Negotiation of Multilateral Security Decisions for Grid Computing

Li Zhou Clifford Neuman
Information Science Institute
University of Southern California
{zhou,bcn}@isi.edu

Abstract

Grid computing enables the sharing of heterogeneous resources within virtual organizations. Since entities in virtual organizations are independently administrated, cross-domain management of access control and intrusion protection is critical for grid services. To provide better protection in grid systems, multiple domains collaborate in making security decisions. Unfortunately, the absence of global trust makes coordination of multilateral decisions difficult. In this paper, we introduce the Grid Access Control and Intrusion Protection Architecture that facilitates the collaboration in making security decisions for grid services. In particular, we present a design of Trust-based Security Variable that assists the negotiation of multilateral security decisions according to the level of trust among grid service providers.

1. Introduction

The grid is a large-scale distributed computing environment that enables the sharing and coordinated use of diverse resources, including computational resources, storage, information and other devices [1]. It supports the simultaneous use of a large number of resources, providing dynamic quality of service enforcement, co-allocation, resource discovery and other services for resource management [2][3].

The grid is built from the resources of a dynamic collection of individuals, institutions and enables the creation of "virtual organizations" [1]. Since these virtual organizations extend across multi-institutional domains, it is usually constructed without global trust. Therefore security issues of authentication, authorization, delegation, policy management and intrusion detection must be carefully considered to protect the shared resources from all possible security threats including unauthorized use of resources, misuse of resources, and denial of service.

Grid entities that run on different hosts may also be highly interrelated. Changes of the security context on one host may have effects on the security contexts of other hosts. If security decisions are made by individual services and on individual hosts, it becomes difficult to support many useful strategies such as global quota, dynamic lockdown and collaborative intrusion detection. This weakens the strength of protection on grid services. It is more desirable to apply security policies according to information shared among collaborative entities.

The Grid Access Control and Intrusion Protection Architecture (GACIPA) presents a generic framework to make multilateral security decisions and provides a means to customize collaborative policies. It provides for fine-grained access control, dynamic intrusion protection, collaboration of security decisions, and the maintenance of global security context. With the help of GACIPA toolkits, users can use text-based policy files to protect grid services with their customized strategies.

One of the most significant challenges in the grid environment is that entities do not fully trust one another. Although one relies on information from collaborators to strengthen its own security protection, it is not desirable to accept all the security proposals from other collaborators unconditionally. Since collaborators may make improper judgments, or even be intruders by themselves, collaboration without constraints is not only incapable of achieving better protection, but also likely to expose one to more potential threats.

Therefore, collaborators should make their multilateral security decisions according to the level of trust among the grid entities. For this purpose, we introduce the concept of trust-based security variable (TSV). The trust-based security variables are shared among a group of collaborative entities, but the grid entities do not keep a same value for each TSV. Instead, all the collaborators propose their own shares to the TSV. Then, each collaborator will integrate these shares into its own TSV value according to how much it can trust the other collaborators.

In summary, the major contributions of this paper are:

●Provide the design of fine-grained access control and intrusion protection for the grid services.

●Present the architecture of making multilateral security decisions among the collaborators.

●Propose the use of trust-based security variables to cope with the drawbacks of multilateral security decision under the absence of global trust.

In section 2, we introduce the protection of grid system through multilateral security decisions. Then, section 3 presents the basic framework of the Grid Access Control & Intrusion Protection Architecture. Design issues of the trust-based security variable are explained in section 4. Section 5 outlines the toolkit of the Generic Authorization & Access-control API (GAA-API), followed by an example of negotiating trust-based multilateral security decisions in Section 6. Section 7 contains the related works. Section 8 presents the implementation status, lists the future works and also summarizes the conclusions.

2. Multilateral security decision

Grid entities are distributed across different hosts, different networks, and different administrative domains. These grid entities may also be highly interrelated. It is very common that the same services (or similar services) run on many different hosts. When a user requests a service, he/she can choose any one (or several) of the service instances to work with.

Similar grid services may produce similar security requirements. When a certain behavior is regarded as insecure by a service on one host, the same behavior is likely to be insecure to similar services on other hosts, too. Thus, the feedback from a security decision on one host can contribute to the subsequent decisions that are made on other hosts. It is necessary for the distributed grid entities to collaborate with one another and protect themselves through multilateral security decisions.

Here are some examples of the multilateral security decisions that can provide better protection on the grid.

●**Global Quota**: In the grid, many resources such as CPU, disk storage, etc. are available on multiple hosts. Users may choose any host (or several hosts) to execute programs or store data. To prevent a user from consuming too many resources, we need to impose a quota of resource consumption on each user. If the quota only applies locally, the user can still accumulate a considerable number of resources by asking for a few from each of the service providers. Therefore, we should track each user on its total amount of resource consumption throughout the distributed grid system, and impose global quotas to reject requests for excessive consumption in real time.

●**Dynamic Lockdown**: Many intruders and viruses use random scans on a great number of computers and try to compromise the ones with particular security vulnerabilities. As one host identifies suspicious behaviors (such as DoS attack, buffer overflow attack, etc), the other neighbors in the grid may experience similar attacks. Since we are not sure whether all these independently administrated hosts can detect and protect themselves from such attack, anyone who has identified suspicions of attack should broadcast alerts to all the other collaborators. Then all the subsequent requests with the similar pattern (or from the same attacker) will be rejected by all the collaborative services.

●**Collaborative Intrusion Detection**: Some intrusions are difficult to detect by individual hosts or services. We have to identify them based on the global knowledge of the entire grid. For instance, in the grid environment, many services on different hosts may use the same

username/password list to authenticate the users. Intruders can scatter their attempts of password guessing to all the distributed hosts. With only a few authentication failures detected on a local host, each individual can hardly identify the distributed password guessing attack. Therefore, the grid entities should keep contact with each other and make their intrusion detection decisions in a collaborative way.

In the following sections, we will specify how these multilateral security decisions are made under the GACIPA framework and how the trust relationships are integrated into the decisions.

3. Framework overview

In the Grid Access Control & Intrusion Protection Architecture (GACIPA), we have two basic building blocks that serve for making multilateral security decisions. They are the Generic Authorization & Access-control Application Programming Interface (GAA-API) and the Global Security Context Agent (GSCA).

The GAA-API provides a generic library for fine-grained access control, intrusion detection and dynamic intrusion response. It is used to evaluate security policies and synthesize the sub-decisions from various security mechanisms, such as password files, certificates and security context into a final decision. Based on the decisions, it can perform auxiliary actions such as logging, notification and the update of the security context.

A stand-alone GAA-API can only make its security decision locally. It lacks the ability to negotiate with other GAA-API instances that run on remote hosts. Therefore it is incapable of providing collaborative protection on the grid system.

The Global Security Context Agent (GSCA) is introduced to manage trust-based security variables (TSV) and help the GAA-API with collaborative work. Through the exchange and maintenance of trust-based security variables, it establishes a global security context that extends beyond the boundaries of services, hosts and administrative domains. Thus, the GAA-API instances can communicate with one another and make multilateral decision by querying and updating trust-based security variables.

The GSCA also provides the integration of trust relationships into the trust-based security variable. The collaborators may have different values for a TSV. First, every collaborator proposes its own share to the TSV. Then, on each GSCA, according to the variable policy that defines how much one can trust the other collaborators, these shares of the TSV are integrated into its own TSV value for the local grid services to query.

Here we explain the procedure of how a security decision is made under the GACIPA framework. As Figure 1 shows, when a grid service receives a request (1), it calls the GAA-API library for an access control decision (2). During the evaluation of security policies, the GAA-API queries its local GSCA for the values of TSV to obtain its concerned global context (3). To provide feedback to affect other domains, the GAA-API may update some TSV via GSCA (4). Finally, the GAA-API returns its final security decision to the grid service (5).

After each update on the trust-based security variable, the local GSCA will exchange the update with the other

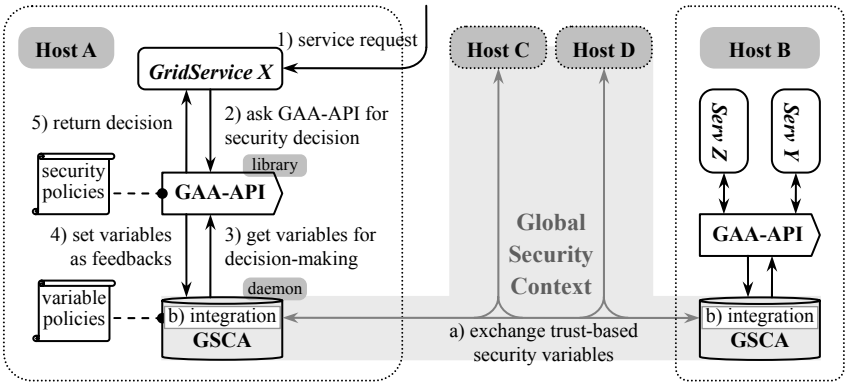

Figure 1: The GACIPA framework

collaborative GSCAs [(a)]. Then, every GSCA will integrate the update into its new TSV value according to its respective variable policies [(b)]. *(Detailed issues of the variable integration will be explained in Section 4.)*

With the help of variable exchange and variable integration, a global security context is established among these collaborative GSCAs. When a security decision is made on one host, its corresponding changes on the security context will be propagated to other GSCAs in real time. Therefore, subsequent security decisions on remote hosts can dynamically adapt to the feedbacks of previous decisions. This is how multilateral security decisions are made under the GACIPA framework.

4. Trust-based security variables

In the grid, collaborators share information for multilateral security decisions. However, no global trust exists among the collaborators. Since collaborators may make improper judgments, or even be intruders by themselves, we can not accept their security proposals unconditionally. To mitigate the conflict between the need for collaboration and the absence of global trust, we provide a design of trust-based security variable (TSV) which integrates the trust relationships into the sharing of security context.

A TSV is maintained among a group of collaborative hosts. For each variable, every collaborator keeps two separate values: **TSV-share** and **TSV-final**. A collaborator updates the TSV through its local TSV-share, and makes queries through its local TSV-final. The underlying communication among the collaborators and the integration of the trust relationship are made transparent to other grid services.

The maintenance of the trust-based security variable (TSV) consists of three phases:

●**Variable Proposing**: Firstly, every collaborator proposes its own share to the TSV, which is called TSV-share. Update operation only takes effect on its local TSV-share. So that different collaborator may keep different values as its TSV-share.

●**Variable Exchange**: Secondly, the TSV-shares will be dynamically exchanged among the collaborators, so that every collaborator can obtain all the latest values of TSV-shares from the other collaborators.

●**Variable Integration**: Finally, each collaborator will integrate the collective of TSV-shares into its own TSV-final. Again, the values of TSV-finals may vary from host to host. The integration of TSV depends on two factors:

(a) Variable policy: each collaborator has its respective policy, which defines how much one can trust the other collaborators. In general, the more you can trust a collaborator, the larger portion of the collaborator's TSV-share will you integrate into your TSV-final.

(b) Integration algorithm: each type of the TSV defines its respective algorithm for the variable integration. The algorithm takes the collective of TSV-shares together with the local policy as input, and derives the local TSV-final as output.

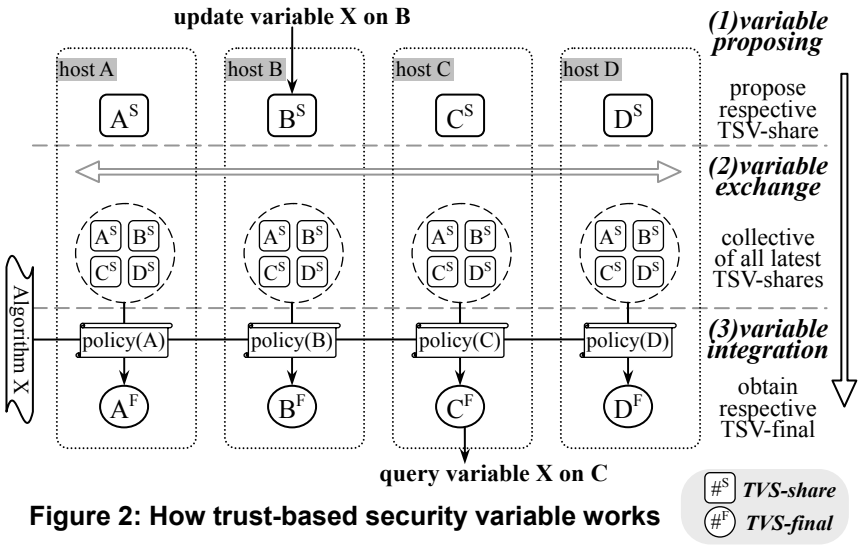

Figure 2: How trust-based security variable works

Figure 2 presents an example in which a trust-based security variable X is maintained among 4 collaborative hosts: A, B, C and D. For the variable X, they propose their respective TSV-shares: A^S, B^S, C^S and D^S. Through the exchange of variable, every collaborator gathers the latest values of all these 4 TSV-shares. Then, according to their respective variable policies, the collaborators will integrate their own TSV-finals: A^F, B^F, C^F and D^F.

In section 4.1 and 4.2, two practical examples will show us how the variable policy and the integration algorithm can be applied to the trust-based security variable, and why they are helpful to negotiate multilateral security decisions under the absence of global trust.

4.1 Trust-based global quota

In the grid, one of the most commonly used security decisions is to restrict the number of resources that are consumed by each user. For a resource that is available on multiple hosts, if the quota only applies locally on each individual host, a user can still obtain a considerable number of resources by asking for a few from every service provider. Therefore, it is necessary to impose a global quota, which restricts each user by total amount of resource consumption throughout the grid system.

However, we do not fully trust all our collaborators. Some collaborator, with the purpose of blocking a user maliciously, may generate a false claim that it has allocated a very large number of resources (which is larger than the global quota) to the user. If all the collaborative service providers accept the false claim unconditionally and believe that this user has exceeded its quota for the resource consumption, then the user can allocate no more resource from any of the service providers, although the actual number of resources he/she has obtained may be far below the quota.

We use one type of the trust-based security variable, *limited-global-quota*, to mitigate the threat of false claim attacks. By using this TSV, one does not unconditionally accept the shares of global quota from the other collaborators. Instead, the variable policy assigns an upper limit of the TSV-share to each collaborator. In general, if you fully trust a collaborator, you can assign "+∞" as its limit. Otherwise, the less you can trust a collaborator, the smaller is the upper limit. When someone purposes a TSV-share that is beyond its upper limit, your integration algorithm will only counts for the value of upper limit into your local TSV-final.

If a collaborator claims that a large number of resources are consumed by a user, which is suspicious to be a false claim attack on the user, and if you do not trust that collaborators very much, the trust-based security variable can mitigate this claim to a degree that you can accept. So the smaller the limit you assign to a collaborator, the less chance will this collaborator have to block any users on your host.

Figure 3 presents an example of global quota, which confines the number of processes that are allocated to a user. We have 6 hosts in the collaborative group. They claim their respective number of processes that are allocated to this user: A^S=0, B^S=15, C^S=2, D^S=3, E^S=1, F^S=5.

In this example, the TSV-share that B acclaims is especially large. As C trusts in B, it puts no limit on the

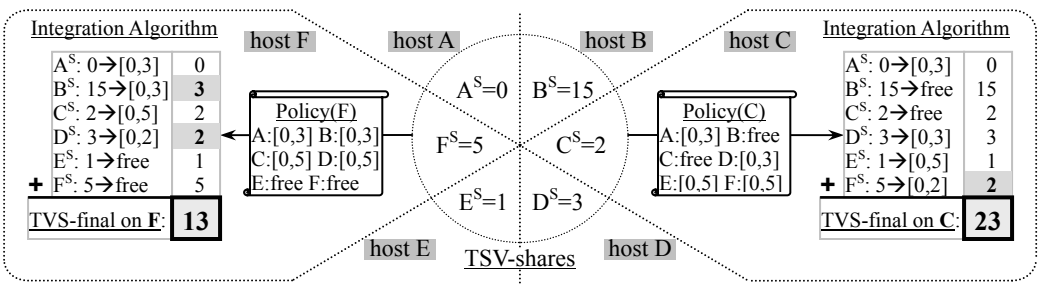

Figure 3: Trust-based security variable - global quota

TSV-share(B). Therefore the TSV-final on C is as large as 23. In contrast, F has much less trust on B, it uses [0,3] as the limits on TSV-share(B). So the contribution of B to its global quota is reduced from 15 to 3, and the TSV-final on F is only 13. Similarly, A, B, D and E will also define their own limits and calculate their own TSV-final according to how much they trust the other collaborators. In this way, potential attacks from unreliable collaborators are contained by the trust-based security variable.

4.2 Trust-based dynamic lockdown

Dynamic lockdown is another type of the multilateral security decision, which helps to protect the grid. As one host identifies suspicious behavior, such as DoS attack, buffer overflow attack, etc., other neighbors may also experience similar attacks. Therefore, it is necessary to exchange the intrusion alerts among the collaborators. After anyone has identified an attack, all the subsequent behaviors with similar pattern can be dynamically contained by the other collaborators.

However, the absence of global trust still leads to apparent weakness. If a collaborator, with the purpose of blocking a user maliciously, generates false intrusion alerts, the user will be banned throughout the collaborative group, although he/she may have done nothing improper.

Multilateral decisions for dynamic lockdown should refer to 3 factors, the degree of intrusion suspicion, the number of reported alerts, and the trust rates on the collaborators. If the suspicion level is higher; or we trust more on the collaborator that reports the alert; or similar intrusion alerts against a same user have been reported by many collaborators, then we are more likely to treat this user as an intruder. On the contrary, if only one (or a few) has reported the alert, and we do not trust it very much, we may ignore this alert or wait until further judgment is made.

Figure 4 presents an example of dynamic lockdown. We have 4 hosts in a collaborative group: *A*, *B*, *C* and *D*. They inspect on 4 users: *w*, *x*, *y* and *z*. Every collaborator evaluates each user on its ***intrusion suspicion***. As the table of intrusion suspicion in Figure 4 shows: on host *A*, the intrusion suspicion of user *z*: $S_z(A)$ is assigned to 0, which means no suspicion is found. $S_w(A)$ and $S_y(A)$ are assigned to 0.5, which means *A* is 50% sure that *w* and *y* are intruders. $S_x(A)$ is assigned to 1, which means *A* has confirmed *x* as an intruder. Similarly, *B*, *C* and *D* will also rate their respective intrusion suspicions on *w*, *x*, *y* and *z*.

The variable policy of host *H (H=A, B, C or D)* consists of two parts, the ***trust rates*** on each collaborator *k*: $R_H(k)$, and the ***threshold*** value to regard a user as intruder: T_H. For each user *u*, we multiply the intrusion suspicion $S_u(k)$ and the trust rate $R_H(k)$ of every collaborator *k*, and sum up the products to the ***overall intrusion rate*** of user *u*: $O_H(u)$. When $O_H(u)$ is equal to or larger than the threshold T_H, the user *u* becomes a member of the ***intruder list*** on *H*, so that dynamic lockdown will be performed against *u* on host *H*. Here is the formula that calculates the intruder list on host *H*.

$$I_H = \{ u \mid \sum_{k=A,B,C,D} S_u(k) \times R_H(k) \geq T_H \qquad u = w, x, y, z \}$$

Again, according to the different trust rates and different thresholds, the TSV intruder list may vary from

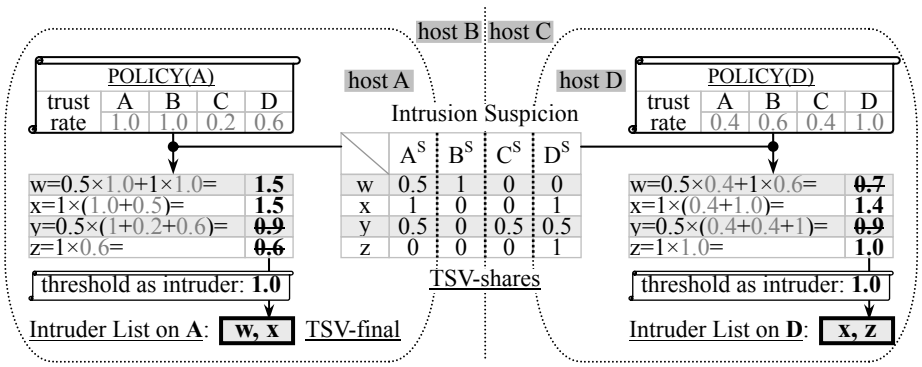

Figure 4: Trust-based security variable – dynamic lockdown

host to host. In the example given above, *A* takes *w* and *x* as the intruders, but *D* takes *x* and *z* instead. With the help of trust-based security variable, the intrusion alerts with lower suspicion, or from less trustful collaborators will count for less to the final intruder lists. Therefore potential threats of the false claim attack are contained.

5. GAA-API: from policy to decision

With the help of the trust-based security variable, we can establish a global security context among the collaborators. In this section, we will explain how the Generic Authorization & Access-control Application Programming Interface (GAA-API) [4] provides a simple and straightforward way to make access control, intrusion protection and other security-related decisions.

For a grid service, security decisions may be based on security attributes such as user identity, request right, user certificate, security context, etc. These attributes may be obtained or validated using various security mechanisms. The final decision is synthesized from the sub-decisions of these security mechanisms under a certain logic model.

It is very common that similar security mechanisms are used by many different services. However, each grid service has its specific requirement for access control. The services may employ different combinations of security mechanisms, use different parameters to evaluate the same security mechanism, or apply different logics to synthesize their final decisions.

If we write independent models of access control for each grid service, the efficiency of software development will be quite low. Moreover, when the administrator needs to introduce a new security mechanism to protect the service, or reconfigure the security strategies, he/she may have to revise the code of the access control module, which increases the cost of system maintenance.

The GAA-API provides us a generic and flexible toolkit for all the grid services to make security decisions. It supports fined-grained access control and application level intrusion protection. The GAA-API can bring us the following benefits:

- Service developers can implement the modules of access control and intrusion protection by simply calling the GAA-API library together with a few service-specific or request-specific parameters.
- System administrators can flexibly customize the service protection by simply configuring the text-based GAA-API policy file. They can also freely add new security mechanism and apply different logic of decision-making through the GAA-API policy.
- Working together with trust-based security variables, the GAA-API can dynamically adapt its service protection according to the changes of the global security context, and make multilateral security decisions among the GAA-APIs on different services and different hosts.
- The GAA-API can also perform various auxiliary actions, such as accounting, logging, email notification, and the update of global security context (via trust-based security variable) according to the GAA-API policies.

The basic structure and the evaluation process of the GAA-API policy are listed here:

Condition: The basic unit in the GAA-API policy file. With the domain-specific security requirements that are defined by condition attributes, each condition evaluates a certain security mechanism, and returns whether the security requirement of this condition is met or not.

Action: Each condition may also have several actions attached. According to the different result from their parent condition, different actions may be executed to do different auxiliary jobs.

Policy: A policy consists of several conditions. For each policy, the GAA-API evaluates from its first condition towards the last condition. A policy grants a request if and only if all its conditions are met. Otherwise, the policy is not satisfied.

PolicySet: The GAA-API policy file defines a set of policies. When the GAA-API receives a user request, it evaluates the policies that match the request on its service name and access right one by one until it finds a policy that grants this request. If no policy is satisfied, the

GAA-API should reject this request or ask for further user proof.

6. An example scenario

Assume that in a grid, we have a number of grid servers providing the service of remote file storage. After authentication with a password, a user can choose any one or more server to upload, download and delete his/her own files.

We should consider the following security issues to protect the remote-file-storage services: First, to prevent a user from occupying too much disk space, we should restrict the total amount of storage that a user is allowed to occupy over the entire grid. Secondly, since we use a password to authenticate a user, the servers should collaborate to fight against distributed password guessing attacks.

Since the servers do not fully trust each other, every service should be aware of potential threats from the other collaborative services. Some service may generate false claims that a user has allocated a large amount of disk space, or the user is doing password guessing attack. If all the other services accept the false claim unconditionally, this user will be banned universally although he/she does nothing improper.

We use the GSCA (Global Security Context Agent) together with the GAA-API to protect the remote-file-storage services from these potential threats. As the security policies in Figure 5 show, two trust-based security variables are maintained by the GSCAs. *$G:attackers* keeps the list of suspicious attackers (as section 4.1 explains), and *$G:storage* keeps the total amount of storage allocated by each user (as section 4.2 explains). In each decision-making process, the remote-file-storage service also provide the GAA-API with 3 request-specific parameters: *$R:user*, *$R:passwd* and *$R:reqspace*, which represent the username, password and the requested disk space.

As a remote-file-storage server *H* receives a file-upload request from user *X*, it calls the GAA-API to evaluate the security policies shown in Figure 5 for access control decision. The GAA-API finds the policy *(#2~13)* that matches on the service name "remote-file-storage" and the access right "upload", and evaluates the conditions of this policy one by one.

Dynamic lockdown: First, the GAA-API checks whether the user *X* is not in the global attacker list *$G:attackers (#3)*. If the condition is not met *(when="fail")*, which means the server *H* regards *X* as an attacker, an alert message will be added to the system log file *(#4)*, and the GAA-API will stop the evaluation of this policy with the result "*unsatisfied*". Since we do not have any more policy for the file upload, the GAA-API returns "*rejected*" as its access control decision. Otherwise, if the condition is met, the GAA-API will go on evaluating the next condition.

Collaborative Intrusion detection: Secondly, the GAA-API checks if the user *X* provides the right password

```
#1  <PolicySet xmlns"urn:gacipa:1.0:policyset" scope="global">
#2      <Policy service="remote-file-storage" right="pre-allocate, upload">
#3          <Condition name="check_variable"> <Value> $R:user NOT IN $G:attackers </Value>
#4              <Action when="fail" name="system_log" value="ban $R:user as suspicious attacker" />
#5          </Condition>
#6          <Condition name="authenticate_user" type="password" value="$R:user/$R:passwd">
#7              <Action when="fail" name="set_variable" value="$G:attackers[$R:user].suspicion+=0.2" />
#8              <Action when="succ" name="set_variable" value="$G:attackers[$R:user].suspicion=0" />
#9          </Condition>
#10         <Condition name="check_variable" value="$G:storage[$R:user]+$R:reqspace<=5G" />
#11         <Condition name="check_variable" value="$G:storage[$R:user].TSVshare+$R:reqspace<=1G" />
#12         <Action when="succ" name="set_variable" value="$G:storage[$R:user]+=$R:reqspace" />
#13     </Policy>
#14     <Policy service="remote-file-storage" right="download"> … … </Policy>
#15     <Policy service="remote-file-storage" right="delete"> … … </Policy>
#16 </PolicySet>
```

dynamic lockdown (#3-5)

collaborative intrusion detetion (#6-9)

global quota (#10-12)

Figure 5: An example of the GAA-API policy

(#6). For each time the authentication fails, *H* will increase its attacker suspicion on *X* by 20% *(#7)*. With the help of trust-based security variable *$G:attackers*, the alert of suspicion prevails to all the collaborative services. If too many authentication failures occur, all the services may ban this user *(#3)* to contain the suspicious password guessing attack. On the contrary, if *X* succeeds in authentication, *H* will reset its local share of the attacker suspicion to zero *(#8)*.

Global Quota: Thirdly, the GAA-API checks whether the disk space that *X* consumes exceeds its local quota *(#11)* and global quota *(#10)*. As for the queries on global quota, the GAA-API reads the value of TSV-final: *$G:storage[$R:user]*, which represents the total number of disk space over the collaborative group. If we concerns about the local consumption of storage instead, the GAA-API reads the value of TSV-share: *$G:storage[$R:user].TSVshare*.

When all the three conditions are met *(when="succ")*, the GAA-API will increase its local share of *$G:storage[$R:user]* by the size of newly requested disk space *(#12)*, and return "*granted*" as its access control decision to the remote-file-storage service.

In this example, we have a number of remote file storage services using the similar policies. They all can propose their own shares to the global quota and the attacker list. As we explain in section 4.1 and 4.2, each collaborator will derive its respective TSV-final and make its own security decision according to how much it can trust the other collaborators.

In addition, user requests for download and deletion are handled in a similar way *(#14-15, for space constraint, we will show the details of these 2 policies in Figure 5)*. However, it is not necessary for download and deletion to check for the quotas on storage size, because they do not allocate any new disk space. We also need the deletion operation to reduce the value of *$G:storage[$R:user]* after releasing the disk storage.

7. Related work

There is much research that relates to the security of grid computing and decision-making for access control. In this section, some of the efforts are briefly introduced.

The Grid Security Infrastructure (GSI) [5], which is a part of the Open Grid Service Architecture (OGSA), proposed a framework of authorization, authentication, delegation and the other security-related issues for the grid computing. It allows each grid entity to enforce its specific security requirements, security mechanism and security policies. But the GSI does not include the issue of collaborative access control and intrusion protection.

The eXtensible Access Control Makeup Language (XACML) [6][7] provides a generic schema of the access control policy. We can use the XACML to make access control decision by matching the XML contents of the request and the policy. The GAA-API policy is similar to the XACML in schema. While the GAA-API further supports multilateral access control and intrusion protection. In further, comparing to the XACML, the GAA-API can manage more complicated cases in decision-making.

The SECURE project [8] demonstrates how the access control system and the security policy can make decisions on the basis of trust relationship and risk analysis. The framework is applied to the file storage and publication service on the grid system. But it lacks the ability to make collaborative negotiation and dynamic adaptation according to the changes of the global security context.

8. Implementation status and future work

The GACIPA framework consists of two building blocks: the Generic Authorization and Access-control API (GAA-API) and the Global Security Context Agent (GSCA). We have already implemented and released the GAA-API [9]. The GAA-API is successfully integrated into the Apache Web Server [10][11], OpenSSH [12][13], TrustBuilder [14][15] to strengthen their security protection.

The GSCA is still under construction. We are

implementing the GSCA with secure communication channels, alternative communication models *(centralized, master/slave, peer-to-peer, etc)*, real-time collaborative group management and automatic policy distribution. We are also providing many types of the trust-based security variables to support most of the commonly-used multilateral security decisions. We have seen two of these described in this paper.

We are also integrating the GAA-API and the GSCA into the Globus Toolkit [16], a widely-used grid middleware that conforms to the standard of Open Grid Service Architecture (OGSA), so that we can provide better security protection on the grid system that are developed by the Globus Toolkit.

In summary, this paper presents a generic framework for the negotiation of multilateral security decision, such as global quota, dynamic lockdown and collaborative intrusion detection. Such decisions among collaborative entities provide better protection on the system. We also design the trust-based security variable that can contain the drawback of multilateral security decision under the circumstance of no global trust among the collaborators, so that every grid service can make its respective decision according to how much it can trust the other collaborators.

Reference

[1] I. Foster, C. Kesselman, S. Tuecke, The Anatomy of the Grid: Enabling Scalable Virtual Organizations**.** *International J. Supercomputer Applications*, 15(3), 2001.

[2] I. Foster, C. Kesselman, G. Tsudik, S. Tuecke, A Security Architecture for Computational Grids. *Proc. 5th ACM Conference on Computer and Communications Security Conference*, pp. 83-92, 1998.

[3] I. Foster, C. Kesselman, J. Nick, S. Tuecke, The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. Open Grid Service Infrastructure WG, Global Grid Forum, June 22, 2002

[4] T. Ryutov and C. Neuman, "The Specification and Enforcement of Advanced Security Policies", *In Proceedings of the Conference on Policies for Distributed Systems and Networks (POLICY 2002)*, June 5-7, 2002, in Monterey, California.

[5] I. Foster, C. Kesselman, G. Tsudik and S. Tuecke, "A Security Architecture for Computational Grids", *Proceeding of the. 5th ACM Conference on Computer and Communications Security Conference*, pp. 83-92, 1998.

[6] S. Godik, T. Moses, et al, "eXtensible Access Control Markup Language (XACML) Version 1.0", *OASIS Standard*, February 18th, 2003

[7] M. Lorch, S. Proctor, R. Lepro, D. Kafura and S. Shah, "Access control: First experiences using XACML for access control in distributed systems", *Proceedings of the 2003 ACM workshop on XML security*, October 2003

[8] N. Dimmock, A. Belokosztolszki, D. Eyers, J. Bacon and K. Moody, "Access management for distributed systems: Using trust and risk in role-based access control policies", *In Proceedings of the ninth ACM symposium on Access control models and technologies*, June 2004

[9] http://www.isi.edu/gost/info/gaa-api

[10] T. Ryutov, C. Neuman, D. Kim and L. Zhou, "Integrated access control/intrusion detection for securing web server", *IEEE Transactions on Parallel and Distributed Systems, Vol. 14, No. 9, September 2003.*

[11] Apache Web Server, http://www.apache.org

[12] T. Ryutov, C. Neuman and D. Kim, "Dynamic Authorization and Intrusion Response in Distributed Systems", *In Proceedings of the 3rd DARPA Information Survivability Conference and Exposition (DISCEX III)*, Washington, D.C. April 22-24, 2003.

[13] OpenSSH, http://www.openssh.org

[14] T. Ryutov, L. Zhou, C. Neuman, T. Leithead and K. Seamons, "Adaptive Trust Negotiation and Access Control".

[15] TrustBuilder, http://isrl.cs.byu.edu/

[16] Globus Toolkit 3, http://www.globus.org.

[17] V. Welch, F. Siebenlist, I. Foster, J. Bresnahan, K. Cajkowski, J. Gawor, C. Kesselman, S. Meder, L. Pearlman and S. Tuecke, "Security for Grid Services". *Twelfth International Symposium on High Performance Distributed Computing (HPDC-12)*, IEEE Press, June 2003.

[18] L. Zhou and C. Neuman, "Federated Access Control and Intrusion Detection for Grid Computing", *USC Internet and Grid Computing Laboratory Technical Report*, May 26, 2004.