8

Trust Management

Claudio A. Ardagna, Ernesto Damiani, Sabrina De Capitani di Vimercati, Sara Foresti, and Pierangela Samarati

Università degli Studi di Milano
Italia

Summary. The amount of data available electronically to a multitude of users has been increasing dramatically over the last few years. The size and dynamics of the user community set requirements that cannot be easily solved by traditional access control solutions. A promising approach for supporting access control in open environments is *trust management*.

This chapter provides an overview of the most significant approaches for managing and negotiating trust between parties. We start by introducing the basic concepts on which trust management systems are built, describing their relationships with access control. We then illustrate credential-based access control languages together with a description of different trust negotiation strategies. We conclude the chapter with a brief overview of reputation-based systems.

8.1 Introduction

Accessing information over the Internet has become an essential requirement in the modern economy, where unknown parties can interact for the purpose of acquiring or offering services. The open and dynamic nature of such a scenario requires the development of new ways of enforcing access control, as identity-based mechanisms are not able to manage these issues any more. In fact, interacting parties may be unknown to each other, unless they have already had transactions before. Consequently, a mechanism that allows one to decide which requesters are qualified to gain access to the resource and, on the other hand, which server is trusted to provide the requested resource, on the basis of certified statements provided by the interacting parties is needed. *Trust management* has been developed for this specific purpose and has received considerable interest from the research community [25]. Early research identified three main components of trust management: (i) *security policies*, which are local trust assertions that the local system trusts unconditionally; (ii) *security credentials*, which are signed trust assertions made by other parties; the signature must be verified before the credential may be used; (iii) *trust relationships*, which are special cases of security policies. Early approaches to

trust management, such as PolicyMaker [5] and KeyNote [4], basically use credentials to describe specific delegation of trusts among keys and to bind public keys to authorizations. Although early trust management systems do provide an interesting framework for reasoning about trust between unknown parties, assigning authorizations to keys may be limiting and make authorization specifications difficult to manage. Moreover, the public key of a subject may eventually be considered as her pseudonym, reducing the main advantages of trust management.

A promising direction to overcome this disadvantage is *digital certificates*. A digital certificate is basically the online counterpart of paper credentials (e.g., driver licenses). Access control models exploiting digital certificates make access decisions on whether or not a party may execute an access on the basis properties that the requesting party may have. These properties can be proven by presenting one or more certificates [6, 13, 15, 18, 34]. The development and effective use of credential-based access control models require tackling several problems related to credential management and disclosure strategies, delegation and revocation of credentials, and the establishment of credential chains [10, 16, 23, 24, 28, 29, 31]. In other words, trust between two interacting parties is established based on the parties' properties, which are proven through the disclosure of digital certificates. First of all, parties must be able to state and enforce access rules based on credentials and communicate these rules to their counterpart to correctly establish a negotiation. The resolution of this problem requires the development of new access control (authorization) languages and systems.

The main advantages of trust management solutions can therefore be summarized as follows.

- Trust management allows unknown parties to access resources/services by showing appropriate credentials that prove their qualifications to get the resources/services.
- Trust management supports delegation and provides decentralization of control as it allows trust chains among parties to propagate access rights.
- Trust management is more expressive than classical access control mechanisms as it allows the addition of new restrictions and conditions without the need to rewrite the applications enforcing access control.
- The use of trust management systems for controlling security-critical services frees the application programmers from designing and implementing security mechanisms for specifying policies, interpreting credentials, and so on.
- Each party can define access control policies to regulate accesses to its resources/services.
- Trust management systems increase the expressiveness and scalability of access control systems.
- Trust establishment involves just two parties: the requester and the service provider.

The concept of *reputation* is closely linked to that of trustworthiness. Reputation is often considered as a collective measure of trustworthiness based on ratings from parties in a community and can be used to establish trust relationships between parties [9]. The basic idea behind reputation management is to let remote parties rate each other, for example, after the completion of a transaction, and use the aggregated ratings about a given party to derive a reputation score. Reputation can then be used by other parties when deciding whether or not to transact with that party in the future. Linking reputations to parties and/or to their attributes impacts on the trust framework inasmuch as it poses additional requirements on credentials production and management. A rapidly growing literature is becoming available around trust and reputation systems, but the relation between these notions needs further clarification.

The purpose of this chapter is to give an overview of existing and proposed approaches to trust management. The remainder of this chapter is organized as follows. Section 8.2 gives an overview of early approaches for trust management, namely PolicyMaker, KeyNote, and rule-controlled environment for evaluation of rules and everything else (REFEREE). Section 8.3 presents the main characteristics of credential-based access control systems and illustrates a credential-based access control language together with some trust negotiation strategies. Section 8.4 presents a brief overview of reputation-based systems. Finally, Sect. 8.5 gives our conclusions.

8.2 Early Approaches to Trust Management

With the growing popularity of the Internet, trust-based systems are becoming increasingly prevalent. In such a context, trust is an inherently dynamic measure. For instance, party A previously trusting party B but its public key authority may decide to stop trust if party B vouches for bad public key bindings. The level of trust may therefore increase or decrease depending on new knowledge and experiences learned from exercising the trust. In general, trust management systems may differ in the approach adopted to establish and evaluate trust relationships between parties. Early approaches to trust management used credential verification to establish a trust relationship with other parties. These systems start from the proposal of binding authorizations with keys rather than with users' identities.

We now give an overview of the early trust management approaches to authorization and access control, focusing on the *PolicyMaker*, *KeyNote*, and *REFEREE* systems.

8.2.1 PolicyMaker and KeyNote

PolicyMaker [5] and KeyNote [4] provide a comprehensive approach to trust management, defining a language used to specify *trusted actions* and relations. In the past, identity-based certificates were used to create an artificial

layer of indirection, linking a user's public key to her identity (e.g., X509 certificates [7]), and the user's identity to the set of actions she is authorized to execute. PolicyMaker and KeyNote are, instead, systems that integrate the specification of policies with the binding of public keys to the actions they are trusted to perform. The ability to express security credentials and policies without requiring the application to manage a mapping between the user identity and authority is of paramount importance in systems where it is necessary to protect users' anonymity (e.g., electronic voting systems).

Both PolicyMaker and KeyNote are based on *credentials* and *policies*, which are correctly referred to as *assertions*. More precisely, signed assertions are credentials, while policies are unsigned assertions. Credentials are mainly used for *trust delegation* under some specific conditions. Consequently a trusted entity can issue a credential to a nontrusted entity that becomes trusted and, in turn, can issue a similar credential to another entity and so on, thus forming a delegation chain without any length restrictions. An entity can therefore obtain access to a certain resource through a delegation from an authorized entity. Delegation is an important feature of a trust management system since it guarantees system scalability. Authority delegations can be graphically represented as a graph, where each node corresponds to a key and an edge from node n_1 to node n_2 indicates that there is a credential delegating authority from n_1 to n_2.

While credential statements are signed with the issuer's private key, policies, on the contrary, are not signed and the issuer entity is a standard entity, represented by keyword `Policy`, meaning that the specific assertion is locally trusted. An access request is accepted if there exists a path from some trusted node (with label `Policy`) to the node corresponding to the requester's key in the delegation graph. Another common characteristic of PolicyMaker and KeyNote is *monotonicity*, meaning that if an assertion is deleted, the set of privileges does not increase.

KeyNote is a trust management system where policies and credentials are expressed in a language directly managed by the KeyNote *compliance checker*. The main difference between PolicyMaker and KeyNote is that the latter directly performs signature verification inside the trust management engine, while PolicyMaker leaves this task up to the calling application. Moreover, as mentioned above, KeyNote defines a specific language for credentials, designed for request and policy evaluation, while PolicyMaker supports credentials written in any programming language. Also, PolicyMaker returns to the calling application a `True` of `False` answer, while KeyNote allows the definition of an application-dependent set of answers. Obviously, the syntax of assertions and requests is different between the two methods.

It is important to note that PolicyMaker and KeyNote do not directly enforce access control policies, but simply provide an advice to the calling application. The advice is based on the list of credentials and policies defined at application side. The calling application then decides whether to follow the received advice or not.

8.2.2 REFEREE

Rule-controlled environment for evaluation of rules, and everything else (REFEREE) [8] is a trust management system for Web applications. Like PolicyMaker, it supports full programmability of assertions (i.e., policies and credentials). The REFEREE system provides both a general policy-evaluation mechanism for Web clients and servers and a language used to define trust policies, putting all trust decisions under explicit policy control. More precisely, the REFEREE model imposes that every operation, including its policy evaluation and credential fetching mechanism, happens under the control of some policy. REFEREE is then a system for writing policies about policies, as well as policies about cryptographic keys, certification authorities, or trust delegation. A significant difference between REFEREE and PolicyMaker and KeyNote is that REFEREE supports non-monotonic assertions: policies and credentials may be used to express denial of specific actions. The three main primitive data types in REFEREE are *tri-values*, *statement lists*, and *programs*. The tri-values are true (accept), false (deny), and unknown (insufficient information). A statement list is a set of assertions expressed in two-element expressions. Both policies and credentials are programs that take a statement list and return a tri-value. An access request (query) to the REFEREE trust engine takes a policy name and additional arguments as input, including credentials or statement lists. REFEREE then downloads the relevant policies and executes them. The output is a tri-value and an optional statement list.

8.3 Credential-Based Trust Management Systems

While the approaches described in the previous section represent a significant step towards the support of access control in open environments, the assignment of authorizations to keys may result limiting, as the public key of a party can be seen as a pseudonym for the party. Therefore, an access control method granting or denying access to resources on the basis of requester's attributes would be advisable. In many situations, before an interaction can start, a certain level of trust is established through the exchange of information (credentials) between the interacting parties. However, the access control process should be able to operate without the requester's knowledge of the set of credentials she should have to access the resource. Consequently, the information about the needed credentials has to be communicated to the counterpart during the access control process itself. The access control decision is therefore obtained through a complex process and completing a service may require communicating information not related to the access itself, but related to additional restrictions on its execution, introducing possible forms of *trust negotiation*. Trust negotiation is an approach to *automated trust establishment*. Automated trust negotiation has gained much consideration in recent

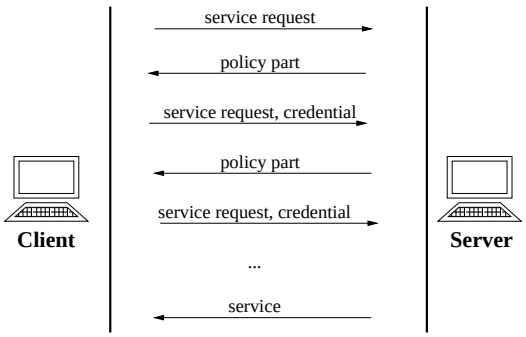

Fig. 8.1. Gradual trust establishment

years and various negotiation strategies have been proposed (see Sect. 8.3.2). In general, the interactions between a server and a client that need to establish a trust relationship can be summarized as follows.

- The client requests an access to a service.
- Upon receiving the request, the server checks if the client has provided the necessary credentials. In the case of a positive answer, the access to the service is granted; otherwise the server sends to the client the policies that she must fulfil to gain access.
- The client selects, if possible, the requested credentials and sends them to the server together with the service request.
- If the submitted credentials are appropriate, the user gains access to the service.

There are two major drawbacks to this protocol: a server has to disclose its, potentially sensitive policies to an unknown client, a client has to release all her relevant credentials in a single step without any possibility of negotiation.

A first improvement to reduce the release of irrelevant information during a trust establishment process consists in a *gradual trust establishment*. With a gradual trust establishment, upon receiving an access request, the server selects the policy that governs the access to the service and discloses only the information that it is willing to show to an unknown party. The client, according to her practices, decides if she is willing to disclose the requested credentials. Note that this incremental exchange of requests and credentials can be iteratively repeated as many times as necessary (see Fig. 8.1).
For the sake of simplicity, Fig. 8.1 shows a one-way protection schema, where the server controls access to some resources and communicates to the client the access control policies that the client should satisfy to gain the access. However, current approaches focus on a full negotiation process, where policies and credentials flow in both directions. In such a scenario, the server defines

policies that protect its sensitive resources and the client defines policies that restrict the disclosure of her credentials: both client and server can then require credentials the counterpart to release their sensitive information.

8.3.1 An Approach for Regulating Service Access and Information Disclosure

To address the aforementioned issues, new credential-based access control languages, models, and mechanisms have been developed. One of the first solution providing a uniform framework for credential-based access control specification and enforcement was presented by Bonatti and Samarati [6]. The framework includes an access control model, a language for expressing access and release policies, and a policy-filtering mechanism to identify the relevant policies for a negotiation. Access regulations are specified by mean of logical rules, where some predicates are explicitly identified. The system is composed of two entities: the *client* that requests access, and the *server* that exposes a set of services. Abstractions can be defined on services, grouping them in sets, called *classes*. Server and client interact by mean of a *negotiation process*, defined as the set of messages exchanges between them. Clients and servers have a *portfolio*, that is a collection of credentials (certified statements) and declarations (unsigned statements). A declaration is a statement issued by the party, while a credential is a statement issued and signed (i.e., certified) by authorities trusted for making the statements [11]. Credentials are essentially digital certificates, and must be unforgeable and verifiable through the issuing certificate authority's public key. In this proposal, credentials are therefore modeled as *credential expressions* of the form *credential_name(attribute_list)*, where *credential_name* is the attribute credential name and *attribute_list* is a possibly empty list of elements of the form *attribute_name=value_term*, where *value_term* is either a ground value or a variable. The main advantage of this proposal is that it provides an infrastructure to exchange the minimal set of certificates, that is, a client communicates the minimal set of certificates to a server, and the server releases the minimal set of conditions required for granting access. For this purpose, the server defines a set of *service accessibility rules*, representing the necessary and sufficient conditions for granting access to a resource. More precisely, this proposal distinguishes two kinds of service accessibility rules: *prerequisites* and *requisites*. Prerequisites are conditions that must be satisfied for a service request to be taken into consideration (they do not guarantee that it will be granted); requisites are conditions that allow the service request to be successfully granted. The basic motivation for this separation is to avoid unnecessary disclosure of information from both parties, and can therefore be seen as twofold: (i) server's privacy, and (ii) client's privacy. Therefore, the server will not disclose a requisite rule until after the client satisfies a corresponding prerequisite rule. Also, both clients and servers can specify a set of *portfolio disclosure rules*, used to define the conditions that govern the release of credentials and declarations.

The rules both in the service accessibility and portfolio disclosure sets are defined through a logic language that includes a set of predicates (listed in the following) whose meaning is expressed on the basis of the current *state*. The state indicates the parties' characteristics and the status of the current negotiation process, that is, the certificates already exchanged, the requests made by the two parties, and so on. Predicates evaluate both information stored at the site (persistent state) and acquired during the negotiation (negotiation state). Information related to a specific negotiation is deleted when the negotiation terminates. In contrast, persistent state includes information that spans different negotiations, such as user profiles maintained at Web sites. The basic predicates of the language can be summarized as follows.

- `credential(c, K)` evaluates to true if the current state contains certificate `c` verifiable using key `K`.
- `declaration(d)` evaluates to true if the current state contains declaration `d`, where `d` is of the form *attribute_name=value_term*.
- `cert-authority(CA,` $\mathtt{K_{CA}}$`)` evaluates to true if the party using it in her policy trusts certificates issued by certificate authority `CA`, whose public key is $\mathtt{K_{CA}}$.
- A set of non-predefined predicates necessary for evaluating the current *state* values. These predicates can evaluate both persistent and negotiation states, and they are defined by each of the parties interacting.
- A set of non-predefined *abbreviation* predicates that are used to abbreviate requirements in the negotiation phase.
- A set of standard mathematical built-in predicates, such as $=$, $\neq$, and $\leq$.

The rules, both for service accessibility and portfolio disclosure, are composed of two elements: the *body*, containing a boolean expression composing, through boolean operators `and`, `or`, `not`, the aforementioned predicates; and the *head*, specifying the services accessible, or the certificates releasable, according to the rule. Figure 8.2 illustrates the following client/server interaction.

- the client sends a request for a service to the server;
- the server asks from the client a set of prerequisites, that is, a set of necessary conditions for granting access;
- the client sends back the required prerequisites;
- if the prerequisites are sufficient, than the server identifies the credentials and declarations needed to grant access to the resource;
- the client evaluates the requests against its portfolio release rules and makes, eventually, some counter-requests;
- the server sends back to the client the required certificates and declarations;
- the client fulfils the server's requests;
- the service is then granted to the client.

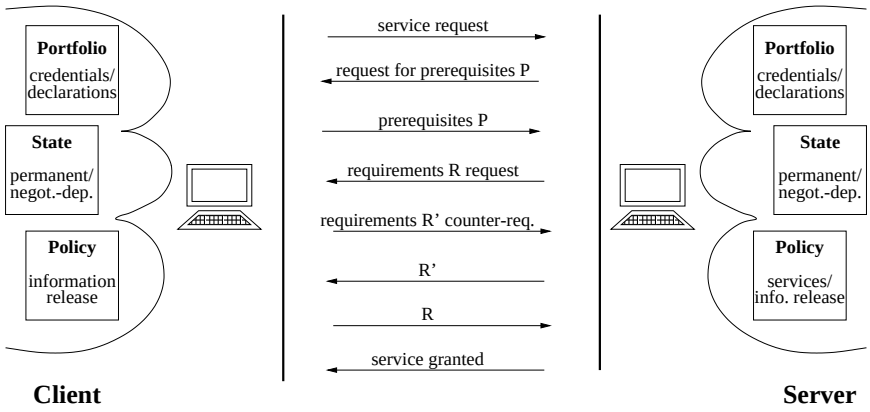

Fig. 8.2. Client server negotiation

Since there may exist different policy combinations that may bring the access request to satisfaction, the communication of credentials and/or declarations could be an expensive task. To overcome this issue, the *abbreviation* predicates are used to abbreviate requests. Besides the necessity of abbreviations, it is also necessary for the server, before releasing rules to the client, to evaluate state predicates that involve private information. For instance, the client is not expected to be asked many times the same information during the same session and if the server has to evaluate if the client is considered not trusted, it cannot communicate this request to the client itself.

Communication of requisites to be satisfied by the requester is then based on a filtering and renaming process applied on the server's policy, which exploits partial evaluation techniques in logic programs [6, 17]. Access is then granted whenever a user satisfies the requirements specified by the filtering rules calculated by means of the original policy and the already released information.

8.3.2 Negotiation Strategies

Besides solutions for uniform frameworks supporting credential-based access control policies [6], different automated trust negotiation proposals have been developed. Trust negotiation occurs whenever credentials themselves carry some sensitive information. In such a situation, a procedure needs to be applied to establish trust through negotiation. Trust is then established gradually by disclosing credentials and requests for credentials. It is however important to note that different parties may have different requirements for how such a negotiation process should be performed and each party can therefore rely on its *trust negotiation strategy*. We now provide a brief description of some negotiation strategies suitable for different scenarios.

In [31] the *prudent negotiation strategy* (PRUNES) has been presented. This strategy ensures that the client communicates her credentials to the server only if access will be granted and the set of certificates communicated to the server is the minimal necessary for granting it. Each party defines a set of *credential policies* that regulates how and under which conditions the party releases its credentials. The negotiation is then a series of requests for credentials and counter-requests on the basis of the parties' credential policies. The established credential policies can be graphically represented through a tree, called a *negotiation search tree*, composed of two kinds of nodes: *credential nodes*, representing the need for a specific credential, and *disjunctive nodes*, representing the logic operators connecting the conditions for credential release. The root of a tree node is a service (i.e., the resource the client wants to access). The negotiation can therefore be seen as a backtracking operation on the tree. The backtracking can be executed according to different strategies. For instance, a *brute-force* backtracking is complete and correct, but it is too expensive to be used in a real scenario. The authors therefore proposed the *PRUNES* method, which prunes the search tree without compromising completeness or correctness of the negotiation process. The basic idea is that if a credential C has just been evaluated and the state of the system is not changed too much, then it is useless to evaluate again the same credential, as the result will be exactly the same as the result previously computed.

In [22] different negotiation strategies are introduced together with the concepts of *safeness* and *completeness*. A strategy is *safe* if all possible negotiations conducted by the parties are safe and hence there exists a sequence of resource disclosures that culminates in the sensitive resource disclosure. A strategy is *complete* if, whenever there exists a safe sequence of disclosure, the original requested resource is released. Negotiation strategies can be divided between *eager* and *parsimonious* credential release strategies. Parties applying the first strategy turn over all their credentials if the disclosure is safe. The eager approach releases credentials as soon as possible, minimizing the time requested for negotiation but increasing the amount of released data. A *naive eager approach* requires the parties to send each other all the credentials for which an authorized path has been found, without the need to distinguish between credentials needed to take the decision and credentials not relevant for the negotiation. The major advantage of this strategy is that there is no need for policy disclosure. On the other side, a great amount of unmotivated disclosure of data is performed. According to a *parsimonious* strategy, the parties delay as much as possible data disclosure until the negotiation reaches a certain state. In addition, parties applying a parsimonious strategy only release credentials upon explicit request by the server (avoiding unnecessary releases).

In [33] a large set of negotiation strategies, called a *disclosure tree strategy* (DTS) family, has been defined. The authors show that, if two parties use different strategies from the DST family, they are able to negotiate trust. The

DTS family is a closed set, that is, if a negotiation strategy can interoperate with any DST strategy, it must also be a member of the DST family.

In [32] a *unified schema for resource protection* (UniPro) was proposed. This mechanism is used to protect the information specified within policies. UniPro gives (opaque) names to policies and allows any named policy P_1 to have its own policy P_2, meaning that the content of P_1 can only be disclosed to parties who have shown that they satisfy P_2.

Another solution for implementing access control based on credentials is the *adaptive trust negotiation and access control* (ATNAC) approach [21]. This method grants or denies access to a resource on the basis of a *suspicion level* associated with subjects. The suspicion level is not fixed but may vary on the basis of the probability that the user has malicious intent. In [26] the authors propose to apply the automated trust negotiation technology for enabling secure transactions between portable devices that have no pre-existing relationship.

8.4 Reputation-Based Trust Management Systems

Related to trust is the concept of *reputation*. Reputation is another popular mechanism that people employ to deal with unknown parties. Reputation-based solutions do not require any prior experience with the party for reputation to be used to infer trustworthiness. It is then suitable for establishing initial trust. Parties in such systems establish trust relationships with other parties and assign trust values to these relationships. Generally, a trust value assigned to a trust relationship is a function of the combination of the party's global reputation and the evaluating party's perception of that party. There is however a clear distinction between *trust* and *reputation*: a trust value T can be computed based on its reputation R, that is, $T = \phi(R, t)$, where t is the time elapsed since the reputation was last modified [2]. Traditionally, research approaches [3, 14] distinguish between two main types of reputation-based trust management systems, namely *centralized reputation systems* and *distributed reputation systems*. In centralized reputation systems, trust information is collected from members of the community in the form of *ratings* on resources. The central authority collects all the ratings and derives a score for each resource. In a distributed reputation system there is not a central location for submitting ratings and obtaining resources' reputation scores; instead, there are distributed stores where ratings can be submitted. Recently, reputation-based trust management systems have been applied in many different contexts such as peer-to-peer (P2P) networks, where the development of P2P systems largely depends on the availability of novel provisions for ensuring that peers obtain reliable information on the quality of the resources they are retrieving [19]. Reputation models allow the expression and reasoning about *trust* in a peer based on its past behavior [20] and interactions other peers have experienced with it. The proposed approaches use different tech-

niques for combining and propagating the ratings [1, 9, 12, 27, 30]. Here we describe a few related examples. In [1] a trust model is proposed, where, after each transaction, and only in the case of malicious behaviour, peers may file a complaint. Before engaging in an interaction with others, peers can query the network about existing complaints on their counterparts. One limitation of this model is that it is based on a binary trust scale (i.e., an entity is either trustworthy or not). Hence, once there is a complaint filed against a peer p, p is considered untrustworthy even though it has been trustworthy for all previous transactions. In [27] a Bayesian network-based trust model is proposed, where peers are evaluated with respect to different capabilities (e.g., capability to provide music files or movies). Basically, peers develop a naive Bayesian network for each peer with which they have interacted and modify their corresponding Bayesian networks after each interaction. When a peer has no experience with another one, it can ask other peers to make recommendations for it. Such recommendations are partitioned into two groups, recommendations from trustworthy peers and recommendation from unknown peers, and are combined by taking a weighted sum. In [30] an adaptive reputation-based trust model for P2P electronic communities is presented. It is based on five trust parameters: feedbacks, number of transactions, credibility of feedbacks, a transaction context factor, and a community context factor. The trust value associated with a peer is then defined as a weighted sum of two parts. The first part is the average amount of credible satisfaction a peer receives for each transaction. The second part increase or decrease the trust value according to community-specific characteristics or situations (e.g., the number of files a peer shares can be seen as a type of community context factor that has to be taken into consideration when evaluating the trustworthiness of a peer).

P2PRep is an example of a reputation-based protocol, formalizing the way each peer stores and shares with the community the reputation of other peers [9]. P2PRep runs in a fully anonymous P2P environment, where peers are identified using self-assigned *opaque identifiers* (e.g., a digest of a public key for which only the peer itself knows the corresponding private key). For simplicity, reputation and trust are represented as fuzzy values in the interval $[0, 1]$. This approach can however be readily extended to more-complex array-based representations taking into account multiple features [2]. The P2PRep protocol consists of five phases. In phase 1, a requester r locates available resources sending a `Query` broadcast message. Other peers answer with a `QueryHit` message notifying r that they may provide the requested resource. Upon receiving a set of `QueryHit` messages, r selects an offerer o and, in phase 2, r polls the community for any available reputation information on o, sending a `Poll` message. `Poll` messages are broadcasted in the same way as `Query` messages. All peers maintain an *experience repository* of their previous experiences with other peers. When a peer receives a `Poll` message, it checks its local repository. If it has some information to offer and wants to express an opinion on the selected offerer o, it generates a vote based on its experiences, and returns a `PollReply` message to the initiator r. As a result

of phase 2, r receives a set V of votes, some of which express a good opinion while others express a bad one. In Phase 3, r evaluates the votes to collapse any set of votes that may belong to a clique and explicitly selects a random set of votes for verifying their trustworthiness [9]. In phase 4 the set of reputations collected in phase 3 is synthesized into an aggregated community-wide reputation value. Based on this reputation value, the requester r can take a decision on whether to access the resource offered by o or not (phase 5). After accessing the resource, r can update its local trust on o (depending on whether the downloaded resource was satisfactory or not). While a naive implementation of P2PRep can be expensive in terms of storage capacity and bandwidth, this cost can be minimized by applying simple heuristics. The amount of storage capacity is proportional to the number of peers with which the initiator has interacted. With respect to the bandwidth, it is easy to see that P2PRep increases the traffic of the P2P network by requiring both direct exchanges and broadcast requests. It is, however, reasonable to assume that the major impact of the protocol on network performance is due to broadcast messages and their answers. To overcome this issue, several heuristics can be applied. For instance, *intelligent routing* techniques can be applied for enabling custom forwarding of poll packets to the right peers. Vote caching is another technique that can be applied to improve the effectiveness of P2PRep. Finally, P2PRep scalability depends on the technique used for vote aggregation.

8.5 Conclusions

We have presented an overview of existing and proposed approaches to trust management, clarifying the link between trust and reputation. We have analyzed the current trends and developments in this area, and described some recent approaches for trust management based on a more sophisticated notion of credential-based language and negotiation to establish trust between unknown parties.

8.6 Acknowledgements

This work was supported in part by the European Union within the PRIME project in the FP6/IST programme under contract IST-2002-507591 and by the Italian MIUR programme within the KIWI and MAPS projects.

References

1. K. Aberer, Z. Despotovic (2001). Managing trust in a peer-2-peer information system. In Proc. of the Tenth International Conference on Information and Knowledge Management (CIKM 2001), Atlanta, Georgia.

2. R. Aringhieri, E. Damiani, S. De Capitani di Vimercati, S. Paraboschi, P. Samarati (2006). Fuzzy techniques for trust and reputation management in anonymous peer-to-peer systems. Journal of the American Society for Information Science and Technology (JASIST), 57(4):528–537.
3. M. Blaze, J. Feigenbaum, J. Ioannidis, A.D. Keromytis (1999). The role of trust management in distributed systems security. Secure Internet Programming, pp. 79–97.
4. M. Blaze, J. Feigenbaum, J. Ioannidis, A.D. Keromytis (1999). The KeyNote Trust Management System (Version 2), Internet RFC 2704 edition.
5. M. Blaze, J. Feigenbaum, J. Lacy (1996). Decentralized trust management. In Proc. of the 17th Symposium on Security and Privacy, Oakland, California, USA.
6. P. Bonatti, P. Samarati (2002). A unified framework for regulating access and information release on the web. Journal of Computer Security, 10(3):241–272.
7. CCITT (Consultative Committee on International Telegraphy and Telephony) (1988). Recommendation $X.509$: The Directory—Authentication Framework.
8. Y. Chu, J. Feigenbaum, B. LaMacchia, P. Resnick, M. Strauss (1997). REFEREE: Trust management for web applications. The World Wide Web Journal, 2(3):127–139.
9. E. Damiani, S. De Capitani di Vimercati, S. Paraboschi, P. Samarati (2003). Managing and sharing servents' reputations in p2p systems. IEEE Transactions on Data and Knowledge Engineering, 15(4):840–854.
10. C.M. Ellison, B. Frantz, B. Lampson, R.L. Rivest, B.M. Thomas, T. Ylonen (1999). SPKI certificate theory. RFC 2693.
11. B. Gladman, C. Ellison, N. Bohm (1999). Digital signatures, certificates and electronic commerce. http://ya.com/bg/digsig.pdf.
12. M. Gupta, O. Judge, M. Ammar (2003). A reputation system for peer-to-peer networks. In Proc. of the ACM 13th International Workshop on Network and Operating Systems Support for Digital Audio and Video, Monterey, California, USA.
13. K. Irwin, T. Yu (2005). Preventing attribute information leakage in automated trust negotiation. In Proc. of the 12th ACM Conference on Computer and Communications Security, Alexandria, VA, USA.
14. A. Jøsang (1996). The right type of trust for distributed systems. In Proc. of the 1996 Workshop on New Security Paradigms, Lake Arrowhead, CA.
15. N. Li, J.C. Mitchell, W.H. Winsborough (2005). Beyond proof-of-compliance: Security analysis in trust management. Journal of the ACM, 52(3):474–514.
16. N. Li, W.H. Winsborough, J.C. Mitchell (2003). Distributed credential chain discovery in trust management. Journal of Computer Security, 11(1):35–86.
17. M. Minoux (1988). LTUR: A Simplified Linear-Time Unit Resolution Algorithm for Horn Formulae and Computer Implementation. Inf. Process. Lett., 29(1):1–12.
18. J. Ni, N. Li, W.H. Winsborough (2005). Automated trust negotiation using cryptographic credentials. In Proc. of the 12th ACM Conference on Computer and Communications Security, Alexandria, VA, USA.
19. A. Oram edt. (2001). Peer-to-Peer: Harnessing the Power of Disruptive Technologies. O'Reilly & Associates.
20. P. Resnick, R. Zeckhauser, E. Friedman, K. Kuwabara (2000). Reputation systems. Communications of the ACM, 43(12):45–48.

21. T. Ryutov, L. Zhou, C. Neuman, T. Leithead, K.E. Seamons (2005). Adaptive trust negotiation and access control. In Proc. of the 10th ACM Symposium on Access Control Models and Technologies, Stockholm, Sweden.
22. K. Seamons, M. Winslett, T. Yu (2001). Limiting the disclosure of access control policies during automated trust negotiation. In Proc. of the Network and Distributed System Security Symposium (NDSS 2001), San Diego, CA, USA.
23. K.E. Seamons, W. Winsborough, M. Winslett (1997). Internet credential acceptance policies. In Proc. of the Workshop on Logic Programming for Internet Applications, Leuven, Belgium.
24. K.E. Seamons, M. Winslett, T. Yu, B. Smith, E. Child, J. Jacobson, H. Mills, L. Yu (2002). Requirements for policy languages for trust negotiation. In Proc. of the 3rd International Workshop on Policies for Distributed Systems and Networks (POLICY 2002), Monterey, CA.
25. Security and trust management (2005). http://www.ercim.org/publication/Ercim_News/enw63/.
26. T.W. van der Horst, T. Sundelin, K.E. Seamons, C.D. Knutson (2004). Mobile trust negotiation: Authentication and authorization in dynamic mobile networks. In Proc. of the Eighth IFIP Conference on Communications and Multimedia Security, Lake Windermere, England.
27. Y. Wang, J. Vassileva (2003). Trust and reputation model in peer-to-peer networks. In Proc. of the Third International Conference on Peer-to-Peer Computing, Linköping, Sweden.
28. L. Wang, D. Wijesekera, S. Jajodia (2004). A logic-based framework for attribute based access control. In Proc. of the 2004 ACM Workshop on Formal Methods in Security Engineering, Washington DC, USA.
29. M. Winslett, N. Ching, V. Jones, I. Slepchin (1997). Assuring security and privacy for digital library transactions on the web: Client and server security policies. In Proc. of the ADL '97 – Forum on Research and Tech. Advances in Digital Libraries, Washington, DC.
30. L. Xiong, L. Liu (2003). A reputation-based trust model for peer-to-peer ecommerce communities. In Proc. of the IEEE International Conference on E-Commerce, Newport Beach, California.
31. T. Yu, X. Ma, M. Winslett (2000). An efficient complete strategy for automated trust negotiation over the internet. In Proc. of the 7th ACM Computer and Communication Security, Athens, Greece.
32. T. Yu, M. Winslett (2003). A unified scheme for resource protection in automated trust negotiation. In Proc. of the IEEE Symposium on Security and Privacy, Berkeley, California.
33. T. Yu, M. Winslett, K.E. Seamons (2001). Interoperable strategies in automated trust negotiation. In Proc. of the 8th ACM Conference on Computer and Communications Security, Philadelphia, Pennsylvania.
34. T. Yu, M. Winslett, K.E. Seamons (2003). Supporting structured credentials and sensitive policies trough interoperable strategies for automated trust. ACM Transactions on Information and System Security (TISSEC), 6(1):1–42.

Index