, February 2001.
10. I. Foster, A. Roy, and V. Sander. A quality of service architecture that combines resource reservation and application adaptation. In *Proceedings of the 8th International Workshop on Quality of Service (IWQQOS 2000)*, June 2000.
11. Grid Computing Portal: `http://hotpage.npaci.edu/cgi-bin/hotpage_top.cgi`.
12. IETF Internet Traffic Engineering Working Group. `http://www.ietf.org/html.charters/ tewg-charter.html`.
13. 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*, August 2001.
14. C. Neuman and T. Ts'o. Kerberos: an authentication service for computer networks. *IEEE Communications*, 32(9):33–38, September 1994.
15. L. Pearlman, V. Welch, I. Foster, C. Kesselman, and S. Tuecke. A community authorization service for group collaboration. In *IEEE Workshop on Policies for Distributed Systems and Networks*, 2002. submitted.
16. MyProxy project. `http://dast.nlanr.net/ Projects/MyProxy`.
17. T. Ryutov and C. Neuman. Representation and evaluation of security policies for distributed system services. In *Proceedings of the DISCEX*, January 2000.
18. 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*, August 2001.
19. SIBBS. The simple inter-domain bandwidth broker specification. `http://qbone.internet2.edu/bb/`.
20. 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.