

Towards a Formal Notion of Trust

Extended Abstract^{*}

Mogens Nielsen
BRICS, University of Aarhus
mn@brics.dk

Karl Krukow[†]
BRICS, University of Aarhus
krukow@brics.dk

ABSTRACT

Trust management systems have been proposed as an alternative to traditional security mechanisms in Global Computing. We present some challenges in establishing a formal foundation for the notion of trust, and some preliminary ideas towards a category of trust models.

Categories and Subject Descriptors

D.3.1 [Programming Languages]: Formal Definitions and Theory—*Semantics*; D.4.6 [Software Engineering]: Security and Protection—*Access controls, authentications*

General Terms

Security, Languages, Theory.

Introduction

Global Computing (GC) is an emerging aspect of computer science and technology. A GC system is composed of a large population of diverse but co-operating entities, which we will call “principals”. Principals will typically be autonomous and mobile, and will have to deal with unforeseen behaviour, like unexpected interactions and disconnected operation. Examples of GC systems could be the Internet, personal area networks, or vehicle area networks.

The properties of the GC infrastructure introduce new security challenges, which are not adequately addressed by traditional security mechanisms. One aspect is that GC security policies must encompass a huge number of potential collaborators, which are mobile and virtually anonymous,

^{*}A full version of this paper will appear as a *BRICS technical report* at www.brics.dk

[†]MN and KK are supported by ‘**SECURE**: Secure Environments for Collaboration among Ubiquitous Roaming Entities’, EU FET-GC IST-2001-32486. BRICS, Basic Research In Computer Science funded by the Danish National Research Foundation.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

PPDP '03 August 27–29, 2003, Uppsala, Sweden.
Copyright 2003 ACM 1-58113-705-2/03/0008 ...\$5.00.

and hence principals cannot rely on a specific security infrastructure such as certificate authorities and authorisation servers.

As an alternative to traditional security mechanisms, it has been suggested from many independent sources to base a GC security infrastructure on *trust management systems*, in which a principal makes safety critical decisions based on *trust policies* and their deployment in the presence of partial knowledge on the trustworthiness of other principals, see [6] for a survey. The basic idea behind trust management systems is that risks taken in the co-operation between principals are based on the trust that principals have in each other, much in the same way that trust plays a role in the day-to-day co-operation between humans and organisations. The ultimate aim is to transfer this role of trust to the GC scenarios. However, trust as it is studied in social sciences is an informal concept, whereas trust as a concept within GC needs to be formal, in the sense that it needs to be implementable and amenable to formal reasoning. Nevertheless, the vast literature on trust from the social sciences, psychology, and philosophy, is a good source of inspiration when attempting to formalise trust. Trust is usually defined as a relationship between a *trustor* (the principal that trusts someone) and a *trustee* (the principal being trusted). In [8], a survey of the literature on trust is provided, classifying trust with respect to e.g. disposition, situation, structure, belief, or intention, and classifying trustees with respect to e.g. competence, benevolence, integrity, or predictability.

In the project SECURE, Secure Environments for Collaboration among Ubiquitous Roaming Entities, we are investigating the design of trust based security mechanisms in the GC setting. A more detailed description of the project including risk models, software frameworks, and applications may be found in [4]. Here we focus on the challenges in establishing a formal model of trust, which on the one hand may accommodate versions of the various informal notions of trust from the human co-operation mentioned above, and on the other hand is a formal model useful as the mathematical foundation for trust management systems in GC.

Trust management systems

Trust management systems were originally introduced by Blaze et al. in [1], and the authors have since then implemented several automated trust management systems including PolicyMaker [3] and KeyNote [2].

In a trust management system the principals involved are atomic entities which assign some degree of trust to other principals based on a combination of behaviour observations, recommendations from other sources, and references to the trust of other principals. This is typically done by each principal defining its own *trust policy*. The degree of trust that a principal P has in principal Q governs the decisions that P makes when interacting with Q. For example, if P trusts Q highly then P might be willing to take the risks involved in interaction, whereas a low degree of trust could mean that the interaction is not worth the risk, or that some form of restricted interaction is preferred.

As a simple example, the decisions made by a trust management system could be whether to give access to some fixed resource, r . Say that this resource allows different levels of access: full use, \top , partial access p , another partial access q and no access \perp . These access rights constitute a complete lattice by taking \top to be the greatest element, \perp to be the least element, and p and q to be unrelated. Note that the order on this lattice expresses *more access rights*. An example of this could be allowing read-write, read-only, write-only or no access, respectively, to some fixed file. Another could be a database, allowing access to all information, some levels of restricted access or no access at all.

Now imagine that principal A controls resource r . When principal B wants to access r it makes a request to A which will then compute its trust in B. If the trust is sufficiently high, access will be granted. A's trust in B is computed according to A's trust policy, for example, if A knows B well, it might know that B is trusted to access-level p , thus its trust policy is to assign p to B. However, in the GC setting it will often be the case that A does not know B, and hence a more complex policy for A could be something like "the trust in any principal P, given that C's trust in P is t and that D's trust in P is t' , is the greatest lower bound of the two, $t \sqcap t'$ ". In this policy A's trust in any P depends on the trust of C and D, which in turn may depend on other principals. Hence the collection of trust policies becomes a web of policies *referencing* each other.

A simple example Policy Language

In any trust management system there is a need for specifying, formally, the trust policies of each principal. Here we give a very simple, yet non-trivial, example of such a language which allows a number of basic primitives and operations that are useful for defining trust policies. The figure (1) gives syntax for this simple, declarative trust policy language which is parametric in a domain of trust values, D . A policy (π) is essentially a non-empty list of "specific policies". Each entry in the list specifies a policy for a specific principal P. There is always one special list-entry which specifies policy for every principal not occurring in the list. One could easily imagine simple extensions where instead of referring to specific principals one could refer to groups of principals. The syntactic category τ will denote trust *values*. We can denote constant values, simple conditionals (e is unspecified here but should range over simple expressions), lattice operations of join and meet and finally referencing. In the GC scenario referencing is particularly important since it is impossible for each principal to have

Figure 1: A minimal policy language, parametric in D , with principals \mathcal{P}

$\pi ::= x : \tau$	(general policy)
$p : \tau; \pi$	($p \in \mathcal{P}$, principal policies)
$\tau ::= d$	(constant, $d \in D$)
if e then τ else τ	(conditional)
$\tau \circ \tau$	(\circ is \sqcup/\sqcap)
$p?q$	(reference to $p, q \in \mathcal{P} \cup \{x\}$)

complete information about all other principals, thus principals depend highly on the experiences and opinions of other principals. Informally, the semantics of the construct $p?q$ is the trust that principal p has in principal q .

Weeks recognised in [10] that many such systems could be expressed by considering complete lattices of "trust values" (authorizations in [10]). Given a collection of principals \mathcal{P} with trust values D , the idea is to view the trust information of a system, that is the trust that principals have in each other, as a function $\mathcal{P} \rightarrow \mathcal{P} \rightarrow D$, where $T(P)(Q)$ means principal P's trust in the principal Q. A particular policy with referencing may then be viewed as a function $(\mathcal{P} \rightarrow \mathcal{P} \rightarrow D) \rightarrow \mathcal{P} \rightarrow D$, and hence the collection of policies may be seen as a function $(\mathcal{P} \rightarrow \mathcal{P} \rightarrow D) \rightarrow (\mathcal{P} \rightarrow \mathcal{P} \rightarrow D)$. Assuming that the trust values D form a complete lattice as in the example above, and that the trust policies (known in [10] as licences) are monotone, the semantics of the collection of trust policies is then defined to be the least fixed point of this function, that is an element of $\mathcal{P} \rightarrow \mathcal{P} \rightarrow D$ representing the actual trust amongst principals. Then trust policies could be expressed in application specific trust policy languages and the semantics of these policies was defined as the least fix-point of the collection of such policies.

As an example, the situation described in the preceding section could be expressed syntactically as the collection

$$\begin{aligned} \pi_A &\equiv x : (C?x) \sqcap (D?x) \\ \pi_C &\equiv B : \top ; x : \perp \\ \pi_D &\equiv B : q ; x : \perp \end{aligned}$$

Suppose that principals A, C and D have defined policies like those above, and that U is some newcomer principal that none of the principals involved in the trust policies of C or D has any knowledge of – a situation that is very likely to occur in a GC setting. If U requests permission to access r , A will compute its trust in U, in this case C's and D's trust will evaluate to \perp since U is unknown, thus also A's trust in U will be \perp . A similar situation occurs if C or D is temporarily offline when A computes its trust. However, this will often be too restrictive for a GC environment in which principals with no prior knowledge of each other should be able to interact in a way that will benefit both.

One problem in the above is that there is no distinction between an *unknown* principal and a principal which is *known*

to be *dis-trusted*. In the example there is no reason to *dis-trust* U , so rather than resulting in \perp , the computation should result in an element, expressing that U is unknown to A (or that there is currently no information about U). Of course this is a “sound” approach but it is also too restrictive for our purposes, hence a notion of *uncertainty* or partial information is needed.

Seen from a programming language perspective, a challenge related to this problem is to identify a family of languages for expressing trust policies. Such languages must be *expressive* enough to make it possible to write complex trust policies specifying the desired dynamic behaviour of the given application. Ideally the family of languages should not only be able to express a range of useful, new GC-trust management systems, but also be capable of expressing a large class of existing trust management systems in a simple and relatively concise manner. Together with such languages comes the need for a general semantic domain which allows for a formal semantics of such languages and facilitating reasoning about properties of policies expressed in these languages. These semantic domains must be able to adequately deal with the notions of uncertainty or partial information so as to be viable in the GC scenario. They must also be “operational” by nature so that if exact semantic computation is not tractable then at least some form of approximation is possible.

An example of how to approach the problem of finding a “semantic domain” is by considering a set of “trust values” with two orderings \preceq and \sqsubseteq (for more technical details the interested reader should refer to [5]). The order \preceq corresponds to the ordering in the example and it expresses *more trust*. In contrast, the order \sqsubseteq expresses *more information*. Now instead of computing fix-points with respect to \preceq we compute it with respect to \sqsubseteq thus emphasising *information*. The situation with the unknown principal could now be handled differently. Since fix-points are computed with respect to \sqsubseteq , the result will now be the bottom element \perp_{\sqsubseteq} which means that A ’s trust in U is the least element in the information order, thus representing that there is complete uncertainty about U . The action that A will take depends on its *security policy*, which specifies how to act *given* the current trust information. Notice that this is different from the approach in [10] where the computed value, say p , immediately tells us that “the principal in question shall have the access right p ”. In our approach we split the problem of computing a trust value from the problem of interpreting that value, as to how the request is handled. So we distinguish a *trust policy* from a *security policy*, the former specifying *how trust is computed* and the latter specifying *how to respond to a request given the computed trust information*.

A family of semantic domains

We give an example of how to construct a semantic domain that has a notion of uncertainty, and which can be used as a mathematical framework for giving semantics to trust policy languages like the one in figure (1). The paper [5] describes the construction in detail along with a collection of nice properties that the construction has. By using ideas from general lattice theory and Scott’s CPOs some useful results are proved including products to express context dependent trust. In this section we briefly show how to construct from

any complete lattice (D, \leq) a trust domain, $I(D, \leq)$, which has the original lattice (D, \leq) as a sublattice and has a natural notion of uncertainty. This construction is known as the interval construction and has previously been studied in the context of computer arithmetic ([9]).

DEFINITION 1. For any complete lattice (D, \leq) and any $d_0, d_1 \in D$ let $[d_0, d_1]$ denote the interval from d_0 to d_1 , that is, $[d_0, d_1] = \{d \in D \mid d_0 \leq d \text{ and } d \leq d_1\}$. We define the intervals of D , ID as the set

$$ID = \{[d_0, d_1] \subseteq D \mid d_0, d_1 \in D, d_0 \leq d_1\}$$

There are two orderings that are useful on this set. The trust ordering \preceq defined by

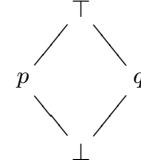
$$[d_0, d_1] \preceq [d'_0, d'_1] \text{ iff } d_0 \leq d'_0 \text{ and } d_1 \leq d'_1$$

and the information ordering \sqsubseteq defined by

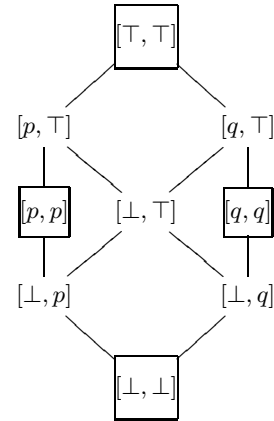
$$[d_0, d_1] \sqsubseteq [d'_0, d'_1] \text{ iff } d_0 \leq d'_0 \text{ and } d'_1 \leq d_1$$

Note that the definition of \sqsubseteq , implies that $[d_0, d_1] \sqsubseteq [d'_0, d'_1]$ is equivalent to saying that $[d'_0, d'_1]$ is contained in $[d_0, d_1]$ as subsets of D . We thus view $[d'_0, d'_1]$ as a more precise element than $[d_0, d_1]$ since the former has *excluded more possibilities*.

If we consider the lattice mentioned previously:



Then applying the interval construction gives the following structure with the trust ordering \preceq



Notice that the original lattice is (isomorphic to) a sublattice of this lattice. The information ordering on this set is defined by reverse inclusion and thus has $[\perp, \top]$ as a least element, expressing complete uncertainty.

We have the following result [5] showing that any interval construction provides us with a domain suitable for giving a formal semantics to a class of policy languages similar to the language in figure (1).

PROPOSITION 2. The structure (ID, \preceq) defined as in definition (1) is a complete lattice. The order \sqsubseteq makes (ID, \sqsubseteq) a CPO with bottom, \perp_{\sqsubseteq} .

In the full paper currently being developed we give a first attempt at defining a category of “trust domains” – that is identifying abstractly the properties that structures should satisfy in order to be useful as semantic domains for trust policy languages in GC environments. Our category contains a range of important examples from the literature, including the belief model from [7]. We have a preliminary result which proves that the interval construction has certain universal properties in this category.

In the GC scenario, with a huge number of networked principals, giving a fixpoint semantics to the collection of trust policies of all principals is dangerous since it is intractable to actually compute the fixpoint. However, in many situations it is not necessary to have the information of the exact fixpoint as many security decisions could be made on the basis of knowing certain properties of the fixpoint or by computing approximations. We see promising possibilities of exploiting many of the results and techniques from declarative programming in this context. Also the theory should be capable of proving the soundness of efficient strategies for “proof carrying requests”, ie. scenarios where the requester is expected to provide evidence that he is trusted sufficiently for his request to be met.

Acknowledgments

We would like to thank the SECURE project consortium and Vladimiro Sassone for their co-operation.

1. REFERENCES

- [1] M. Blaze, J. Feigenbaum, and J. Lacy. Decentralized trust management. In *Proc. IEEE Conference on Security and Privacy, Oakland*, 1996.
- [2] M. Blaze, J. Feigenbaum, and J. Lacy. KeyNote: Trust management for public-key infrastructure. *Springer LNCS*, 1550:59–63, 1999.
- [3] M. Blaze, J. Feigenbaum, and M. Strauss. Compliance checking in the policymaker trust management system. In *Financial Cryptography*, pages 254–274, 1998.
- [4] V. Cahill et al. Using trust for secure collaboration in uncertain environments. to appear in *IEEE Pervasive Computing*, 2003.
- [5] M. Carbone and V. Sassone. A formal model for trust in dynamic networks. to appear in the proceedings from Software Engineering and Formal Methods, SEFM’03, IEEE Computer Society Press, 2003.
- [6] T. Grandison and M. Sloman. A survey of trust in internet application. *IEEE Communications Surveys, Fourth Quarter*, 2000.
- [7] A. Jøsang. A logic for uncertain probabilities. *Fuzziness and Knowledge-Based Systems*, 9(3), 2001.
- [8] D. H. McKnight and N. L. Chervany. The meanings of trust. *Trust in Cyber-Societies - LNAI*, 2246:27–54, 2001.
- [9] W. L. M. Ulrich W. Kulisch. *Computer Arithmetic in Theory and Practice*. Computer Science and Applied Mathematics. Academic Press Inc., 1981.
- [10] S. Weeks. Understanding trust management systems. In *Proc. IEEE Symposium on Security and Privacy, Oakland*, 2001.