Adaptive Trust Negotiation and Access Control for Grids

Tatyana Ryutov, Li Zhou, Clifford Neuman, Noria Foukia
Information Sciences Institute
University of Southern California
{tryutov, zhou, bcn, foukia}@isi.edu

Travis Leithead, Kent E. Seamons
Internet Security Research Lab
Brigham Young University
{tleithea, seamons}@cs.byu.edu

***Abstract*— Access control in computational grids is typically provided by a combination of identity certificates and local accounts. This approach does not scale as the number of users and resources increase. Moreover, identity-based access control is not sufficient because users and resources may reside in different security domains and may not have pre-existing knowledge about one another. Trust negotiation is well-suited for Grid computing because it allows participants to establish mutual trust based on attributes other than identity.**

The Adaptive Trust Negotiation and Access Control (ATNAC) framework addresses the problem of access control in open systems by protecting itself from adversaries who may want to misuse, exhaust or deny service to resources. ATNAC is based on the GAA-API, which provides adaptive access control capturing dynamically changing system security requirements. The GAA-API utilizes TrustBuilder to establish a sufficient level of trust between the negotiating participants, based on the sensitivity of the access request and a suspicion level associated with the requester. A federated security context allows Grid participants to communicate their security appraisal and make judgments based on collective wisdom and the level of trust among them. We plan to apply ATNAC techniques to negotiation agreements in virtual organizations and P2P environments.

I. Introduction

As the grid transitions from a purely scientific community to a heterogeneous, commercial, open community, it enters an environment of mutual suspicion. Malicious users can damage the grid by stealing sensitive information, corrupting data, and exhausting grid resources by severely exceeding the allocated resources (quota). Grid users are concerned with the privacy, confidentiality and integrity of their data.

Access control in computational grids is typically provided by a combination of identity certificates and local accounts [1]. Publicly available access control policies specify which users have access to what resources. Users are generally required to pre-register with a service provider before requesting a service. This approach does not scale as the number of users and resources increase and user population becomes highly dynamic. Furthermore, users and resources may be from different security domains with no pre-existing knowledge about one another. Digital certificates help to address some of these issues, but they do not guarantee that resources or users can be trusted.

Trust negotiation [2] is a promising approach for establishing trust in an open environment and it is well-suited for grid computing because it allows participants to dynamically establish mutual trust based on attributes other than identity. Trust is based on user's attributes rather than identity, removing the necessity for pre-registration to gain access to a system. The use of unforgeable digital attribute credentials signed by trusted issuers allows users in different security domains to exchange credentials containing attributes in an effort to establish mutual trust. Trust negotiations establish higher levels of mutual trust as access control policies for increasingly sensitive credentials are satisfied. In successful negotiations, credentials are eventually exchanged that satisfy the policies.

Therefore, before Grid users and resources can cooperate, each must negotiate and meet the authentication requirements of the other. Since neither party is previously known, both may be reluctant to reveal sensitive personal attributes until the other party satisfies certain trust requirements. Thus, secure Grid computing must support access control policies that regulate not only resource access, but also the disclosure of sensitive user information to strangers.

We developed an ***Adaptive Trust Negotiation and Access Control (ATNAC)*** framework to help solve the problems of access control in open systems. The framework is based on two well established systems. The GAA-API [3], [4], [5] provides adaptive access control that captures dynamically changing system security requirements. The TrustBuilder [6] system regulates when and how sensitive information is disclosed to other parties. In this paper, we describe the synthesis of these two systems into one access control architecture for open grid environments. This combination extends the capabilities of each system.

In particular, the framework allows us to detect and thwart certain attacks on Grid transactions, adapt information disclosure and security policies according to a suspicion level, and to support cost effective trust negotiation, such that TrustBuilder is invoked only when negotiation is required by access control policies.

The framework adapts the security policies according to the sensitivity of an access request and a ***suspicion level*** (SL) associated with the requester. The ATNAC framework detects suspicious or malicious activity by examining a requestor's behavior patterns and TrustBuilder's failure codes. If any suspicious event is identified, security policies adapt in real time to dynamically protect the system from potential threats.

For instance, a misbehaving user may trigger an increased suspicion level, be denied access, or banned outright. A federated security context allows Grid participants to communicate their security appraisal and make access control decisions based on collective wisdom.

Traditional trust management solutions [7] do not adequately address dynamic aspects of trust. The preconfigured, coarse and static specification of trust in conventional systems is not consistent with human intuitions of trust [8]. An individual's opinion of another entity evolves based on available evidence. Thus, trust relationships evolve over time and require monitoring and reevaluation. Trust negotiation (TN) systems must support the monitoring of conditions on which trust evaluation is based, dynamically update those trust ratings, and modify system protection levels as a result.

It is difficult to give an exact definition of trust due to its complex subjective nature. However, we believe that trust can be explicitly modelled using the concepts of agreements and suspicion levels. Trust reflects beliefs an entity a has about entity b, that b will act in a certain way, based on information about b's attributes (such as competence, technical capabilities, and skills) and/or recommendations from trusted entities. Trust is inherently linked to risk: a is vulnerable if b does not fulfill a's expectations. The concept of an agreement can be used to make an explicit declaration of the expected participants' behavior. the agreement may include sanctions that are applied in the case of non-conformance to the agreed behavior. Agreements are created during an agreement negotiation process where participants iteratively exchange agreement proposals. The process is guided by the participants' private interests expressed in local policies.

A suspicion level is the means of revoking previously established trust (as declared in an agreement). The suspicion level depends on history (past transactions with the party) and is determined by monitoring the following aspects of participants' behavior:

1) The "lack of goodwill" is manifest by signs of non-cooperation and proactive opportunistic behavior during the agreement negotiation process.
2) Non-compliance represents the belief that a participant is not acting reliably according to the established agreement, possibly exploiting individual objectives and not conforming to the agreed policies.

The focus of this paper is to support adaptive, fine-grained access control and trust negotiation on the Grid in the presence of deliberate or inadvertent attacks. Attacks may come in the form of malicious, infected or badly written code, attempts to obtain sensitive information, or Denial of Service (DoS) attacks against trust negotiation using its underlying protocols. The techniques outlined in this paper may be applied to protect requesters from malicious service providers; however, because this paper focuses on protecting the service provider, our examples exhibit some degree of asymmetry in the trust placed on the participating parties.

Additionally, techniques outlined in this paper could be used for trust management in other environments including e-business, P2P systems, and virtual organizations (VO). These dynamic and temporal environments present the following trust management challenges:

- Temporary relationships, as opposed to long-lived, promote a "take and run" behavior.
- Parties may participate in a task without previous knowledge of the other participants.
- Competitive or adversarial interactions are characteristic to e-business and inter-organizational interactions in general.
- Negotiation of agreements may involve sensitive participant's information. For example, participants may require that certain attributes, such as a participant's identity or their participating vendors, be hidden during negotiations to ensure fairness and equal opportunity. In this case, attributes are revealed at the end of the negotiation or when a required level of trust is established.
- Information regarding collaborators may itself be sensitive, and disclosure of a list of acceptable members of a federation may not be releasable to members of the federation.

We believe that trust in VOs can be explicitly modeled using the concepts of agreements and suspicion levels.

II. The ATNAC Framework

The ATNAC framework shown in Figure 1 is based on two software components: TrustBuilder and the GAA-API. TrustBuilder employs X.509v3 digital certificates to describe attribute credentials and enforces sensitive security policies described using TPL [9]. XML-based TPL policies allow the specification of groups for required credential attributes. GAA-API enforces access control policies expressed in the Extended Access Control List (EACL) format [4] to govern access to Grid public and sensitive resources, services, operations, and policies. EACL is a simple language that describes security policies that govern access to protected resources. An EACL specifies positive and negative access rights with an optional set of associated conditions that describe the context in which each access right is granted or denied.

A web interface becomes a gateway for users to access Grid services in an environment that hides the complexity of a heterogeneous, distributed backend. Users submit and collect results for their jobs on remote resources through this interface. Motivated by the importance of web servers for grids, we have integrated the GAA-API with the Apache Web server. In the GAA-API/TrustBuilder integration, a typical session involving trust negotiation proceeds as follows:

The service requester sends the request to the service provider **(1)**. The request is forwarded to the GAA-API. The GAA-API fetches the required credentials from the database. The API uses the credentials to evaluate the policy and either grant or deny the request **(6)**. If the required credentials are not available, or the system threat level is high, the API submits the request for required credentials to the client **(2.1)**. The Analyzer calculates the suspicion level for the particular requestor/connection. Next, the GAA-API passes the suspicion

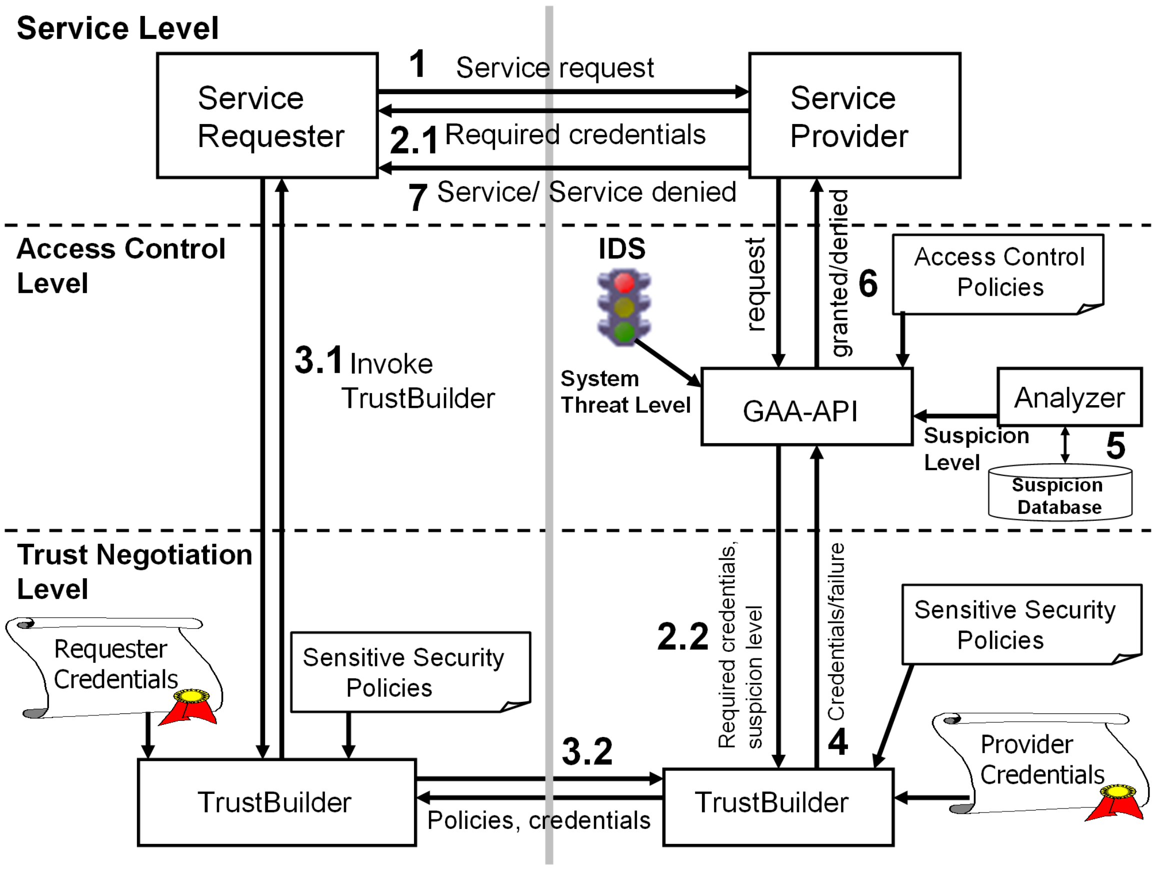

Fig. 1. The ATNAC Framework

level and a list of required credentials to TrustBuilder **(2.2)**. The requester receives a response requiring further proofs **(2.1)** and invokes a trust negotiation with the server **(3.1)**. TrustBuilder obtains the requested client credentials through a trust negotiation, guided by the suspicion level **(3.2)**. After a successful negotiation, the verified credentials are passed to the GAA-API **(4)**. The Analyzer checks the session history in order to identify active attacks or potential threats. The Analyzer stores this information in the Suspicion Database using the IP address or username as the index key **(5)**. The updated suspicion level is used by the GAA-API to process future requests from the same requester. The GAA-API uses the negotiation results to grant access to the service requester **(7)**.

III. Suspicion Level

The ATNAC framework provides monitoring and policy adaptation at the access control and trust negotiation levels. Our approach is based on assigning each monitored entity a suspicion level (SL). The SL indicates how likely it is that the requester is acting improperly.

ATNAC maintains a separate SL for each requester (host or user) that requests a service. In order to protect the system against spoofed IP addresses, we use either a requester's verified host identity (e.g., host-based authentication via X.509 certificates) or a verified user identity (if the access control policy requires one) to attribute the SL. The ATNAC framework stores this information in the *Suspicion Database*. The SL itself is comprised of four components:

$$SL =< S_J, S_{IL}, S_{DoS}, S_O >$$

S_J indicates whether the jobs run by a user conform to the imposed limits. S_{IL} is attributed to sensitive information leakage attempts. S_{DoS} indicates a probability of DoS attack on behalf of the requester. Finally, S_O indicates other suspicious behavior (e.g., misuses of a user's identity or impersonation attempts). The Analyzer increases the SL for each occurring suspicious event and decreases the SL when "positive" events occur (e.g., successful trust negotiation and/or successfully completed grid transaction). The value by which the SL is increased depends on the degree of confidence in the judgment that the event indicates malicious activity.

The framework also maintains a federated SL to help detect a highly distributed attack. The individual SLs (associated with a user ID or IP address) may not reach a critical threshold to identify a distributed attack, but a federated SL may be able to capture this kind of event.

The next session discusses the calculation of the S_J component of the SL. See Ryutov et al. [10] for discussion of the calculation of S_{IL}, S_{DoS}, and S_O.

A. Evaluating S_J

Typically, a user request to execute a job (e.g., using Resource Specification Language [11]) specifies the binary to be executed, and additional parameters such as required memory, CPU or number of processors needed. The ATNAC framework applies different policies according to the estimated job requirements.

A user may intentionally specify incorrect job requirements for a range of motives. For example, a user may underestimate the job execution time to get an earlier job start, or underestimate memory and processing requirements to get a cheaper admission price if payment is involved. As a countermeasure, the system must inspect the actual resource consumption during or after the execution of each job.

In practice, it is very hard to accurately estimate the resource consumption before a job is executed. Resource consumption attributed to a single job may vary significantly due to different inputs and/or diverse working environments.

Therefore, the system must allow the actual resource consumption to exceed the estimated values by some degree. Only when the user repeatedly keeps underestimating the requested resources, should this behavior be considered suspicious and impose more stringent policies. The behavior may be a result of careless code writing or due to code infection. In all cases, the system must be protected against noncomplying jobs.

To do this, ATNAC employs the Analyzer to compare the estimated cost $EC_M(J)$ and the actual cost $AC_M(J)$, where J represents a job and parameter M represents memory, storage, bandwidth, CPU, etc. Then the Analyzer derives the underestimation rate (in the form of an ordered list) for the job $UE(J)$. A positive $UE(J)$ value represents underestimation and a negative value represents overestimation. Finally, the Analyzer aggregates the underestimation rates of all the jobs that each user U has submitted and derives S_J for the user:

$$S_J = S(X,U) = \frac{\sum_{J \in jobs(U) \cap jobs(X)} UE(J) \times \alpha^{now-t(J)}}{\sum_{J \in jobs(U) \cap jobs(X)} \alpha^{now-t(J)}} \tag{1}$$

The coefficient α is a decimal value slightly smaller than 1.0 (e.g. 0.99), which makes the aggregation of $UE(J)$ time-digressive. A recently occurring job produces a higher S_J.

IV. Federated Security Context (FSC)

Grid entities may be highly interrelated and distributed across different hosts, networks, and administrative domains. 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. Therefore, when an entity perceives a threat, it should distribute warnings to the other entities that could be affected by that threat. The framework maintains SLs that are shared among a group of grid entities in order to communicate their security appraisal and make judgments based on collective wisdom.

The Federated Security Context (FSC) is designed to transparently negotiate multilateral decisions among a group of collaborative entities without global trust. The FSC fits an environment where a number of independently administrated entities need to share and aggregate information to strengthen their security protection. However, one cannot fully trust the information received from another entity without imposing a constraint. Thus, each entity aggregates the information based on how much they trust each other. The degree to which X trusts Y is defined by a trust rating $TR(X,Y)$, a value between 0.0 and 1.0.

The FSC agents among a group of collaborators maintain a list of Trust-based Security Variables (TSVs). Every collaborator keeps two separate values for each TSV: TSV-share and TSV-federate. TSV-share reflects a local view of the running state and TSV-federate reflects a global view of the system-wide running state which is aggregated from the TSV-share values according to the trust rating for all collaborators. Every collaborator updates the TSV through its TSV-share value, and queries the TSV through its TSV-federate value. The underlying communication among the collaborative entities and the determination of the federated trust values is transparent to any other web services.

The maintenance of the TSV consists of three phases as depicted in Figure 2. First, through a write operation, every collaborator proposes its own share to the TSV (TSV-share) (1). Secondly, the TSV-shares are dynamically exchanged among the collaborators (2). Finally, each collaborator integrates the collection of TSV-shares into its own TSV-federate according to how much each can trust the other (3).

The TSV aggregates local suspicion levels (TSV-share) into federated suspicion levels (TSV-federate). Greater trust in a collaborator implies more trust in the collaborators local SL. Thus, each entity derives its respective federate SL for user U according to the trust rating it assigns to the other entities that comprise its collaboration group CG. The following formula specifies how this is accomplished.

$$SL^{federate}(X,U) = \frac{\sum_{Y \in CG} SL^{local}(X,U) \times TR(X,Y)}{\sum_{Y \in CG} TR(X,Y)} \tag{2}$$

Also, the TSV imposes a global quota on the number of allocated resources in the following job submission example. Suppose a malicious grid server propagates fake claims that it has allocated a number of resources to a user. If all the other servers unconditionally accept the fake claims, the user may be blocked throughout the collaborative group. To mitigate this threat, the framework imposes a global quota based on trust ratings. The allocated resource claims from a server will be mitigated by an upper limit value that is proportional to the trust rating of this server. The following formula illustrates how to calculate the total number of allocated resources (AR) based on the global quota (GQ) and the trust rating (TR).

$$AR^{federate}(X,U) = \sum_{Y \in CG} min(AR^{local}(X,U), GQ \times TR(X,Y)) \tag{3}$$

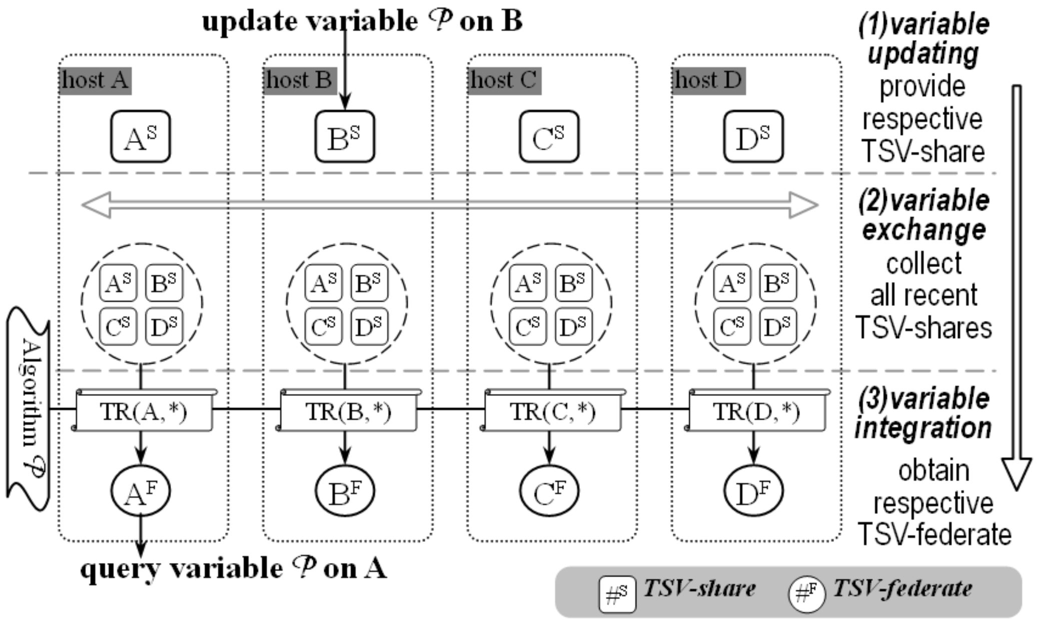

Fig. 2. Trust-based Security Variable

V. Job Submission Examples

The ATNAC policies are conditioned on the sensitivity of the required resource and the current SL for each requester. The sensitivity of the request depends on the sensitivity of the requested resource, the type of job to be executed, and other temporal and environmental constraints.

To demonstrate the ATNAC architecture, we present two job submission examples. In both examples, users access a web page to submit jobs to be executed on distributed grid resources, and collect the results.

Consider a grid environment in which grid resources belong to two research labs L and M. The resources include expensive online measurement instruments that can be accessed only by users with the qualification Q. The data from some projects are confidential and must be protected. The data is stored in the database D and protected by a non disclosure agreement signed by L and M. The information, for example, may contain computationally expensive unpublished results of a novel experiment conducted by a group of experts E (consisting of scientists who work for L and M) that could be exploited by a competing party. Even knowledge about the existence of the group of experts E working on the project could be considered sensitive, and the agreement itself may leak sensitive information and must be protected.

Before the server accepts a job or grants access to grid resources, the server requires the user to present various proofs. Figure 3 shows an access control policy with dynamic requirements evaluated by the GAA-API. The S_J thresholds shown are arbitrarily divided into three levels based on a linear interpretation of the SL and convenience of explanation. Figure 4 illustrates a high level view of server and client proofs with their trust negotiation policies used in these examples.

A. Example 1–GAA-API Access Control Policy Evaluation

In this example, we concentrate on the access control policy conditioned on the job characteristics and suspicion level S_J for a given user:

1) **Job characteristics**: In this example, jobs that require moderate resources (i.e., a moderate $EC_M(J)$ value) and no access to sensitive assets are considered **non sensitive**, jobs that require access to an expensive on-line instrument are labeled as **sensitive**, and jobs that require access to sensitive information (e.g., results of the experiment) are classified as **very sensitive**.
2) **Suspicion level**: For the current user, S_J is determined by the Analyzer as described in section III-A. If a user has established a good record in his/her past transactions, a lower suspicion level is assigned. However, if a user's record is bad, the S_J component of the SL will be higher.

According to Figure 3, when the suspicion level is *low* ($0 \leq S_J < 0.33$), the server allows a user to submit non sensitive jobs. As transaction sensitivity increases, the server selects stricter access control policy requirements. To submit a

Required Credentials		**Job Characteristics**		
		non-sensitive	sensitive	very sensitive
Suspicion Level	**low** < 0.33	accept	Q	E
	mid[0.33,0.66]	$P_L \vert P_M$	$(P_L \vert P_M)\&Q$	$E\&P_M$
	high > 0.66	P_M	reject	reject

Fig. 3. An Access Control Policy

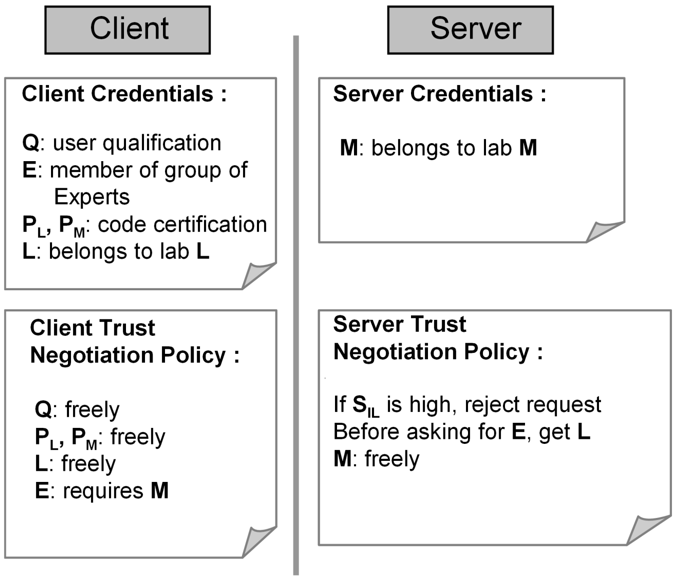
Client
Server
Client Credentials :
Q: user qualification
E: member of group of Experts
P_L, P_M: code certification
L: belongs to lab L
Server Credentials :
M: belongs to lab M
Client Trust Negotiation Policy :
Q: freely
P_L, P_M: freely
L: freely
E: requires M
Server Trust Negotiation Policy :
If S_IL is high, reject request
Before asking for E, get L
M: freely

Fig. 4. Credentials and Trust Negotiation Policies

sensitive job, the user must present credential Q proving that the user has required qualification to access the instrument. To run a very sensitive job, the user must be a member of a group of experts E that conducts a sensitive experiment and has exclusive access the database.

When S_J for a user is *medium* ($0.33 \leq S_J \leq 0.66$), the server asks for the following credentials depending on the job sensitivity:

1) If the job is non sensitive, require a code certification P_L or P_M by L or M.
2) If the job is sensitive, the user must prove the code certification as described above *and* prove that the user has the required qualification Q.
3) If the job is very sensitive, require code certification by P_M and prove that the user belongs to the group of experts E. This policy is sensitive and is protected by TrustBuilder.

Assume that a user wants to submit a non sensitive job and the Analyzer associates the requester with an S_J of 0.8. When the GAA-API receives the job submission request, it consults the access control policy and determines that in order to grant this request the user has to provide a proof of code certification by P_M. Since this policy is not sensitive, the GAA-API asks the user for the credential directly (without calls to TrustBuilder). If the user provides the requested credential, access is granted, otherwise the request is denied.

B. Example 2–Negotiation for Sensitive Credentials

When the GAA-API determines that its access control policy requirements are sensitive, trust negotiation is used to obtain the required proofs after establishing a certain level of trust. In this example, assume a user with a low S_J needs to execute a job (classified as *very sensitive*) requiring access to sensitive data stored in database D. According to the access control policy derived from Figure 3, to grant the request, the user must prove that she belongs to the group E. The GAA-API invokes TrustBuilder to initiate a trust negotiation.

TrustBuilder enforces strict credential release policies by making sure that the requester is involved with the experiment but without revealing the existence of the agreement or the experiment to a curious requester. Therefore, TrustBuilder must avoid directly requesting credentials that allow the requester to infer that the experiment is in progress. This is enforced by TrustBuilder policies protecting sensitive credential requests as illustrated in Figure 4. Suppose that the user belongs to L. According to the server trust negotiation policy, a request for sensitive credential E first requires proof that she belongs to a lab L which is trusted.

The TrustBuilder server begins a negotiation by sending a request to the TrustBuilder client asking for credential L.

The client evaluates the trust negotiation policy and concludes that it can release L and sends it to the server.

The server can now release the sensitive policy, and asks for E.

In order to release E, the client needs to make sure that the server is a member of lab M, so the client asks for M.

The server sends M and the client responds with E.

At this point the negotiation succeeds, access control policies are satisfied and the request to submit the job is granted.

If the user is not a part of the experiment and tries to obtain sensitive info by sending irrelevant credentials or sending policies that require the server to reveal sensitive information, the suspicion level S_{IL} that is attributed to the sensitive information leak will be increased by the Analyzer. If S_{IL} becomes too high, the server rejects any further requests from that user.

VI. Related Work

Some of the trust negotiation systems and languages developed in recent years include PeerTrust, SD3, RT, Cassandra, and χ-TNL.

PeerTrust [12] is a trust management system that uses a simple and expressive policy language based on distributed logic programs. PeerTrust agents perform automated trust negotiation to obtain access to sensitive resources. The underlying trust negotiation methods are similar to those explained in this paper. PeerTrust policies are evaluated by a Datalog-based logic engine that allows delegation of authority, signed rules, expression of complex conditions, and sensitive policies. Previous work on PeerTrust looked at incorporating it into Grid environments [13].

The Secure Dynamically Distributed Datalog (SD3) trust management system [14] is closely related to PeerTrust. It has a policy language that allows users to specify high level security policies. SD3 handles policy evaluation and certificate verification. Since the policy language in SD3 is an extension of Datalog, security policies are a set of assumptions and inference rules.

Li et al. [15] introduce RT, a family of Role-based Trust management languages for representing credential and policies in distributed authorizations.

Cassandra [16], a rule-based policy specification language and evaluation engine, combines features of RT [8] and is also related to PeerTrust [12]. Based on Datalog$_C$, Cassandra provides a complete trust management framework with primitives for performing actions (is X permitted to view resource Y), activating and deactivating roles on services (entity A may change role R on a specific service), and requesting credentials. Cassandra contains the primitives necessary for bilateral credential exchange and supports distributed trust negotiation. The Cassandra system has been implemented in a case study for managing access control to Electron Health Records in England's National Health Service data-spine.

Bonatti and Samarati [17] proposed a policy-based framework and an interaction model for regulating access to network services. This trust establishment framework uses logical rules for accessing services and avoiding unnecessary disclosure of sensitive information. Each party maintains either interaction-specific or persistent state (e.g., credentials and provided services).

Bertino et al. [18] present χ-TNL, an XML-based language for conducting trust negotiation. χ-TNL provides a medium to transport information about the negotiating parties called a certificate. A certificate can be either a credential or a declaration. A credential is a list of properties of a negotiating party certified by a Certificate Authority. A declaration contains helpful information (e.g., policies) for the negotiation process.

The ATNAC framework presented in this paper complements previous work on trust negotiation and is the first example of a system that supports both fine-grained adaptive access control and adaptive trust negotiation. The novelty of this work lies in the cooperation between the access control and TN systems that allows ATNAC to support the dynamic adjustment of security policies based on a particular component of the suspicion level. The framework was first presented in [10]. The contribution of this paper is to introduce the framework to the grid community. The focus of the earlier work was on DDoS attacks against trust negotiation in an e-Business setting. In this paper, we introduced a new dimension to the suspicion level: user/job behavior characteristics, and provided examples from the Grid community to illustrate the usability of the framework.

VII. Conclusions and Future Work

The ATNAC framework described in this paper brings adaptive trust negotiation and access-control security services to the Grid. ATNAC allows us to:

- Establish trust between service requesters and providers not in the same security domain;
- Improve scalability by avoiding pre-registration and maintaining local accounts;
- Protect sensitive information including credentials and policies. Service providers might be cautions about presenting certain credentials and might not want to divulge which credentials are needed until they have confidence in a requester.
- Respond to inappropriate behavior.

Our framework integrates TrustBuilder, an implementation of trust negotiation, with the GAA-API, an adaptive access control API. The GAA-API/TrustBuilder implementation is available at **`http://gaaapi.sysproject.info`**.

Several areas of future work include the analysis of our method for calculating suspicion levels, tightening the integration of the trust negotiation and access control systems, and identifying alternate methods for detecting release of sensitive information. We also plan to analyze the current framework and define abstract interfaces for the communication between GAA-API and TrustBuilder. A well-defined interface will allow integration with other trust negotiation systems.

We are working on integrating the GAA-API into GridSite [19], a Grid-aware web application that can operate as a host for grid services. GridSite provides a library for manipulating grid credentials (conventional X.509 certificates and Globus GSI proxies), and XML Grid Access Control Lists (GACL). We are interested in exploring how the GAA-API can interpret policies expressed in GACL and the benefits of such integration.

We are applying ATNAC techniques to negotiation of agreements in virtual organization and P2P environments, rather than using them solely to negotiate proofs of security attributes. Ultimately we intend to build a web based interface that allows two parties to negotiate an agreement in real time. Most of the interactions will be done automatically. An input from a user will be required only to resolve any conflicts.

Acknowledgment

This research was supported by funding from the National Science Foundation under grants no. CCR-0325951 and ACI-0325409, as well as prime cooperative agreement no. IIS-0331707, and The Regents of the University of California. The views and conclusions contained herein are those of the authors and should not be interpreted as representing the official policies or endorsement of the funding agencies. Figures and descriptions were provided by the authors and are used with permission.

References

[1] I. Foster and C. Kesselman, *The Grid 2: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 2004.

[2] W. H. Winsborough, K. E. Seamons, and V. E. Jones, "Automated trust negotiation," in *Proc. 1st DARPA Info. Survivability Conf. and Exposition (DISCEX I)*, Hilton Head, SC, Jan. 2000, pp. 88–102.

[3] T. Ryutov and C. Neuman, "The specification and enforcement of advanced security policies," in *Proc. of the Conf. on Policies for Distributed Systems and Networks*, Monterey, CA, June 2002, pp. 128–139.

[4] T. Ryutov, C. Neuman, and D. Kim, "Dynamic authorization and intrusion response in distributed systems," in *Proc. 3rd DARPA Info. Survivability Conf. and Exposition (DISCEX III)*, Washington DC, April 2003, pp. 50–61.

[5] T. Ryutov, C. Neuman, D. Kim, and L. Zhou, "Integrated access control and intrusion detection for web servers," *IEEE Trans. Parallel and Distributed Systems*, vol. 14, no. 9, pp. 841–850, September 2003.

[6] M. Winslett, T. Yu, K. E. Seamons, A. Hess, J. Jacobson, R. Jarvis, B. Smith, and L. Yu, "Negotiating trust on the web," *IEEE Internet Computing*, vol. 6, no. 6, pp. 30–37, 2002.

[7] M. Blaze, J. Feigenbaum, and J. Lacy, "Decentralized trust management," in *Proc. IEEE Symposium on Security and Privacy (SP'96)*, Oakland, CA, May 1996, pp. 96–17.

[8] A. J. sang, "The right type of trust for distributed systems," in *ACM New Security Paradigms Workshop*, Lake Arrowhead, CA, 1996, pp. 119–131.

[9] A. Herzberg, Y. Mass, J. Mihaeli, D. Naor, and Y. Ravid, "Access control meets public key infrastructure, or: Assigning roles to strangers," in *Proc. IEEE Symposium on Security and Privacy (SP'00)*, Oakland, CA, May 2000, pp. 2–14.

[10] T. Ryutov, L. Zhou, C. Neuman, T. Leithead, and K. Seamons, "Adaptive trust negotiation and access control," in *10th Symposium on Access Control Models and Technologies (SACMAT'05)*, Stockholm, Sweden, June 2005, pp. 139–146.

[11] *Resource Specification Language*. [Online]. Available: http://www-fp.globus.org/gram/rsl_spec1.html

[12] W. Nejdl, D. Olmedilla, and M. Winslett, "Peertrust: Automated trust negotiation for peers on the semantic web," in *Proc. of the Workshop on Secure Data Management in a Connected World (SDM'04)*, Toronto, Canada, August/September 2004.

[13] J. Basney, W. Nejdl, D. Olmedilla, V. Welch, and M. Winslett, "Negotiating trust on the grid," in *Proc. 2nd Workshop on Semantics in P2P and Grid Computing*, New York, NY, May 2004.

[14] T. Jim, "Sd3: A trust management system with certified evaluation," in *Proc. IEEE Symposium on Security and Privacy (SP'01)*, Oakland, CA, May 2001, pp. 106–115.

[15] N. Li, J. C. Mitchell, and W. H. Winsborough, "Design of a role-based trust management framework," in *IEEE Symposium on Security and Privacy*, Oakland, CA, May 2002, pp. 114–130.

[16] M. Y. Becker and P. Sewell, "Cassandra: Distributed access control policies with tunable expressiveness," in *Proc. IEEE 5th International Workshop on Policies for Distributed Systems and Networks*, Yorktown Heights, NY, June 2004, pp. 159–168.

[17] P. Bonatti and P. Samarati, "A unified framework for regulating access and information release on the web," *Computer Security*, vol. 10, no. 3, pp. 241–271, 2002.

[18] E. Bertino, E. Ferrari, and A. Squicciarini, "χ-TNL: An XML-based language for trust negotiation," in *Fourth IEEE International Workshop on Policies for Distributed Systems and Networks*, Como, Italy, June 2003, pp. 81–84.

[19] *GridSite*. [Online]. Available: http://www.gridpp.ac.uk/gridsite/