A Note About the Semantics of Delegation

B. Crispo[1] and B. Christianson[2]

[1] University of Cambridge, England {bc201@cl.cam.ac.uk}
[2] University of Hertfordshire, England {B.Christianson@herts.ac.uk}

Abstract. In many applications, mobile agents are used by a client to delegate a task. This task is usually performed by the agent on behalf of the client, by visiting various service provider's sites distributed over a network. This use of mobile agents raises many interesting security issues concerned with the trust relationships established through delegation mechanisms between client and agent, agent and service provider and client and service provider. In this paper we will explain why the traditional semantics of delegation used by existing access control mechanisms, either centralised or distributed, are generally not satisfactory to prevent and detect deception and why these problems are even more critical when these semantics are used in mobile agent paradigms.
In this paper we will point out the potential problems that may raise when existing semantics of delegation are used by agents and we will show that these problems depend on which trust relationships among the entities of the system are assumed.

1 Access Control Mechanisms

As soon as services and resources are shared among different users, there is the need for access control policies in order to protect, account and audit the use of these resources. Access control and the related protection mechanisms to enforce it, have historically been an important area of research for operating system designers. One of the basic mechanisms supported by almost all the access control mechanisms is delegation. Delegation is an essential feature particularly in distributed systems where the resources needed by a single user are rarely all local to the machine where the user is logged on, thus the need to remotely access these resources requires delegation mechanisms. Moreover, the increasing complexity of tasks and activities performed by each individual user through the use of computers, makes it practical to have the possibility to delegate some of these tasks and functions to other entities of the system.

Mobile agents are the most recent and in some respects most sophisticated technology that can support applications to answer to these needs. Mobile agents are often used as tools to support users, by carrying out a task on the users behalf. Usually this task requires navigation through several different sites distributed over a wide area network and additionally it often requires to invoke some operations on these hosting sites and to eventually store and return the output of these operations to the user to whom agents belong.

A typical example may be the following: let us suppose we have a cheapest-flights-finder service that offers the service of finding the cheapest fare and to book the requested flight on behalf of the customer and offers to pay back the customer if she succeeds to find, the same flight, at a cheaper fare. The flights-finder asks for an annual subscription for such a service. Let us suppose we have a big company that usually needs to arrange a lot of business flights every year. The company delegates this task to the described cheapest-flights-finder service. Delegation is necessary because the request is done by the flights-finder on behalf of the company's employee to whom the ticket will be issued. The airline, chosen by the agent, sells the ticket by issuing the voucher and sending the payment receipt to the company's account office.

The delegation mechanism used in this example must be able to address several security concerns. Because of the pay-back option the actions performed by the company (grantor) and by the flights-finder (grantee) must be distinguishable by an arbitrator in case of a dispute. Furthermore, it must also be impossible, or at least computationally infeasible, for the grantor(grantee) to forge without detection operation invocations that use the delegated rights that belong to the grantee(grantor). In other words must be possible to detect if the grantor attempts to fraudulently masquerade as the grantee and vice versa. The grantor should not be able to forge the booking request, otherwise she could always choose the second cheapest option and then claim her money back from the flights-finder service. We claim that the existing delegation mechanisms fail to prevent or detect the frauds just described and in the rest of the paper we explain in detail, why.

2 Notation

We introduce the notation the we will use through this paper. With the term *principal* we denote a generic entity of the system (i.e., program, person, agent, server, etc.). *Grantor* denotes the principal that delegate and *grantee* the principal that has been delegated. The end-point denotes the principal that provide the service that will be requested by using delegation. The end-point is the principal that finally will receive a request of service sent by the grantee on behalf of her grantor.

We will use the letter K to denote a cryptographic key. In this paper, we will mention only public key cryptography *(PK)*. PK assumes that each principal possesses a unique cryptographic key pair: a *public* and a *private* key, denoted respectively by K^+ and K^-. Public keys are public and every principal knows for each principal which is the correspondent public key. This knowledge is guaranteed by certification that here we assume already implemented. The owner of each key is denoted by a subscript of her name (i.e., K_{Alice} denotes Alice's key). K^+_{Alice} denotes Alice's public key and K^-_{Alice} denotes Alice's private key.

Each key pair is used by its owner to sign and verify messages. Let us suppose *SIG()* and *VER()* to be the signing and the verifying primitives used by the public key system. Then C=SIG(M,K^-_{Alice}) is the digital signature of Alice over the message M generated by using the private key K^{-}_{Alice}. The signature C can be verified by calculating:

VER(C,K^{+}_{Alice})=VER(SIG(M,K^{-}_{Alice}),K^{+}_{Alice})

by any principal of the system that eventually receives C and knows Alice's public key. The underlying cryptographic algorithm guarantees that the digital signature C is unique and can be generated only by using K^{-}_{Alice} and that is computationally infeasible to forge without detection the digital signature C for the message M by knowing only M, C and the public key K^{+}_{Alice}. Of course the system also assumes that the private key is never disclosed or shared by the owner with anybody else. Finally $''$,$''$ denotes concatenation.

3 Delegation of Rights

In this section we will show that most of the existing delegation mechanisms, such as the ones introduced by Gasser and McDermott [4], Varadharajan *et al.* [10], Neuman [8], and by Abadi *et al.* [1, 5] do not consider these possibilities of deceptions and frauds if perpetrate by the grantor. Delegation of rights is defined as

> *the process whereby a principal authorises an agent to act on her behalf, by transferring a set of her rights to the agent, possibly for a specific period of time.*

where the principal is used to denote a generic entity of the system. This semantic for delegation allows the transfer of rights but at the same time it assumes that the responsibility attached to this rights are always shared or retained by the grantor. When we say that the principal A is *responsible* for a particular action X, we mean that A is the principal that is the most likely to have performed technically the action X. Our definition of responsibility is for auditing purposes and not for legal ones, because legal issues cannot be solved entirely only by technical means but usually need also the support of other, more conventional, forms of evidence (e.g., in paper form). By 'most likely' we mean that an unbiased external observer that reviews the history of the system through the audit files after the fact will infer that A has performed X.

The delegation of rights semantics given above, assumes that the grantor is trusted not to abuse the capability that she always keeps to exercise the rights masquerading as the grantee. The rights are not given away by the grantor to the grantee but rather they are shared between the two. This strong trust assumptions on which most of the existing mechanisms rely, can be better understood by analysing the way in which delegation of rights is implemented.

The delegation of rights from the grantor to the grantee is performed by handing-off a credential, called a *delegation token*, whose integrity and possibly secrecy are assured by cryptographic tools. Digital signatures are used to provide authenticity and integrity and encryption to provide secrecy if it is required. The delegation token specifies grantor, delegation key, rights that are delegated to the grantee and possibly the validity period of the token.

$SIG(($Grantor, $delegation\_key^{+}$, Rights, Validity Period), $K^{-}_{Grantor})$

The $delegation\_key^{+}$ is the public key of a key pair whose private key must be used by the grantee to exercise the right specified in the token. Thus when the flight-finder agent of our toy example, acting as a grantee on behalf of the company will, for example, visit an airline, it will query the airline's database by presenting a request signed by using the delegation private key followed by the delegation token she possess as proof that the agent is delegated by the company to do so.

$SIG(($Grantee, Request), $delegation\_key^{-})$, $SIG(($Grantor, $delegation\_key^{+}$, Rights, Validity Period), $K^{-}_{Grantor})$

The main problem with delegation of rights is that the key pair used as delegation key pair is generated by the grantor. Then the public key is passed to the grantee in the delegation token while the private key is passed by the grantor to the grantee by mean of a secure channel established somehow between grantor and grantee. A channel is defined as secure if it is secure against passive and active attacks, thus the information sent through this channel is guaranteed to be genuine and confidential. This means that the grantor can always generate the above request and then falsely claim that it was generated by the grantee instead. Thus in our example the company can forge booking requests to the most expansive airline as if they were sent by the agent and then claim the money back to the flight-finder service on the basis of their agreement. Thus the particular trust assumptions made by delegation of rights make it impossible to build auditing mechanisms where is possible irrefutably to distinguish if an action was really performed by the grantee or performed by the grantor but recorded in the audit file as if it was performed by the grantee.

4 Delegation of Responsibility

Delegation of rights assumes that the grantor never cheats on the grantee. Deceptions and frauds are simply not considered in the threat model envisaged by the existing mechanisms, that all seem to assume that the attacker is outside the system. Tracing clear boundaries in actual distributed open systems where agents can cross different domains, is certainly a difficult if not impossible task.

The threat model that applications employing mobile agent technology have to consider, particularly in commercial or financial environments should not assume that a particular entity of the system must be trusted *a priori*, but rather should start by applying the *principle of the least trust* that says that every entity of the system may have a reason to lie or misbehave, thus the security mechanisms must consider this threat.

For this reason we introduce a new semantic of delegation that prevent the kind of attacks discussed in the previous section.

We call this new type of delegation, delegation of responsibility defined as:

> *the process whereby a principal authorises an agent to act on her behalf, possibly for a specific period of time, during which it is always possible to distinguish whether a particular action, among those delegated, was performed by the principal or by the agent acting on her behalf.*

With delegation of responsibility it is always possible to distinguish beyond reasonable doubt if a request was performed by the grantor or by the grantee, because they have no capability to forge each other requests. We will explain more in detail this claim showing the protocol that we use to implement this new semantic of delegation.

We introduce an high level description of the protocol that is used to delegate a task, Ω from the grantor to the grantee. Let us suppose that Ω is the task of updating the database D, physically stored and maintained at the remote end-point X.

Grantor $\longrightarrow$ Grantee:	I Grantor, wish to delegate you grantee, task Ω. Please let me know which key will you use to perform Ω on my behalf.
Grantee $\longrightarrow$ Grantor:	I grantee, will use the private key which signatures can be verified by $delegation\_key^{+}$
Grantor $\longrightarrow$ Grantee:	A delegation token T, containing Grantor and Grantee's names, $delegation\_key^{+}$, Ω and the validity period, signed by the Grantor stating that she delegates the Grantee to perform Ω on her behalf

Upon the successful termination of the above protocol, the grantee can perform the task of updating D on behalf of the grantor by signing with his $delegation\_key^{-}$ the requests to do so to the end-point X.

Grantee $\longrightarrow$ X: $SIG(($Grantee, Update D$), delegation_key^{-})$, T

T serves as a proof to the end-point X, that the grantee was authorised by the grantor to perform the updating of D. Because the delegation secret key is chosen by the grantee and its knowledge and/or use is never shared with the grantor (or anybody else), the above request, for the assumption made by public key systems, can only be generated by the grantee. Thus the frauds that are possible with delegation of rights are not possible anymore with delegation of responsibility.

5 Discussion

In this section we focus on the differences between the way in which delegation is used in traditional distributed systems (i.e., DSSA described in [3]) and in applications that employ autonomous agents. These differences emphasise why with agents is crucial to design delegation mechanisms that minimise the assumptions of trust.

The are mainly two reasons why people need delegation:

- Because the number of tasks that they need to perform personally is so high that they find easier to delegate some of these tasks to other people they choose
- Because they do not have the competence to perform a task by themselves but nevertheless they need the execution of this task. Thus they delegate this execution to a person or a service that has the necessary competence

Most of the literature in the security area has focussed on the first reason. Implicitly in the existing mechanisms that implement delegation of rights, is assumed that the grantor knows personally, because of an already existing relationship, the grantee. Besides in many cases the grantor is in a position of power over the grantee (i.e., manager and secretary), this also allows the grantor to eventually undo an operation if she is not satisfied by the way in which her grantee executed it. Finally most of the time this relationship is within a well defined organisation. These assumptions are reflected in the threats model adopted by these delegation mechanisms that consider only attacks that can be posed by outsiders of the organisation, while they consider the members of the organisation, grantors and grantees and end-points, indistinguishably all trusted.

With agents and even more with agencies [6, 7], people will delegate motivated by the second reason at least as often as for the first one. Thus both reasons must be considered in the threat model. When agents are used to supply the competence that users need but do not have, usually the grantor does not have previous direct relationships with the grantee besides grantor and grantee usually do not belong to the same organisation. So the grantee is chosen on the basis of reputation, brand name, recommendation of a friend, but not because of direct trust. The grantor is likely to be in a pair relationship with the grantee and not in a position of power over the grantee, and this make much more difficult to undo grantee's action if the grantor is not satisfied. All these reasons make now unaccettable to trust *a priori* all grantors and grantees of the system. Even defining who is inside and who outside the system becames difficult.

Most of the existing delegation mechanisms have been designed for distributed systems typically composed by many general-purpose workstations distributed physically over a local or wide area network and on which users could login to the system, and by a small number of special servers (i.e., a database), colour printers, specialised and expensive hardware. Because this second kind of components of the system were quite expensive they were not available locally to each node of the system but they were located only in one or few nodes and accessed by any user remotely. With these architectures, if say a user needs to search for some information in a database maintained centrally on a remote server, she can delegate the remote database server to do the search on her behalf and then receives the result back from the server. In the above systems, grantors and grantees reside on the same system even if in a distributed fashion. Furthermore usually the strategy used to perform these services once delegated is well known to both grantors and grantees (i.e., the scheduling policy of the printer) and the grantor may restrict the set of nodes the grantee is allowed to visit, in performing the delegated task, because they are the only ones the grantors knows and trusts in some way (i.e., they all are in a particular domain).

With agents, especially with intelligent ones [9, 2], the conditions are often very different. Usually grantors(clients) do not know in advance (and possibly neither afterwards) which strategy is adopted by the grantee(agent) to execute the task she has delegated to him. When the strategy is unknown to the grantor(client), grantees(agents) may independently choose through which sites to "migrate" to achieve their goals on behalf of their grantors, The consequent lack of transparency that this use of agents causes make it even more crucial the need of delegation mechanisms that do not assume any pre-existing trust relationship between grantors and grantee in order to avoid the possible attacks described in the previous sections.

The environments and the type of applications in which agents are commonly used, strongly motivate, even more than other paradigms, the necessity of delegation mechanisms, as the one we introduced, secure against deceptions and frauds attempted possibly by any entity of the system.

6 Conclusion

In this paper we have analysed the security of the existing delegation mechanisms, all of which allow to delegate rights but not the responsibility attached to these rights. We have also described why this mechanisms fail to prevent some class of frauds, typically the ones attempted by the grantor. We have then introduced a new semantics of delegation and a protocol to implement it. Our solution allows to prevent this class of frauds because the cryptographic key used to exercise the delegated task is generated, known and used only by the grantee and never by the grantor. We described the principles that are at the basis of our protocol. Many other details must be considered before actually being able to implement the protocol. Issues as authentication of principals and protection against replay attacks or the man-in-the-middle attacks has been voluntarily left outside the scope of this paper because not essential for the sake of our arguments . We have finally analysed the dependencies that exist between system architecture and trust relationships and how they influence the threat model that must be considered by delegation mechanisms. We have also pointed out that application using agents introduce a new class of security challenges that must be considered and addressed during the design of new delegation protocols.

References

1. M. Abadi, M. Burrows, B.W. Lampson, and G. Plotkin. A Calculus for Access Control in Distibuted Systems. *ACM Transaction on Programming Languages and Systems*, (15):706–734, September 1993.
2. M.A. Boden. Agents and Creativity. *Communications of ACM*, 37(7):117–121, 1994.
3. M. Gasser, A. Goldstein, C. Kaufman, and B.W. Lampson. The Digital Distributed System Security Architecture. In *Proc. Of the 1989 National Computer Security Conference*, pages 305–319, October 1989.
4. M. Gasser and E. McDermott. An Architecture for Practical Delegation in a Distributed System. In *Proceedings of the IEEE Symposium on Security and Privacy*, 1990.
5. B.W. Lampson, M. Abadi, M. Burrows, and E. Wobber. Authentication in Distributed System: Theory and Practice. *ACM Transaction on Computer Systems*, (10):265–310, November 1992.
6. M. Minsky. *The Society of Mind.* New York, NY; Simon and Schuster., 1985.
7. M. Minsky. A Conversation with Marvin Minsky about Agents. *Communications of ACM*, 37(7):23–29, 1994.
8. B.C. Neuman. Proxy-Based Authorization and Accounting for Distributed System. In *Proceedings of the 13th International Conference on Distributed Systems*, May 1993.
9. D. Riecken. M: An Architecture of Integrated Agents. *Communications of ACM*, 37(7):107–116, 1994.
10. V.Varadharajan, P. Allen, and S. Black. An Analysis of the Proxy Problem in Distributed System. In *Proceedings of the IEEE Symposium on Security and Privacy*, 1991.