

# Commodity Entities: Classifying instead of Identifying in Reputation Systems

**Delbert Hart**

SUNY Plattsburgh, U.S.A.

hartdr@plattsburgh.edu

## ABSTRACT

It is more important to know what an entities behavior is likely to be than it is to know its identifier. The commodity entity model for reputation management uses long lived reputation servers which are responsible for certifying the behavior of (commodity) entities as belonging to groups whose members' behaviors respect certain constraints. We show how this is feasible in this paper and discuss the benefits of the approach.

**Keywords:** network security, trust management, distributed agents

## 1 INTRODUCTION

Trust is a measure of how confident one is that another party will complete its role in a joint action. Thus the two aspects that trust is dependent on is the other entity and the joint action. Presumably we know what the joint action is, the other entity is the only unknown.

For most people, it is difficult to trust a stranger because you know nothing of their past and you do not know if you will ever see them again. Even if you had access to identity credentials you would not know whether the person was honest or not. Identity credentials enable you to distinguish between entities, but they tell very little about those entities. The same problem occurs with trust and interacting programs.

Consider why entities participate in a joint action. It is because the benefit gained from the joint action is greater than the benefit from individual action. This differential between joint action benefit and individual action benefit is the motivation to participate. To gain the benefit of a joint action an entity must expend some resources first. This presents an opportunity for a dishonest entity to exploit the an honest entity's willingness to participate in a joint action.

One solution taken by many systems is based on offering credentials of an entities' past actions or identification that can be correlated with the entity. There are problems with these approaches though, which mainly stem from the open nature of the systems. Entities enter and leave the system which introduces volatility and vulnerabilities associated with exploiting the trust mechanisms. Trustworthiness is associated with entities that may only be in the system long enough to take advantage of it. The more closed the environment is made to be,

the easier it is to assess the trustworthiness of the participating entities. Another way to view it is that the shorter the lifespan of an entity in a system, the greater the temptation for it to misbehave.

In this paper we describe a trust framework that transfers the tracking of trust from these transitory entities to longer lived entities, the reputation servers. We introduce mechanisms to cope with this shift in perspective of who is accountable in these interactive systems. In particular the reputation servers need to be able to protect themselves from the more transitory entities in the system. The effect of this system will be to push more responsibility onto the reputation servers. We will argue that this is beneficial though as it allows a variety of different mechanisms to be tested and deployed. It supports innovation and customization by allowing reputation servers to compete with each other using different mechanisms and the ability of the reputation servers to specialize in areas of competence.

Reputation servers play a more active role in transactions than that of credential providers. Typically a credential provider is similar to a trusted repository where information about an entity is stored. A reputation server creates groups (classes) of entities. Each of these classes has properties, or group norms, that define the behavior of entities that belong to that class. The reputation of the reputation server itself will also be on the line. Its job is to classify entities into different reputation groups. Its reputation will depend on how well it does its classification. The participants in a joint action will not view each other as unique individuals, rather they will simply consider the other a member of a group that is suitable to complete the action with. Consider how people participate in retail transactions in stores. Typically the cashier and the customer do not know each other, rather they each are members of a group

that can participate in the retail transaction.

Two key features to implement our system are strong transactional trails and Reputation Envoys. Many other systems utilize strong transactional trails. It allows entities to engage in a transaction and later use the results of the transaction to effect the reputation of the participants.

Reputation servers want to have guarantees with respect to an entities behavior over time. It can use past performance, but some attacks like the traitor attack will use a sequence of good behavior in order to perpetrate a larger attack in the future. One way to is to allow the reputation server to approve transactions as they occur. The problems with this is that it introduces an overhead to all transactions and it can result in a loss of privacy for the member entity. The solution we utilize is to use Reputation Envoys as a mechanism to protect the reputation server's interest in the member entity acting in a manner consistent with its classification. We also show how Reputation Envoys can protect member entities from exploitation due to lapses in operational security.

The next section describes the architecture of IdGuard, an example of the commodity entities model. We then discuss how commodity entities model would work in different scenarios.

## 2 IdGuard

### 2.1 Overview

IdGuard is an instance of the commodity entity model. The central idea is that it is more important to know an entity's behavior than it is to know its identity. There are two major types of entities in IdGuard, reputation servers and (commodity) entities. Entities petition reputation servers to issue credentials validating that the entity is a member of some group(s). Although there is no theoretical difference, it is convenient to distinguish between the entities in a joint action, thus we will refer to one as the (commodity) entity and the other as a service provider.

The reputation server will require proof that the entity behaves in a manner that is consistent with the group norms for the group it wants to be validated into. IdGuard validates group membership based on the identity of the owner of the entity, i.e., student users can run programs that can get membership in student groups. Other validation mechanisms are possible, for example an entity could provide a valid history of transactions to prove past behavior, or it could pay a deposit that would be forfeit if it violates the behavior standards. The validation mechanisms are left to individual reputation servers to decide. Delegating this responsibility is useful in that it allows a variety of formulas and techniques to be used simultaneously. The overall system can adapt and evolve to employ better techniques as they are developed. The diversity of mechanisms protects the

overall system from the risks associated with a monoculture, i.e., an exploit is unlikely to damage all entities in the system.

When two entities want to engage in a joint action they present the credentials that they have from the reputation servers proving their group membership. The trustworthiness of the credential is based on the reputation of the reputation server and possibly the reputation of the specific group. In most reputation servers, the reputation is primarily about the entity participating in the joint action. In IdGuard, this is inverted. The main reputation is the reputation server's reputation, then the group, and then the individual. This inversion is important because it places the most trust with the longest lived entities. It also creates incentives for the reputation server to be careful with the issuance of credentials. When a commodity entity acts outside the bounds of the group norms it means that the reputation server has misclassified it, and it reflects badly not only on the entity and group, but also on the reputation server.

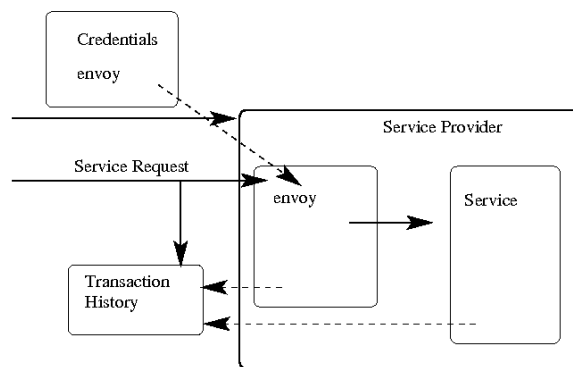
To limit the damage that can be done to a reputation server's reputation, embedded in the credentials is a Reputation Envoy. The envoy is a mobile agent that is run on service provider and its purpose is to veto behavior that is clearly outside of the group norms. The envoy also ensures that an adequate audit log is created in case there is a dispute with respect to the transaction.

The history log of the joint action is created by both parties. Communications between the parties are digitally signed and logged to the history. If the service provider decides that the commodity entity has violated the group norms of its credentials, then the service provider can forward the history to the reputation server. What the reputation server does with the history log varies depending on whether an actual violation occurred, but may include reparations to the service provider and actions against the commodity entity.

### 2.2 Transactions

The purpose of a reputation system is to facilitate transactions. Before going into the details of the components of the system we present a walk through of a typical transaction. Fig. 1 illustrates several components of the system. After a transaction has been initiated the commodity entity will send the service provider his credentials. Embedded in the credentials will be a Reputation Envoy associated with the entity's membership in the group. Optionally, at this point the service provider may attempt to contact the reputation server to check if the credentials are still valid. The service provider at this point will have enough information to decide whether the credentials provided are sufficient to proceed. Assuming that they are then the messages passed between the service provider and the entity are passed through the Reputation Envoy for

auditing. The Reputation Envoy is responsible for ensuring that the transaction is consistent with the group norms and for logging history data. The service provider may also add information to the history. After the transaction is complete either party may submit the history data to the reputation server. It is not required to submit the transaction history, but they can be used to report the entity's conformance or non-conformance with the group norms.



**Figure 1:** Before requesting a service the user sends his credentials which have an embedded Reputation Envoy. When a service is requested it is filtered through the Reputation Envoy to verify that it is consistent with the user's public policy. The request, as well as the results of the envoy's analysis and the service are recorded in a signed history.

The relationship between commodity entities and service providers is symmetric, the distinction between the two in this paper is artificially introduced in order to simplify the discussion. During the transaction credentials/envoys are sent to the commodity entity to verify the service provider's group membership and behavior.

### 2.3 Reputation Servers

Roughly speaking, reputation is a predictor of the trustworthiness of an entity. It is only an approximation though usually based on past behavior. An overview of reputation as it applies to people online is given in [1]. Although there are differences when applied to automated systems, many of the same concepts apply.

Typically reputation is tracked using a hierarchical model (similar to an X.509) or using a localized / neighbor trust models, e.g. [2]. Other models often refer to entities that provide reputation information as credential providers, and it is usually assumed that they can be trusted.

The reputation server issues a certificate that links the commodity entity to the group that it is a member of. If the commodity entity wants to enhance its privacy and anonymity it can create pseudonyms for itself and have the server issue certificates for each of the pseudonyms. Typically

the certificates will have an expiration, and internally if the commodity entity is using pseudonyms, then the reputation server will internally maintain information to be able to track them back to the commodity entity. In many systems anonymity is an invitation for abuse, but here accountability is maintained by the reputation server being able to update the reputation to reflect any abuses committed by the commodity entity. While the commodity entities can have different levels of anonymity, the reputation servers are not anonymous.

The certificates issued by the reputation server also contain Reputation Envoys, discussed in more detail in section 2.5, whose role is to enforce some behavioral constraints on the commodity entity's participation in a transaction. Envoys are important to the reputation server because they provide a mechanism prevent abuse of its certificates.

One of the more similar approaches to how we view reputation servers are the brokers used in [3]. The difference between brokers and reputation servers is that brokers are closer to trusted third parties. They can also fall back on a reputation authority that can provide information. The brokers also collect all transaction data. Trust management is delegated to the brokers in whom the member agents must implicitly trust.

In IdGuard reputation servers are not necessarily considered trust worthy, rather they each have their own reputation to maintain. The reputation server's reputation is based on how accurately it classifies other entities into behavioral groups. Reputation servers' reputation is maintained on an ad-hoc basis, entities do their own tracking of reputation servers' reputation. Other, more traditional, mechanisms for reputation management though could be used to manage the reputation servers' reputations. A meta-reputation server could also be used to track the performance of the reputation servers. We envision that eventually reputation servers would compete with one another on the basis of how accurately they classify entities.

The reputation server's address and port are obtained via an external channel, e.g., in a configuration file. In an environment with competing reputation servers though a broadcast protocol to request reputation services could be implemented.

A different take on the identity problem is entity recognition[4]. Entity recognition looks at the problem of being able to tell whether the entity you are currently interacting with is the same one that you previously had interacted with. Entity recognition does not provide as much information as typical identification does. It is weaker than what is needed for our system though. In IdGuard we need the reputation servers to assert that the commodity entity belongs to a group and to state what the properties of the members of that group are.

It is possible to view group membership as a form of attribute-based[5] authentication. Existing

attribute work though has not focused on the utility of classification in place of identification. Existing work in this field also does not consider the importance of the reputation of the classifier and the stake the classifier has in enforcing behavior norms across the group.

The commodity entity model does not specify the mechanisms used for reputation servers to classify entities into groups. Our initial design in IdGuard uses the underlying user rights to tie the entity back to an actual person. This works well in a closed system that has strong authentication mechanisms. In systems that are more open other mechanisms could be used, we discuss some possibilities in section 4.

## 2.4 Transaction History

Transactions in IdGuard generate a nonrepudiable history, meaning that either party can prove what occurred in the transaction. A strong transaction history allows the parties to update the reputation of both parties and try to appeal for remedies to the reputation servers if the conditions of the transaction were violated. Whether the reputation server will try to make amends for misbehavior on the part of a commodity entity depends on the guarantees that it associated with the group. The IdGuard system currently does not have any recourse mechanisms built into it other than invalidating an entity's group membership. It would be reasonable though for a reputation server to promise to make amends for misbehavior. Suppose commodity entities were required to make a monetary deposit as a condition upon entry to the group, then the reputation server could back the group's reputation with the promise to make monetary compensation for dishonest transactions. Without non-repudiation of histories then it would be possible for service providers to try and defraud the reputation server by reporting non-existent bad behavior.

The history consists of copies of all digitally signed control messages. Each message is stamped with the sender's logical clock. Each participant also maintains their own copy of the history. For data messages a signed digest is kept.

Some other reputations systems use verifiable histories. [6] is an example of a system that uses receipts and a history based mechanism to track reputation. One interesting aspect of it is that it uses zero knowledge proofs for identity proofs. This level of technical sophistication is not necessary though for our system since we are not so interested in individual identities as we are in classification into appropriate group membership.

Another interesting use of histories for a reputation system is [7] which uses a history based policy language to specify whether interactions are allowed or not. One implementation of it is automata based. We do not currently have a mechanism like

this in IdGuard, although there is nothing incompatible with this approach and the commodity entity model. The primary reason for the histories is to provide evidence for the inclusion or exclusion of an entity as a group member.

## 2.5 Reputation Envoys

Reputation Envoys are mobile agents that are run on a service provider on behalf of the reputation server. The envoy's role is to verify that the requests that the entity is making are consistent with the group norms for the credentials it is embedded in. Although there are many problems with mobile agents[8] most of them are not applicable to how we use them here. We chose to use mobile agents as an implementation mechanism in IdGuard because of their flexibility to express different policies. Another advantage of using code as a form of policy expression is that it makes the creation and evaluation of policies available to a wider audience of programmers/developers.

The envoys run in a restricted environment to prevent malicious or negligent reputation servers from causing havoc on the service providers. In practice though reputation envoys are likely to be standardized resulting in few unique mobile agents. The envoys are relatively short and can be embedded in the credentials, or the credentials can contain a URL at which the envoy's source code can be accessed. IdGuard uses envoys written in JavaScript which is sand-boxed within a Java runtime environment. An API is provided for the envoy to send and receive data to the service provider. The envoy is not allowed to access operating system resources such as sockets or file descriptors. A monitor is also used to watch for infinite loops or excessive processor utilization.

Embedding the envoys in the credentials is important as it guarantees that the commodity entity can not decouple the envoy mechanism from its proof of group membership. The reputation server depends on the envoy to prevent egregious behavior by either the commodity entity or the service provider. While the envoy can not actually prevent misbehavior, it provides immediate feedback to the service provider about approved behavior and what is appropriate for the commodity group. If a service provider continues with actions that the envoy does not approve of, then the service provider in a sense "voids the warranty" and the reputation server may choose not take action if the commodity entity misbehaves.

The transaction log will contain enough information for the reputation server to determine afterwards whether or not the envoy would have approved of the transaction. Keeping a digitally signed log of the messages as it proceeds prevents one of the parties from tampering with the log.

Another interesting mechanism in trust systems is that of sticky policies[9]. Sticky policies are meant





