XenoTrust: Event-based distributed trust management

Boris Dragovic, Evangelos Kotsovinos, Steven Hand, and Peter R. Pietzuch
University of Cambridge Computer Laboratory
J J Thomson Avenue, Cambridge, UK, CB3 0FD
{firstname.lastname}@cl.cam.ac.uk

Abstract

This paper describes XenoTrust, the trust management architecture used in the XenoServer Open Platform: a public infrastructure for wide-area computing, capable of hosting tasks that span the full spectrum of distributed paradigms. We suggest that using an event-based publish/subscribe methodology for the storage, retrieval and aggregation of reputation information can help exploiting asynchrony and simplicity, as well as improving scalability.

1 Introduction

The XenoServer project [15] is developing a public infrastructure for general-purpose, public distributed computing. Users of our platform can run programs at points throughout the network in order to reduce communication latencies, avoid network bottlenecks and minimise long-haul network charges, or deploy large-scale experimental services. Resource accounting is an integral part of the XenoServer Open Platform, with clients paying for the resources used by their programs and server operators being paid for running the programs that they host.

The nature of our business model for the XenoServer Open Platform as an *open* and *public* infrastructure imposes several security and consumer protection requirements; users must be protected from intruders and snoops, as well as from other participants who provide bad services or abuse the platform. In the real world, servers and clients operate autonomously. Servers may be unreliable; they may try to overcharge clients, may not run programs faithfully, or may even try to invade clients' privacy and extract secrets. Clients may attempt to abuse the platform; they may try to avoid paying their bills, or to run programs with nefarious, anti-social or illegal goals. It is easy to see the need for an efficient and scalable trust management infrastructure.

The architecture of our platform sets it apart from those within which existing distributed trust management systems operate. Unlike simple peer-to-peer recommendation services, we are concerned with *pseudonymous* users, associated with real-world identities, running real tasks on real servers for real money within a *global-scale* federated system whose constituent parts may have different notions of "correct" behaviour.

This paper describes the design and initial implementation of XenoTrust, the trust and reputation management architecture that is used in the XenoServer Open Platform, particularly focusing on its event-based functionality. In companion papers, we introduce the platform as a whole [11], and identify the threat model for XenoTrust [9].

2 Platform Overview

Figure 1 illustrates the architecture of the XenoServer Open Platform, showing the various roles and interfaces.

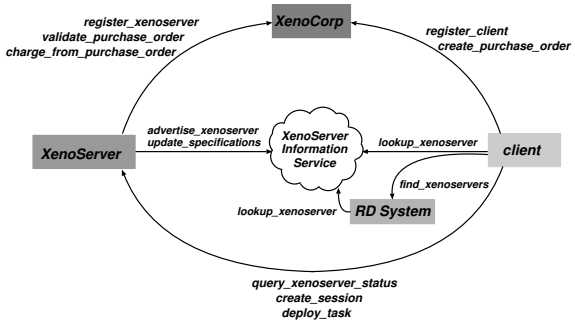

Figure 1. The XenoServer Open Platform

XenoServers host clients' tasks in exchange for money, and may be run by disparate organisations, akin to the way in which server hosting facilities currently operate. XenoServers function on a commercial basis using well-maintained machines with long-term network presence — not in "spare cycles" on users' desktop systems. This, along with the existence of XenoCorps, is an important distinction between our system and "peer-to-peer" efforts.

XenoCorp provides authentication, auditing, charging

and payment, and has contractual relationships with clients and with XenoServer operators — much as VISA or MasterCard act as intermediaries between credit card holders and the merchants from which they make purchases. It is the third party trusted by both clients and servers.

The XenoServer Information Service (XIS) is used for storing XenoServers' status updates; each XenoServer periodically reports that it is alive, and provides information about its capabilities and load. XIS uses *XenoStore*, which provides the efficient distributed storage required to store the updates. Clients may use the XIS directly to select appropriate XenoServers; however, we anticipate that most will make use of XenoSearch, which performs a matchmaking process between clients and XenoServers, receiving specifications of clients' requirements and using a search algorithm to identify a set of suitable servers.

3 XenoTrust Architecture

We consider XenoCorp to be self-authenticating, as a well-known and trusted authority which is the root of the trust delegation in the XenoServer Open Platform. Authentication of other entities in the XenoServer world is carried out by credentials issued to the entities by the XenoCorp they register with — more details on this process can be found in [11]. Entities in our platform are associated with real world, legal identities, through a XenoCorp.

As discussed in [9], we believe augmenting XenoCorp with a logically centralized "consumers relations" service is insufficient; it is unrealistic to expect all participants to agree on common standards of behaviour — and even if a single "acceptable use policy" could be defined it would most likely be written in a natural language.

Instead we suggested a two-level approach to managing trust by distinguishing *authoritative* and *reputation-based* trust: *Authoritative trust* is a boolean property established between a XenoCorp and the clients and servers that register with it. *Reputation-based trust* is a discrete continuous property which quantifies, in a particular setting, the trustworthiness that one component ascribes to another. It is distributed and highly subjective, in the sense that each entity has its own, independent view of others' reputations.

3.1 Operations and interfaces

The purpose of XenoTrust is to model, administer, accommodate and distribute the fine-grained reputation-based trust between participants in the XenoServer Open Platform. The service is open to all participants and is fully optional, consistent with the human social behaviour analogy in that in the real world we can choose to what extent, if at all, beliefs of others influence our own.

Reputation representation. Conceptually, each component forms a *reputation vector* which stores values about different aspects of the reputation of other components in the platform. This information is built through inter-component interaction based on the individual experiences and criteria. Different aspects of component behaviour — like performance or honesty — are reflected in the reputation vectors through fields called *tokens*.

Statement advertisement. A community as a whole may benefit from exchanging reputation information, or, inline with the real world example, *gossipping*. This is of particular interest to newcomers who have not yet formed opinions based on personal experience.

A *statement* is the basic unit of reputation information advertised. A statement is a tuple {`advertiser, subject, token, value(s), timestamp`}, comprising the identity of the advertiser and the subject, a token denoting the aspect of the subject's reputation that is being considered, a series of values indicating the extent of this reputation, and a timestamp. The tuple is signed using the advertiser's authoritative credentials to prevent forgery but, beyond this, XenoTrust does nothing ensure that statements are in any sense valid: as with real-world advertisements, users are under no compulsion to believe what they see or to pay any attention to it. Moreover, statement advertisements are chosen to be *soft-state*, as the requirements from our trust management architecture do not impose, we believe, a need for long-term, reliable storage of reputation information (see Section 3.4).

Rule-set deployment. Components are welcome to search XenoTrust directly for statements that have been made; e.g. to issue queries of the form "return all the statements naming participant A". However, this scheme is far from scalable and will only take into account direct statements made about the component in question. Instead, the approach that we take is to move the computation of reputation vectors from the participants involved into XenoTrust itself. This allows aggregation of information to take place within XenoTrust and, crucially, allows it to be updated incrementally as new statements are received. The results of this computation can also be shared between participants using the same rule-sets to derive their reputation vectors.

Reputation retrieval. When participants deploy rule-sets, they specify whether they will use the event-based mechanism, or simply use polling. In the former case, XenoTrust will notify them when a significant change happens (see Section 3.3). In the latter case, the final step in using XenoTrust is for a component to *query* the rule-sets that it has deployed in order to retrieve elements from its reputation vector.

3.2 Rule-set language

The choice of a rule-set description language is crucial, as it specifies the expressiveness, as well as complexity, of the XenoTrust model. In order to accommodate a wide variety of different types of queries while keeping the XenoTrust interface simple and efficient, XenoTrust directly supports only simple *atomic* rule-sets. Services running above XenoTrust (e.g. XenoSearch) are responsible for decomposing complex rule-sets, forwarding the atomic ones to XenoTrust, and subsequently combining the results.

The atomic rule-sets supported by XenoTrust take the form: `{principal, property, advertiser, function, [trigger]}`, where:

- *principal* is the subject of the query;
- *property* is the token that the query refers to — e.g. performance, bandwidth, stability;
- *advertiser* is a nonempty set of components whose advertisements are considered during rule-set evaluation;
- *function* is a means to aggregate reputation information from advertisements matching the above criteria — e.g. min, max, mean, avg, and;
- *trigger* is the threshold beyond which changes to the result of the rule-set evaluation triggers an event notification to the user.

Specifying a trigger is optional; components that do not wish to subscribe for explicit event notification simply do not include triggers in their rule-sets.

3.3 Event-based design

Most of the existing trust management architectures depend on the traditional *request/reply* paradigm; participants request trust or reputation information from the infrastructure, and then receive replies in return. This imposes a need for *polling*, with all the associated communication overhead. To avoid this overhead we advocate the use of an *event-based*, publish/subscribe [10] framework for the efficient advertisement of changes in reputation information.

Participants can *subscribe* to XenoTrust to receive events whenever a defined change in the evaluation of their rule-sets occurs. XenoTrust keeps track of the changes imposed when incoming reputation statements are *published* (advertised), calculates their impact on those rule-sets subscribed for event notification, determines which participants need to receive updates and notifies them accordingly. This event-based design also allows very efficient rule-set aggregation.

3.4 Discussion

In the following we discuss some open issues and questions, and present our initial thoughts.

Bootstrapping One obvious problem is the absence of reputation information in XenoTrust at the start-of-day. This means that, for some time after the platform has started its operation, querying XenoTrust will be of little value. However, if the right *economic incentives* are provided for components to participate in the scheme, it is expected that the critical mass of reputation information will be gathered reasonably soon after the start-of-day.

Negative reputation problem Observing human social behaviour, it is more common for people to give negative feedback in response to poor service than to praise service received at or above expectations. Therefore, it can be expected that the majority of statements will contain negative reputation information, ultimately leading to a continual global decrease in reputations, which could even get stuck to the minimum. One simple solution might involve making reputations relative to the average.

Statement expiry time Statement advertisement is soft-state; statements are unreliably stored, and can be deleted when they expire or when there is no more space to store them. A malicious component might thus behave badly in the hope that negative reports will expire before subsequent interactions. However, we expect the lifetime of statements will not be so short as to encourage such behaviour; and even if it were, clients also maintain a local reputation vector, which they may consult and/or reinject to XenoTrust.

Sybil and Replay attacks *Sybil attacks* [8] are based on a single malicious/faulty entity presenting multiple identities to create a false reputation picture in the system. We explored the threat in depth in [9]. There are two main reasons why Sybil attacks are unlikely in our architecture: difficulty and cost of obtaining identities and inability to estimate the impact of an attack, due to the nature of the trust model.

Replay attacks denote attempts to copy packets that pass by and reinsert them later into a network to fool the legitimate receiver [17]. With respect to our system, this would comprise submitting multiple copies of reputation statements. Signatures themselves provide only content authenticity and integrity; our approach was to include *timestamps* creating the notion of statement freshness. Secure clock related issues are addressed in [16, 3].

4 Deploying XenoTrust

This section considers the deployment of the XenoTrust service and relevant parts of the two services it co-operates with: XenoSearch and XenoStore.

Statement advertisements Reputation statements are stored in XenoStore through XIS; for every new participant in the XenoServer Open Platform, XenoCorp allocates a XenoStore *container* for it, part of which is used for storing the reputation statements the new participant submits. Those statements are made accessible to authenticated users. XenoTrust then collects those reputation statements from the respective containers, and stores them together in a structured manner. The exact way of structuring data inside XenoTrust can vary from one implementation to another; in our prototype, we decided to use a simple, off-the-shelf Relational Database Management System (MySQL).

Effectively, XenoTrust may collect statements from the various containers using *distributed call-backs*, where XenoTrust gets explicit notification of new statements being stored in a XenoStore container, or *polling*, where XenoTrust has to regularly check the individual XenoStore containers for new statement advertisements. For our prototype implementation we decided to employ the latter approach. Although using call-backs would slightly simplify the implementation of XenoTrust, it would not offer many substantial advantages. Also, the complexity of the XenoServer Information Service's design would rise dramatically, as it is a general-purpose system and call-backs and interfaces would have to be reasonably generic.

Rule-set deployment The first step in obtaining reputation information from XenoTrust is deploying a suitable rule-set. Following the description of the rule-set language XenoTrust supports in Section 3.2, we decided to use SQL for implementing rule-sets in our prototype; when a rule-set is submitted to XenoTrust, the *rule-set parser* module checks if it is a valid, atomic rule-set, and translates it to an SQL query. Apart from simplicity, performance, public acceptance and ease of use, one of the incentives for using SQL is the ease of translating atomic rule-sets to it.

To illustrate the operation of the rule-set parser with respect to the translation described above, as well as the decomposition of complex rule-sets to atomic ones — done by XenoSearch — we consider the following example: client A submits the query "What is the average performance reputation of XenoServer B". As we described in Section 3.2, a rule-set is effectively a {`principal, property, advertiser, function, [trigger]`} tuple. Deploying a rule-set involves calling the `deploy` interface exported by XenoTrust, and providing the rule-set tuple. Thus, this rule-set can be deployed by using the interface `deploy(B, performance, -, AVG)`. The parser then translates this to

```
SELECT AVG(performance) FROM reputations
                        WHERE subject = 'B'
```

In the case of a complex rule-set like "What is the average performance reputation of XenoServer B, computed on statements made by components that I valued to be in the range of 0.5-0.7 with respect to their honesty", XenoSearch will decompose the rule-set into several atomic ones: "Who are the XenoServers about who A has issued statements, valuing their honesty to be between 0.5-0.7?" and "What is the average performance for B computed on statements issued by X", where X is each of the XenoServers that the first query returned. Accordingly, it will deploy the two rule-sets by calling `deploy(-, performance, A, -)` and `deploy(B, performance, X, AVG)`. The parser will then generate the following SQL queries:

```
SELECT id FROM reputations WHERE advertiser = 'A'
          AND (honesty > 0.5 and honesty < 0.6)

SELECT AVG(performance) FROM reputations
       WHERE principal=B AND subject=X
```

We believe that, given the simple rule-set language we propose, and the expressiveness of SQL queries, XenoSearch along with the XenoTrust rule-set parser can decompose and translate to SQL queries for most of the complicated questions that clients may pose.

Reputation retrieval Each rule-set deployed in XenoTrust is marked with a unique identifier, which is passed back to the user who deployed it. This way, the user can refer to the rule-set in the future, either to view, change or remove the rule-set, or to ask for it to be evaluated. The results of a rule-set that has been deployed can be obtained either by *polling* XenoTrust, asking for a rule-set evaluation and providing its identifier, or by subscribing for explicit *event notification*, specifying on what occasion the participant would like to be notified — as explained in Section 3.3

The implementation of polling is simple; XenoTrust exports the *evaluate* interface through which users can invoke this operation. For rule-sets that have subscribed to event notification, XenoTrust evaluates them periodically after checking the XenoStore containers for new statement advertisements. If the result evaluated result crosses the trigger value, an event is multicast to the interested users.

Event composition Regarding events, we believe that XenoTrust should support *atomic events*, which are primitive notifications. An atomic event in XenoTrust is triggered when the result of an atomic rule-set changes by more than a predefined threshold — the "trigger" value in the rule-set. However, applications will be more interested in the combination of multiple atomic rule-set changes, i.e. in the occurrence of *composite event patterns* that can be used for trust-based decisions.

We delegate the task of monitoring complex changes in rule-sets to a dedicated *composite event detection layer* that runs on top of our platform as a generic middleware service. The framework described in [14] enables clients to

detect arbitrary patterns of events in a distributed system in an efficient and scalable way. Few assumptions are made about the kind of publish/subscribe system the composite detection service is deployed on. The language used to specify composite event subscriptions is an extension of a regular language with operators for timing, concurrency, and parameterisation. As a result, instances of composite events can easily be detected with extended finite state automata. Since certain composite events in XenoTrust will be more popular than others, it is important to be able to reuse composite event subscriptions. The composite event framework supports this by decomposing complex composite event subscriptions into subexpressions that are then distributed throughout the system in the form of subdetectors.

5 Related Work

Over the last ten years, the notion of trust has evolved immensely, from the first theoretical models [4], to spanning electronic communities [12]. The development of trust models has progressed in multiple directions, such as online auctions and retailers — like eBay and the amazon marketplace — and "regulator agent" based models [7]. Problems in this class of systems include very limited expressiveness, single-dimensional reputation and assumed uniformity of criteria. Another popular research avenue has been in the area of peer-to-peer systems. Various models [1, 18, 2] have been developed, though showing excessive resource usage (network and peer), information redundancy, and high vulnerability. A different notion of trust, viewed from the perspective of formulating security policies and credentials, is represented by projects such as PGP, X.509, PolicyMaker [6] and KeyNote [5]. Probabilistic approaches like Maurer [13] have also been proposed.

Our system differs from these in that it combines hard security trust and soft, reputation-based trust within a flexible and scalable architecture, suitable for global-scale systems. Furthermore, we are not aware of any work incorporating the notion of explicit event notification in a trust management architecture.

6 Conclusion

In this paper, we have presented the design and initial implementation of our distributed trust and reputation management architecture, named XenoTrust, which provides interfaces for the users to submit their own perception about others' reputations, allows for simple reputation aggregation rule-sets to be deployed, and provides asynchronous event notification on significant reputation changes for its users. We believe that our approach of exploiting distributed storage, simple rule-sets, as well as event notification gives our system unique flexibility, efficiency and ease of use.

7 Acknowledgements

We would like to thank Jon Crowcroft, Tim Deegan and Tim Harris for their valuable suggestions, as well as Marconi PLC for the financial support of Evangelos Kotsovinos' research.

References

[1] A. Abdul-Rahman and S. Hailes. Supporting Trust in Virtual Communities. In *Proc. HICSS*, January 2000.

[2] K. Aberer and Z. Despotovic. Managing Trust in a Peer-2-Peer Information System. In *CIKM*, pages 310–317, 2001.

[3] H. Attiya, A. Herzberg, and S. Rajsbaum. Optimal clock synchronization under different delay assumptions (preliminary version). In *Symposium on Principles of Distributed Computing*, pages 109–120, 1992.

[4] T. Beth, M. Borcherding, and B. Klein. Valuation of Trust in Open Networks. In *Proc. ESORICS'94*, pages 3–18, 1994.

[5] M. Blaze, J. Feigenbaum, and A. D. Keromytis. KeyNote: Trust Management for Public-Key Infrastructures (Position Paper). *LNCS*, 1550:59–63, 1999.

[6] M. Blaze, J. Feigenbaum, and J. Lacy. Decentralized Trust Management. In *Proceedings of the 1996 IEEE Symposium on Security and Privacy*, pages 164–173, May 1996.

[7] A. Chavez and P. Maes. Kasbah: An agent marketplace for buying and selling goods. In *Proc. PAAM'96*, pages 75–90, London, UK, 1996.

[8] J. R. Douceur. The Sybil Attack. In *Proc. IPTPS*, 2002.

[9] B. Dragovic, S. Hand, T. Harris, E. Kotsovinos, and A. Twigg. Managing trust and reputation in the XenoServer Open Platform. Jan. 2003. Accepted for publication at 1st International Conference on Trust Management.

[10] P. Eugster, P. Felber, R. Guerraoui, and A. Kermarrec. The many faces of publish/subscribe. Technical report, EPFL, Lausanne, Switzerland, 2001.

[11] S. Hand, T. Harris, E. Kotsovinos, and I. Pratt. Controlling the XenoServer Open Platform. In *Proc. OPENARCH '03*, Apr. 2003.

[12] S. Marsh. *Formalising Trust as a Computational Concept*. PhD thesis, Department of Mathematics and Computer Science, University of Stirling, 1994.

[13] U. Maurer. Modelling a Public-Key Infrastructure. In *ESORICS: European Symposium on Research in Computer Security*. LNCS, Springer-Verlag, 1996.

[14] P. R. Pietzuch, B. Shand, and J. Bacon. A Framework for Event Composition in Distributed Systems. In *Proc. Middleware'03*, June 2003.

[15] D. Reed, I. Pratt, P. Menage, S. Early, and N. Stratford. Xenoservers: accounted execution of untrusted code. In *Proc. HotOS-VII*, 1999.

[16] F. B. Schneider. A paradigm for reliable clock synchronization. Technical Report TR86-735, 1986.

[17] P. Syverson. A taxonomy of replay attacks. In *Computer Security Foundations Workshop VII*. IEEE Computer Society Press, 1994.

[18] B. Yu and M. P. Singh. A Social Mechanism of Reputation Management in Electronic Communities. In *Cooperative Information Agents*, pages 154–165, 2000.