Adaptive Trust Negotiation and Access Control

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

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

Abstract

Electronic transactions regularly occur between business partners in separate security domains. Trust negotiation provides an open authentication and access-control environment for such transactions, but it suffers from malicious attacks leading to denial of service or leakage of sensitive information. In this paper we introduce adaptive trust negotiation and access control as a means to counter such attacks.

1. Introduction and Motivation

Electronic business transactions often take place between entities that are strangers to one another. This presents a challenge for establishing trust between service requesters and providers not in the same security domain. Buyers must trust that sellers will provide the services they advertise and will not disclose private buyer information. Sellers must trust that the buyers will pay for services or are not underage for accessing services or purchasing certain goods.

In these situations, problems of trust are made more complex because participants conduct business transactions that extend beyond their local security domain. Since on-line communications provide a nearly anonymous medium, people are more inclined to cheat than in face-to-face interactions. The participants can conduct fraudulent and malicious actions in order to damage a competitor and steal money or sensitive information.

Our solution to these problems is based on two well established systems. The GAA-API provides adaptive access control that captures dynamically changing system security requirements. The TrustBuilder 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 electronic business services. This combination extends the capabilities of each system. In particular, the GAA-API/TrustBuilder integration allows us to:

- detect and thwart certain attacks on electronic business transactions;
- support cost effective trust negotiation, such that TrustBuilder is invoked only when negotiation is required by access control policies;
- adapt information disclosure and resource access policies according to a level of suspicion.

In this paper, we demonstrate how our system adapts the security policies according to sensitivity of access request and a suspicion level associated with the requester. When sensitivity of the request is low, the server allows a customer to purchase goods with weaker proofs of payment (e.g., one-time credit card). This provides better protection for the customer, but increases the financial risk of the online store. When transaction sensitivity becomes higher (e.g., a customer attempts to make a large purchase), the server requires more trust-worthy credentials, such as a valid user identity or user credit rating. This provides a better protection for the server and guards against potential attackers by requiring more proofs before granting the request.

The GAA-API/TrustBuilder framework detects malicious user activity by examining failure types and behavior patterns of the access control and trust negotiation. If any suspicious event is identified (e.g., a user is repeatedly presenting forged credit card credentials with incorrect card expiration dates), security policies adapt in real time and dynamically protect the system from potential threats. For instance, the system can react by denying the access, increasing the suspicion level for the misbehaving user and banning the user.

Traditional on-line security measures involve identity-based access controls that assume that service providers and requesters are known to each other. Publicly available access control policies specify which users have access to what resources. Users are typically required to pre-register with a service provider before requesting a service. This can often discourage on-line commerce.

Parties without pre-existing relationships cannot perform sensitive transactions based on identity. Thus, to conduct business there must be a means of establishing a sufficient level of mutual trust. Since neither party is known to the other, both parties may have sensitive information that they are reluctant to reveal until the other party is authenticated at a certain level. Therefore, secure electronic business transactions must support access control policies that regulate not only the granting of resources, but also the disclosure of sensitive user information to strangers.

Trust negotiation (TN) is a recent approach to access control and authentication that enables resource requesters and providers in open systems to establish trust based on attributes other than identity. One approach to trust negotiation is to establish trust through the gradual, iterative, and mutual disclosure of credentials and access control policies. Trust negotiation, used in conjunction with a system that supports adaptive fine-grained access control, provides a framework for protecting a system against attacks on electronic business transactions.

The focus of this paper is to support adaptive, fine-grained access control and trust negotiation in the presence of malicious service requesters. Malicious requests may come in the form of denial of service (DoS) attacks against trust negotiation using its underlying protocols or attempts to obtain sensitive information.

In this paper, we address these issues by introducing *adaptive trust negotiation*. The novelty of this work lies in the TN and access-control framework where authentication and access-control are guided by static constraints such as sensitivity of the request, time and location, and also dynamic constraints such as negotiation context, the service provider's past experience and perceived system treat level. We introduce the notion of a general suspicion level that depends on these dynamic constraints. The techniques outlined in this paper may be used to protect requesters from malicious service providers; however, this paper focuses on protecting the server provider—as such, our examples exhibit some degree of asymmetry in the trust placed on the participating parties.

In section 2, we present trust negotiation and the attacks that motivate the need for adaptive trust negotiation. Section 3 presents the GAA-API, a generic access-control API. Section 4 outlines how TrustBuilder and the GAA-API work together to provide the solutions previously mentioned. The implementation of our architecture is set forth in section 5. Section 6 contains related work, and section 7 summarizes our conclusions and directions for future work.

2. Adaptive trust negotiation

With public key cryptography, the use of unforgeable digital attribute credentials signed by trusted issuers allows users in different security domains to exchange credentials containing attributes about themselves in an effort to establish mutual trust. Trust is based on a user's attributes rather than identity, removing the necessity for pre-registration to gain access to a system. 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 access policy for the desired resource.

A simple example of trust negotiation takes place between a client that wishes to make a purchase on a trust negotiation-enabled server. The client initiates a trust negotiation by indicating his intention to make a purchase (an action requiring the server to trust that the client has the capability to pay). Thus, the server will reply with an access-control policy, a policy specifying the required attributes necessary for the server to trust the client. Such a policy might require attributes like an online payment service account (e.g., PayPal), temporary or permanent credit card numbers. Because these

policies specify attributes, any user possessing the required attributes[1] can be authenticated. The client evaluates the policy using trust negotiation software, and discloses non-sensitive credentials or the policies governing the sensitive credentials such as the client's credit card credential. Using automated trust negotiation, both client and server mutually authenticate the other, a practice uncommon in current authentication systems. Finally, the server's access-control policy is satisfied and the purchase order is placed.

Trust negotiation has been explored by several groups of researchers over the past five years (e.g., [15], [14], [8], [2]). The previous research has primarily focused on the details of performing a single trust negotiation. In this paper, we take a step back and consider the behavior of a trust negotiation system in operation. Trust negotiation systems are vulnerable to Denial of Service (DoS) attacks, a threat that has never been previously explored, as well as sensitive information leaks. Both forms of attack pose a significant threat to a trust negotiation system in operation.

There are several ways that an adversary might launch a DoS attack against a trust negotiation system at the application layer. Attacks could result from initiating a large number of TN sessions with the server, disclosing a very complex policy that the server must evaluate, and disclosing many credentials to the server that are invalid or irrelevant to the negotiation. The net result of such attacks is that the server expends an excessive amount of computational resources on illegitimate requests and is hampered in its ability to service requests from legitimate clients.

Another type of attack that an adversary can launch against trust negotiation is to negotiate with the intent of collecting or inferring sensitive information instead of establishing trust to proceed with a transaction. To achieve this, an attacker might send policies that require the other party to reveal sensitive credentials.

TN systems must be adaptive to detect and thwart such attacks. They 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. Trust is not static but changes over time because of provider experience or past interactions with the requestor: multiple successful past interactions increase trust, whereas suspicious provider or requester behavior (e.g., a large number of failed negotiations) negatively influence perceived trust and increases suspicion.

Adaptive trust negotiation makes use of dynamic run-time system suspicion levels to adjust trust negotiation polices, limit the receivable number of credentials and policies, and tighten session timeout, minimum turnaround time, and round-count values. Suspicion levels are calculated based on static constraints known to the service provider (e.g., requester location and time) and dynamic constraints (e.g., requested resource, previous experience with this requester, probability of a DoS attack, system threat level), access-control and trust negotiation feedback. The system threat level is provided by Intrusion Detection Systems (IDS). For example, a low system threat level indicates normal system operation, medium level means a suspicion of attack, and high level means that the system is under attack.

Trust negotiation can be quite expensive in terms of computation, network bandwidth and number of rounds. Adaptive TN can reduce the overhead in certain circumstances by associating less restrictive policies (which require a fewer number of credentials) with lower suspicion levels. This will result in a higher probability of finding a mutually satisfactory policy and improving the success rate of negotiations. Fine-grained access control can also reduce the overhead by initiating the TN only when it is required by the access control policies. For example, a policy may allow access to a sensitive resource for any requesters connecting from a trusted domain, without evaluating credentials that are subject to trust negotiation. Requests originating from outside of the trusted domain require those credentials, and thus trigger the TN process to establish the level of trust required before the credentials are provided.

[1] Users must typically provide additional proof that they own the disclosed credential, and the credentials must be valid (e.g., contain the correct digital signature, valid expiration date, not revoked, etc.) as well as trusted by the other negotiating party.

3. Adaptive access control

The Generic Authorization and Access-control API (GAA-API) [10], [10], [11], and [13] is a middleware API for authorization and access-control. The API supports fine-grained access control and application level intrusion detection and response. The GAA-API allows dynamic adaptation to network threat conditions communicated by an Intrusion Detection Service (IDS). The GAA-API can also detect some intrusions by evaluating access requests and determining whether the requests are allowed and if they represent a threat according to a policy. This information is used in evaluation of conditions to activate and deactivate particular policy entries that are used to mediate access to application-level objects.

If response to an attack scenario must be fine-grained (i.e., short of shutting down all connections into a system completely, yet still comprehensive enough to protect against attacks that originate from multiple points on the network), then higher-level software must adapt behavior in response to a perceived threat. This support must be at the application level in servers because often only the servers themselves have knowledge of the operations that are requested, and the objects to be manipulated by the request. The GAA-API supports such adaptation by employing policy evaluation mechanism extended with the ability to generate real time actions, such as checking a current system threat level, generating audit records and updating firewall rules. However, the GAA-API neither supports trust negotiation nor protection of sensitive policies. Therefore, the GAA-API /TrustBuilder integration described in this paper leverages the best features of the two systems in order to support adaptive trust negotiation.

While the GAA-API can be used by a number of different applications[2] with no modifications to the API code, we focus on the Web server in this paper, because this environment is much more familiar to the reader and thus can best explain our work.

4. Framework Overview

To protect their resources, e-business providers maintain the following security entities:

- **Public resources, services and credentials** that are accessible to clients, e.g., database of products and VeriSign certified company credential.
- **Public security policies** that govern resources, services, operations and credentials and can be freely available to clients.
- **Sensitive resources** that can be accessed online only by a limited number of users, e.g., customer lists, marketing strategies, custom computer programs, manufacturing techniques, and business forms.
- **Sensitive services** such as company's on-line internal services: e-library, etc.
- **Sensitive credentials** such as the number of customer complains filed with the requester
- **Sensitive operations** (access rights). Some operations (even on public resources) can be restricted, e.g., database queries that leak information to competitors.
- **Sensitive security policies** may include:
    - *Access control policies* that govern sensitive resources, services, operations, and
    - *Trust negotiation policies* that specify which credentials to disclose at a given state of the trust negotiation, and the conditions to disclose them.

Because the requester security entities are just a subset of those used by the provider (requesters may not have service operations), we focused on the provider security entities discussed above.

[2] The GAA-API can provide enhanced security for applications with different security requirements. We have integrated the GAA-API with the Apache Web server, SOCKS5, sshd, and FreeS/WAN IPsec for Linux.

In our framework, adaptive trust negotiation and access control are managed by two separate software components: TrustBuilder and the GAA-API respectively. TrustBuilder, developed at the Internet Security Research Lab at Brigham Young University (see http://isrl.cs.byu.edu/), employs X.509v3 digital certificates and TPL policies [5] to describe attribute credentials and trust negotiation policies, respectively. Access control policies that govern public and sensitive resources, services, operations, and policies are expressed in EACL format [12] enforced by the GAA-API. Trust negotiation policies that regulate the disclosure of sensitive credentials are enforced by TrustBuilder. Figure1 introduces our framework.

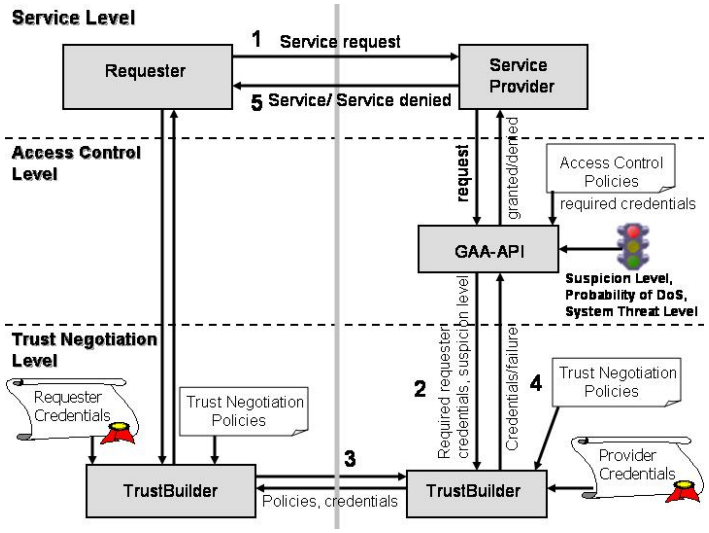

Figure 1: Overview of TrustBuilder/GAA-API integrated framework

In e-business applications, requesters send service requests to the service provider. The service provider forwards the requests to the GAA-API. The API first evaluates the access control policy that governs access to the requested service and either accepts or rejects the request according to the policy. The API employs context information (e.g., system threat level reported by an IDS, probability of a DoS attack and prior experience for this the requester) to come up with access control decision **(1).**

During policy processing, the GAA-API may initiate an interaction with TrustBuilder in order to establish sufficient trust with an entity outside of its known security domain. The Analyzer (a GAA-API module) calculates the suspicion level for particular requestor/connection (described in Section 5.1). The level is based on a number of parameters such as sensitivity of a resource, service or operation, system threat level reported by an IDS, location of the requestor (e.g., international buyers may be less trusted), etc. Next the GAA-API passes the suspicion level and a list of required credentials (specified in the access control policy) to TrustBuilder **(2).**

TrustBuilder attempts to obtain the requested client credentials by engaging in a trust negotiation and regulating its trust negotiation policies based on the current suspicion level **(3).**

Upon TN completion, TrustBuilder reports the result to the GAA-API. If the negotiation succeeds, the required credentials are passed to the GAA-API **(4).**

The GAA-API uses the TN results to evaluate the access control policy and either grants or denies the requested service. Feedback from TrustBuilder is also used to affect the current suspicion level **(5).**

Our system can guard against some DoS attacks by employing external limits on the time each party is willing to devote to a negotiation for a particular kind of sensitive resource. At the application level the system can detect unusually large number of initiated trust negotiations and/or failed negotiations from the same requester and either deny access or increase the suspicion level. Increasing the suspicion level tightens the access control and credential release policies.

5. GAA-API/TrustBuilder Implementation

In order to demonstrate the adaptive access control and trust negotiation model described in this paper, we present an architecture targeted specifically at e-commerce environments. We integrated the GAA-API into an Apache web server [12], whose original access control (*mod_access***)** and authentication (*mod_auth***)** modules are replaced by calls to the GAA-API. The GAA-API is integrated into Apache by modifying the *check_dir_access* function. The "glue" code extracts the information about requests from the Apache core modules, initializes the GAA-API, calls the API functions to evaluate policies and, finally, returns access control decision and status values to the modules.

TrustBuilder runs on server and clients. We considered two ways to integrate TrustBuilder into the client system: browser plug-in and proxy server. While the plug-in approach is more straightforward, it required specific modification of the client browser. By using an HTTP proxy running on the client host or gateway machine, the web browser remained untouched. We chose to launch TrustBuilder from the client proxy in order to support standard web browsers such as Internet Explorer, Mozilla, or Opera. Therefore, modifications were only necessary on the client proxy.

In the GAA-API/TrustBuilder architecture, a typical session involving trust negotiation consists of two phases:
1. The server asks the client to present particular credentials as defined by the access control policy.
2. A trust negotiation is performed to obtain and verify all necessary credentials.
Figure 2 shows the detailed interactions in a GAA-API/TrustBuilder session. A user generates a request on the web browser, and the request is passed to the web server via a client proxy **(1)**. When the web server receives the request, it

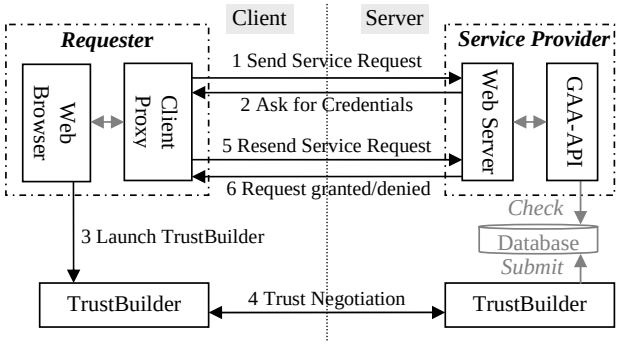

Figure 2: GAA-API/TrustBuilder Implementation for Apache Web Server

asks the GAA-API to evaluate its access control policies and decide whether this request should be granted, rejected or reevaluated with further proofs (username/password, credentials, etc.). If the credentials required by the policies are currently unavailable (or invalid) in the server database, the GAA-API will ask the client for these credentials via TrustBuilder **(2).**

In the second phase, the client proxy receives an HTTP response requiring further authentication via TrustBuilder and the required credentials, and launches the TrustBuilder client **(3)** to initiate a trust negotiation with the TrustBuilder server **(4)**. The server enforces a proper trust negotiation policy, guided by the system suspicion level. If the TrustBuilder server's policy is satisfied and all the client's credentials are verified successfully (e.g., the required

attributes are contained in the credentials, the certificates are not expired, etc.), then the verified credentials will be logged into the database.

On the client side, when the trust negotiation is successfully completed, the proxy resends the original request to the web server **(5).** The GAA-API will reevaluate its access control policies, and if all the necessary credentials are available in the database, then the service request for this session will be granted; otherwise, it will be rejected **(6).**

5.1 Adaptive Security Policies

E-business is under an increasing number of attacks that are becoming more sophisticated and destructive. Attackers exploit system vulnerabilities to gain unauthorized access, which results in monetary loss, theft of proprietary information and negative impact on a company's reputation. Therefore, it is essential to have tightly integrated security mechanisms that can identify ongoing or possible attacks and dynamically adapt security policies to protect the system.

The GAA-API/TrustBuilder architecture provides policy adaptation at the access control and trust negotiation levels to detect and respond to security attacks (Figure 3).

At the access control level, the GAA-API evaluates access control policies to determine which credentials are necessary to satisfy the access request. The types of the required credentials depend on the sensitivity of the request, system threat level, the suspicion level and the DoS probability for the particular requester. The GAA-API maintains this information in the *Session History Database*. This information is stored in the database using a randomly generated *SessionID* as an index key. The GAA-API fist tries to fetch the required credentials from the database **(1)**. If the necessary credentials are found, the credentials are still valid (did not expire) and the system threat level is low, the API uses the fetched credentials to evaluate the policy.
Otherwise, the API submits the request for required credentials and the session's suspicion level to the database **(2).**

At the trust negotiation level, the TrustBuilder server uses the *SessionID* to extract the required credentials and suspicion level for this session from the database **(3)**. Next, TrustBuilder dynamically chooses the trust negotiation policy, according to the session suspicion level and the credentials requested by the TrustBuilder client. After the trust negotiation completes, TrustBuilder logs the results (e.g., obtained credentials, failure modes) to the *Session History Database* **(4).**

Finally, the Analyzer checks the session history in order to identify active attacks or potential threats and calculates the suspicion level and DoS threat level for each user. 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 **(6).**

If any suspicious event is identified (e.g., a user is repeatedly presenting forged credit card credentials with incorrect card expiration dates), security policies adapt in real time and dynamically protect the system from potential threats. For instance, an invalid credential may be linked to an attempt at impersonation or use of an unauthorized credit card account. We implemented a simple and straightforward strategy preventing such attacks. The next example explains our algorithm. 1) By default, the suspicion level of a user is set to *medium*. 2) If a user successfully completes its previous trust negotiation, the suspicion level will be adjusted to *low*. 3) If a user has experienced one or two consecutive trust negotiation failures (i.e., failure to present the required credentials or certificate integrity errors), the suspicion level will be set back to *medium.* 4) If a user has consecutively experienced trust negotiation failures (i.e., two or more times), the suspicion level will be raised to *high*.

5.2 Detection of denial of service attacks

Another benefit of the GAA-API/TrustBuilder architecture is protecting the system from potential denial of service attacks at the application level. In some cases, an attacker can use a simple request to invoke expensive operations on the server that involve, for example, public key cryptographic operations. Thus, a large number of such requests can greatly degrade the server performance or even disable the server. The TrustBuilder server is especially vulnerable to such attacks, because the trust negotiation process is very expensive. To identify and control potential DoS threats, we apply several adaptive policies under the GAA-API/TrustBuilder architecture. Before starting a trust negotiation, the TrustBuilder server first verifies that the GAA-API asked the client for credentials. Otherwise, the trust negotiation request is irrelevant to the services to be authorized; we should regard it as an error or even a DoS suspicion. With this regulation imposed, the attacker has to compromise the GAA-API before it can gain any access to the more vulnerable TrustBuilder server. Otherwise, the attacker can easily exhaust the server resources by launching excessive trust negotiations.

During a trust negotiation process, the credentials sent by the client must conform to the access control policies for this session; Otherwise, this trust negotiation process aborts immediately. This regulation prevents attackers from sending a great number of irrelevant credentials for the TrustBuilder server to verify, which can degrade the server performance significantly. To counteract such DoS attacks, once the TrustBuilder server identifies an abnormal number of client credentials within one trust negotiation, it generates an alert, which bans this user immediately.

When a client does not respond to a trust negotiation conversation, the server resources allocated for this negotiation persist until a timeout failure occurs. So, if attackers open a great number of negotiations without giving further responses, it can exhaust the server resources. However, such timeout failure may also be a result of accidental network partition. From a single timeout failure it is impossible to distinguish a DoS attack from a network failure. The Analyzer is used to identify that the user has generated similar requests in an abnormally high frequency. The user suspicion level would increase (slowing down his ability to make rapid requests) or the user would be banned outright. To mitigate the server's burden under such attacks, the GAA-API will adjust the server timeout to a smaller value, or completely ban this user from any access.

In summary, the GAA-API/TrustBuilder framework detects malicious activity by examining failure types and behavior patterns of the trust negotiation. Credentials with improper attributes may refer to misuses of a user's identity. Repeated failures in identity verification may lead to attempts of impersonation, and consecutive timeouts may be the result of a DoS attack.

5.3 On-line store example

To demonstrate the adaptive GAA-API/TrustBuilder architecture, we present an online purchasing example. An online-store is selling goods to the public. Customers can access the store web page and submit purchase requests that specify the goods they want to buy. Before the store server grants a purchase order, it requires the customer to present various proofs of payment. Figure 4 illustrates a simplified access control policy, server and client credentials, and server and client trust negotiation policies for this example. The access control policy depends on the sensitivity of each request based on two factors:

1. *Purchase amount*. Larger purchase amounts mean higher transaction sensitivity.
2. *Suspicion level for current user*, which is drawn from the history for the user. 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 transaction sensitivity increases.

When sensitivity of the request is low, the server allows a customer to purchase with a one-time credit card or PayPal service. This provides better protection for the customer, but increases the financial risk of the online store. When

transaction sensitivity becomes higher, the server requires more trust-worthy credentials, such as a permanent credit card credential, valid user identity, or user credit rating.

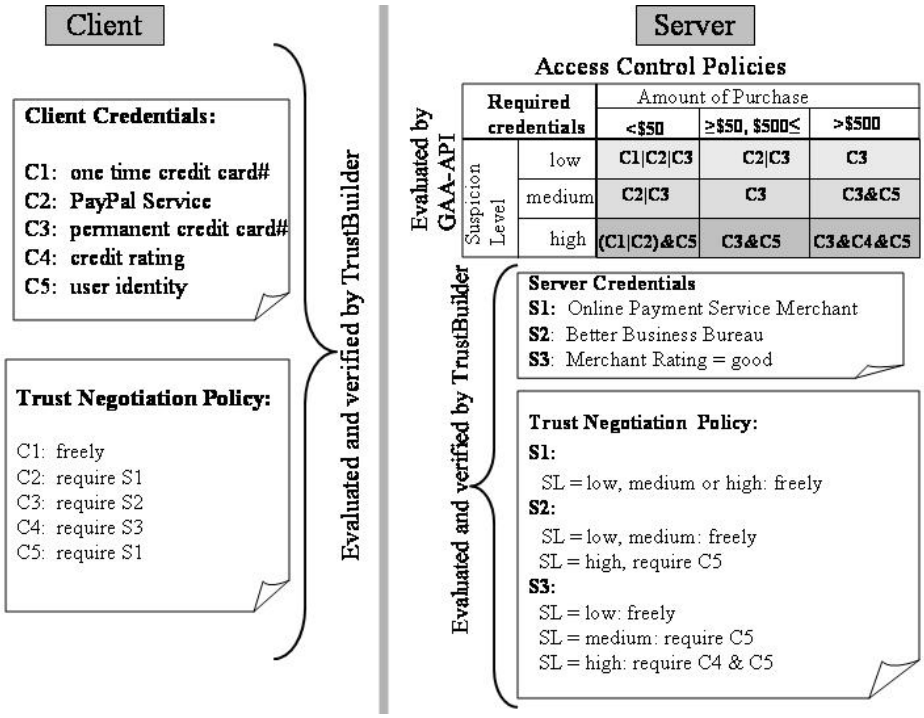

Figure 4: Policies and credentials for the online-purchasing example

The strategy described above is defined by an access control policy. The table in Figure 4 shows that the required credentials depend on the purchase amount and the suspicion level. For example, when the suspicion level for a customer is *medium*, the server asks for the following credentials depending on the purchase amount of current transaction:

- If the purchase amount is less than $50, require a PayPal payment or permanent credit card number credential.
- If the purchase amount is between $50 and $500, permanent credit card number is required.
- If the purchase amount is greater than $500, require the permanent credit card number and the user identity.

TrustBuilder is responsible for obtaining and verifying the required credentials. Based on the credential requirements and the suspicion level, the TrustBuilder server adapts the trust negotiation policies, exchanges the credentials, and builds trust with the TrustBuilder client. For example, assume that a requester wants to make a small purchase (<$50). Assume that the Analyzer associates the requester with a suspicion level (SL) *medium*. When the GAA-API receives the purchase request, it consults the access control policy and determines that in order to grant this request, the client has to provide a permanent credit card or PayPal credential (i.e., C2 | C3 in Figure 4). The GAA-API invokes TrustBuilder and passes it the suspicion level and the request for the credentials.

Next the TrustBuilder server sends a request to the TrustBuilder client stating that the client must present a permanent credit card (C3) or PayPal credential (C2). The client evaluates trust negotiation policy and determines that in order to release C3, he must have proof that the online store belongs to the Better Business Bureau (S2), and in order to release C2 he must be sure that the merchant will accept his PayPal account credential (S1).

The TrustBuilder client sends this trust negotiation policy back to the server. The TrustBuilder server determines that the PayPal merchant credential (S1) can be given out to anyone, independent of the current suspicion level. The Better Business Bureau credential (S2) may also be given out freely, but only to those clients who have a suspicion level less than *high*. Both credentials are then returned to the client to satisfy the trust negotiation policy for release of the PayPal and permanent credit card credentials. Since the server required one or the other (or both) then it doesn't matter which credential the client releases to the server because either one would grant access to the resource. When the client supplies the necessary valid credential, and the credential is successfully verified at the server, the purchase request is granted. If the client tries to cheat and sends a fake credit card credential that is not verified by the TrustBuilder server, the request will be denied and the suspicion level will be elevated to *high* and more restrictive policies will be applied. Similarly, if the purchase amount were higher, then the policy would be also stricter (see Figure 4 with suspicion level at *medium* and compare the small purchase policy vs. the large purchase policy).

Therefore, with trust negotiation feedback, security policies adapt in real time and guard against potential attackers by requiring more proofs before they can succeed in compromising the system.

6. Related Work

There are a number of research efforts underway to explore various aspects of trust negotiation. In particular, the development of logic-based languages that are suitable for trust negotiation has been an important thrust. In this section, we briefly describe some of these efforts.

Bonatti and Samarati [3] proposed a framework based on a policy language 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).

RT [8] is a role-based trust management language and runtime engine that maps entities to roles based on their properties shown in digital credentials. The system is able to locate and retrieve credentials that are not available locally. It also provides support for preventing the leakage of sensitive credential information based on the behavior of the negotiation participants.

SD3 [6] is a trust management system consisting of a high-level policy language, a certificate retrieval mechanism, and a local policy compliance checker. The evaluation engine determines the answer to a query and also computes a proof that the answer follows from the security policy.

Cassandra [1] is a rule-based policy specification language and evaluation engine, combines. 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. Cassandra has been implemented in a case study for managing access-control to Electronic Health Records in England's National Health Service data-spine.

PeerTrust [9] 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.

Anonymous credential systems attempt to prevent the disclosure of private user information contained in certificates and the linkability of that information when certificates are presented to different entities. Idemix [4] (identity mixer), is an anonymous credential system or pseudonym system providing users the ability to prove facts about themselves (attributes) anonymously. Such anonymity is desirable in trust negotiation and access-control systems when sensitive credentials may disclose private user information to an untrusted service provider.

The work presented in this paper complements the 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.

7. Conclusions and Future Work

In this paper, we have presented an adaptive security framework for protecting sensitive resources in electronic commerce. This framework combines adaptive trust negotiation and fine-grained access control. Trust negotiation is a new paradigm for access control and authentication, which builds trust by accumulating attributes contained in credentials. Adaptive TN responds to requester sensitivity through the suspicion level by dynamically adjusting its policies, message-accept window, round count, and by providing feedback in the form of alerts when policies are not satisfied, credential verifications fail, or proof of credential ownership is invalid. Adaptive access control monitors requester activity and calculates suspicion levels based on feedback from the trust negotiation software. The access control policies are updated to reflect changes in suspicion level and only invoke trust negotiation as needed. 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**.

Using our framework, we are able to support the dynamic adjustment of security policies based on suspicion level and general system threat level. The system can detect and protect against some DoS attacks on the service provider. We also guard against sensitive information leak using dynamic policies both in the GAA-API and TrustBuilder. Finally, we support cost-effective trust negotiation by only invoking trust negotiation as directed by the access control policies. Our framework brings adaptive trust negotiation and access-control security services to e-commerce in open systems, eliminating the need for identity registration.

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. The current implementation of adaptive trust negotiation uses a simple algorithm for calculating the Suspicion and DoS threat levels. In the future, we will consider the application of anomaly detection techniques that determine normal user activity and then flag all behavior that falls outside the scope of normal as anomalous. 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.

Determining exactly when sensitive information is being disclosed is a difficult task. One method was presented in the paper: tightening the policy requirements based on a suspicion level and probability of a DoS attack. Another method for detecting release of sensitive information is to maintain a specific context for each trust negotiation. By observing the context surrounding each credential disclosure, irrelevant or sensitive credential releases may be detected, and feedback provided. Many other possible detection techniques are possible including statistical learning and ontology methods.

While this work was specifically targeted as a useful system for electronic commerce applications, it will also be useful in other contexts. Typically, access control in computational grids is accomplished by a combination of identity certificates and local accounts. Resource providers have to maintain a local account for every potential user. This approach does not scale well as the numbers of potential users and resources increase. Users will also need a paradigm

that is more flexible and is able to gradually establish trust, so that they do not have to present their most sensitive certificates to another party before first establishing a minimum level of trust in the other party.

Acknowledgments: The funding support of this work by the NSF ITR Grant ACI-0325409 is appreciated. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the NFS. Figures and descriptions are provided by the authors and are used with permission.

References

[1] Moritz Y. Becker and Peter Sewell, Cassandra: distributed access control policies with tunable expressiveness, In Policies in Distributed Systems and Networks, June 2004.

[2] Elisa Bertino, Elena Ferrari, and Anna Cinzia Squicciarini, Trust-X: A Peer-to-Peer Framework for Trust Establishment, In IEEE Transactions on Knowledge and Data Engineering, Vol. 16, No. 7, July 2004.

[3] Piero Bonatti and Pierangela Samarati, A Unified Framework for Regulating Access and Information Release on the Web, In Journal of Computer Security, pages 241-271, Vol. 10, Issue 3, 2002.

[4] Jan Camenisch and Els Van Herreweghen, Design and Implementation of the Idemix Anonymous Credential System, Research Report RZ 3419, IBM Research Division, June 2002.

[5] Amir Herzberg, Yosi Mass, Joris Mihaeli, Dalit Naor, and Yiftach Ravid, Access control meets public key infrastructure, or: Assigning roles to strangers, In Proceedings of the 2000 IEEE Symposium on Security and Privacy, pages 2–14, May 2000.

[6] Trevor Jim, SD3: A Trust Management System With Certified Evaluation, In IEEE Symposium on Security and Privacy, Oakland, CA, May 2001.

[7] John Kohl and Clifford Neuman, RFC 1510: The Kerberos network authentication service (V5), IETF RFC, September 1993.

[8] Ninghui Li, John Mitchell and Will Winsborough, RT: A role-based trust-management framework, In proceedings of The Third DARPA Information Survivability Conference and Exposition (DISCEX III), April 2003.

[9] Wolfgang Nejdl, Daniel Olmedilla, and Marianne Winslett, PeerTrust: Automated Trust Negotiation for Peers on the Semantic Web, In Proceedings of the Workshop on Secure Data Management in a Connected World (SDM'04) in conjunction with 30th International Conference on Very Large Data Bases, August/September 2004.

[10] Tatyana Ryutov and Clifford Neuman, The Specification and Enforcement of Advanced Security Policies, In Proceedings of the Conference on Policies for Distributed Systems and Networks (POLICY 2002), Monterey, California, June 5-7, 2002.

[11] Tatyana Ryutov, Clifford Neuman and Dongho 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.

[12] Tatyana Ryutov, Clifford Neuman, Dongho Kim and Li Zhou, Integrated Access Control and Intrusion Detection for Web Servers, In IEEE Transactions on Parallel and Distributed Systems, pages 841-850, Vol. 14, No. 9, September 2003.

[13] Tatyana Ryutov, Clifford Neuman, Dongho Kim and Li Zhou, Integrated Access Control and Intrusion Detection for Web Servers, In Proceedings of the IEEE Transactions on Parallel and Distributed Systems, Vol. 14, No. 9, September 2003.

[14] Halvard Skogsrud, Boualem Benatallah, and Fabio Casati, Model-driven trust negotiation for Web services, IEEE Internet Computing, Vol. 7, No. 6, November/December 2003.

[15] William H. Winsborough, Kent E. Seamons, and Vicki E. Jones, Automated trust negotiation, In DARPA Information Survivability Conference and Exposition, volume I, pages 88-102, Hilton Head, SC, January 2000.