Delegation of Tasks and Rights

Gerald Vogt
Munich Network Management Team – University of Technology Munich
Oettingenstr. 67, 80538 Munich, Germany
gvogt@acm.org

The delegation of management tasks is usually accompanied with some transfer of rights. However, in many existing systems it is not possible to link this transfer with the execution of the task as the transfer usually happens in a separate independent step. Moreover, it is often not possible to control the proper use of the transferred rights. This makes it very vulnerable to abuse. In this paper we propose an approach for delegation based on mobile agents that allows control and supervision of the delegated tasks on a technical level. It will be the basis for an implementation in the Mobile Agent System Architecture (MASA).

Keywords: Security Management, Delegation of Tasks, Authorization of Tasks

1 Introduction

Delegation as concept is very common. It happens everywhere and all the time: Whenever work is passed to subordinates, there is delegation of tasks. Certainly, the superior still has his own responsibility to supervise the correct and proper execution of the delegated task. But this is not sufficient. Delegation of a task also includes – often implicitly – a delegation of obligation (to do the task), responsibility (to do the task properly) and authority (to be allowed to do the task). Particularly the necessary authorization to do the task often makes problems. It is often impossible to give authorization without giving the ability to exploit this authorization to do something else.

Delegation of IT management tasks often has severe consequences in regard to possible security threats. For example, delegating software installation often means to give administrative privileges to somebody. This, however, can be exploited in many ways, e.g., to read arbitrary files or to install a "backdoor" program for later access.

This is due to the fact that commonly used security models have a subject-object view [8]. They only consider the access relation of a subject to an object. They ignore the context of an actual access, e.g., the scope of the task in which this operation is performed. This separation between assignment and usage of rights makes it impossible to grant rights for a specific delegated task or to control its use in the context of the delegation. Moreover, to detect abuse of delegated rights, it is still necessary to supervise them independently and to revoke them as fast as possible if there is a "misbehavior".

As security models are not task-oriented, the granularity of the access rights usually can not cope with the specific requirements of the delegated task. If the granularity is coarse exploitation of rights is easy because the delegated rights allow operations that are not required for the task. If the granularity is very fine, the overhead to pass rights can be enormous and may require a lot of (re-)configuration just for a simple task. Particularly for management tasks that require access across different domains, e.g., in provider hierarchies or in a end-to-end scenario, this is often a limiting factor as any right passed to a different domain should always be as restricted as possible to protect the local management system.

The main requirements for a system that overcomes these deficiencies are therefore:

- Efficient handling of different levels of granularity, i.e., being able to grant access to a big variety of operations while still having fine-grained control on important operations in the context of the delegated task.
- Continued control of delegated task, i.e., the one delegating must still be able to supervise important parts of the task due to his responsibility. Execution of an abusive operation call should be immediately denied *before* any damage occurs.

- Fast and simple revocations of rights. Revocation of rights in case of an abuse or at the end of the task should happen immediately.
- Sharing responsibility of a task and thus sharing control of the execution.

Fig. 1 shows an example scenario for the installation of an update of a software package on workstations of department A: As installation means additional expenses for A, the head of A must approve it and delegates the installation task to the head of the IT management department that is responsible for software installations. However, as this installation will affect the user workstations the delegation includes the requirement that each workstation user must approve the installation on his workstation, e.g., to save particular personal settings before the installation. The head of IT must approve each installation for internal billing or because it may require more licences. He finally delegates the task to a subordinate who performs the actual installation on the workstations. Each of the participants shares a part of the responsibility.

In this paper we present an approach that allows to manage and control the security constraints of delegated tasks in order to overcome the limitations of existing subject-object-oriented security models. Introducing the concept of *multiple authorization*, this leads to a system where a secure delegation of tasks is possible. The concept allows that several participants authorize an operation instead of just one, thereby coming up with a mean to delegate while still taking (part of the) responsibility for the task. Software agents are a particularly good way to implement and enforce these requirements of multiple authorization because multiple agents can cooperate to fulfill their goals. Thus, authorizing operations of others can be part of it. This has advantages over approaches when there is an actual transfer of rights: The right always remains in the possession of the one delegating and thus he has still control of it. This limits the possibilities to exploit or abuse rights. Yet, to be clear, multiple authorization is no replacement for a good security management and implementation of the security system. Assignment of rights to users or creation of security policies must still be done with great care. However, passing rights to others in the context of a delegated task becomes more flexible and more secure. Moreover, at the current stage, although the approach presented is generic and generally usable, it is not intended as replacement of other security models but as an extension. This means that the main application is considered in the area of exceptional or highly security sensitive cases when there is a need for particularly fine-grained and well-suited access control. In the context of this paper, it is not for the “base” security. More research is necessary to study its application and constraints for a general security model based on this approach.

This approach will be the basis for a coming implementation in the Mobile Agent System Architecture [5][9] (MASA). MASA is a MAF [13] (*Mobile Agent Facility* – also known as MASIF)-compliant platform for mobile agents that has particularly been designed for use in dynamic and distributed IT management systems. MASA integrates a security model that allows flexible authentication and authorization of agents. Multiple authorization is intended as an extension to this security model.

In the following section we summarize existing approaches with respect to the requirements of task delegation. In section 3 we then briefly describe mobile agent technology in the area of IT management before we present how these systems offer a particular good solution for delegation in section 4. The discussion of some important aspects and issues of the new security architecture follows in section 5 before we finally conclude in section 6.

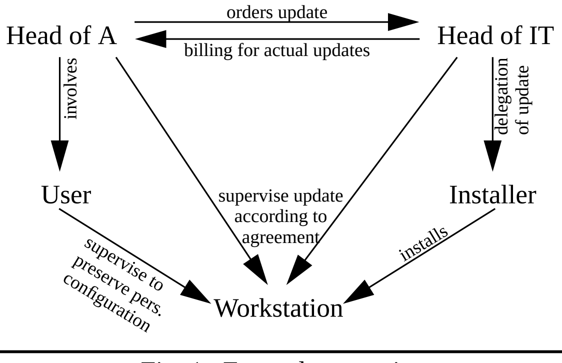
Head of A
orders update
billing for actual updates
Head of IT
involves
delegation of update
User
supervise update according to agreement
Installer
supervise to preserve pers. configuration
installs
Workstation

Fig. 1. Example scenario

2 Related Work

One still very common approach for delegation is “all-or-nothing”. These are all systems without fine grained administrative rights but with a single administrator access like many UNIX derivatives. This means that anyone who knows the administrator password of a machine can virtually do anything he wants without any restriction. This is certainly very dangerous as it is hard to monitor what actually happens.

Systems like OpenVMS [2] has a more fine grained system of rights. In addition, it has a special management right for the assignment of rights. This can be considered as a right for delegation. However, once assigned, the system does not know whether a right has been delegated. Revoking the management right does not imply revoking everything granted with this right. In addition, there is no kind of control what actually is done with this right or who received a right in this way and if he uses it the intended way.

Policy-based systems like [11] establish a consistent set of rules that define rights and obligations of the users. They may include rules for delegation of tasks and rights. Delegations may be cascaded or limited to a single step. Nevertheless, the problems of supervision and control of the specific task persist. Revocation of rights is also difficult as it takes time to propagate new policies to all receivers. Some approaches like [10] suggest that delegated rights expire after some time, e.g. 30 minutes, and must be refreshed if still required. However, time constraints for automated processes and agents must be stricter and there is often still enough time to do unintended things.

Task-based Authorization Controls (TBAC) [17] is an approach towards a task-oriented security model. It extends the classical subject-object access control by “validity counts” and “authorization-steps”. A protection state represents the active permissions of each step. While this helps to link authorizations to specific tasks, it needs a consistent way to control the operations to be authorized to prevent any kind of abuse.

SPKI [3] and X.509 Attribute Authorities [7] allow a fine-grained access control by signed attributes that specify the access rights. Both consider concepts for delegation of rights. However, it is very hard to bring the delegation into the context of a specific task. Thus, the problem remains that rights are passed independently from a task and it is not possible to control the usage on the background of the delegated task. The X.509 privilege policy may be considered for an extension into this direction. Yet, the draft standard does not define any syntax.

3 Mobile Agents in IT-Management

Integration of mobile agent technology in IT management is one approach towards a flexible, distributed management system [1][4][6][12][14] in order to overcome deficiencies of existing centralized approaches. *Mobile Agents* (MA) are autonomous software units. They act on behalf of their creator and are able to migrate inside the network to other places. Therefore, the necessary infrastructure consists of *Mobile Agent Systems* (MAS) executing agents and providing the necessary environment, e.g. communication services to mobile agents. As the MAS executes the agents it also offers access to required objects and management interfaces and must therefore control any access to protect the objects and the whole system. (Fig. 2) The MAS adapts the underlying, heterogeneous host systems to build a homogeneous execution platform on top of it. For the rest of this paper we assume, that the host system relies on the local MAS to provide a properly working access control to any host resource. In this sense, the MAS becomes integral

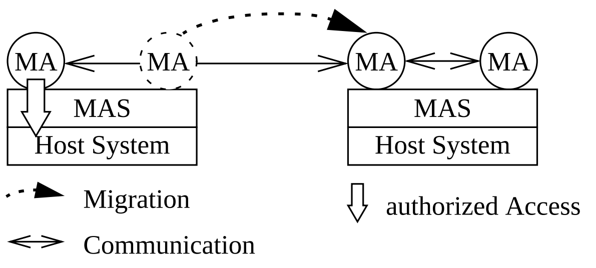

Fig. 2. Basic structure of a mobile agent infrastructure

part of the host system and, thus, an agent is not able to augment the privileges of the MAS or to have more rights than the MAS on the host system.

A mobile agent consists of program code to be executed and data required for execution. Despite of this, additional security attributes maintain integrity and security of the agent and the agent systems [15][16]. Of particular interest in the scope of this paper are the credentials or authentication information carried by an agent. For example, the initiator of an agent should digitally sign the agent to protect the integrity. This signature identifies the person who sends the agent and, thus, can also define the basic set of rights. In this sense, a legal signature is the access key to the system. Depending on the implementation, the agent may contain a more detailed description which rights it actually carries.

When an agent arrives, the MAS checks the signature and any further security related information of the agent and configures the run-time environment for this agent according to the rights of the agent. When the agent accesses a protected object or operation, it checks if it has the right to do so and either grants or rejects it. We call this *simple authorization* (Fig. 3). When a mobile agent migrates to another place, it carries its current state together with other collected data. The MAS will usually add its signature to this information to maintain the integrity.

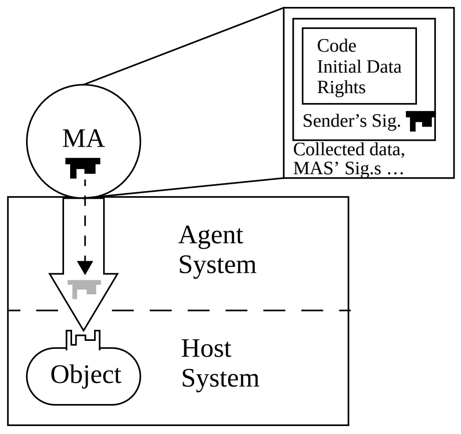

Fig. 3. Example for simple authorization

4 Multiple Authorization & Delegation

Mobile agents offer a way to handle authorization problems related with delegation intuitively without extending the rights of somebody for a period of time. Instead, other agents of the persons delegating authorize important, sensitive actions. We call this concept *multiple authorization* in contrast to a simple authorization when a single subject is enough, i.e., more than one authorization is required before access to an operation is granted. Once all required authorizations are present the operation will be executed. Every subject required to authorize the operation has the opportunity to make any kind of checks on the operation to be executed and the state of the system. It may consult other security services, if necessary. Moreover, it can make some auditing and logging before it decides whether to grant or refuse authorization. It can also depend the own authorization on more authorizations by other agents to include them in the responsibility for the execution. (Fig. 4).

Thus, there is no need for a transfer of rights or even transfer chains between agents to delegate the task. This means, that it is technically possible to implement the ideal form of delegation: the one delegating is still in control of the essential parts and can supervise the actions of the subordinate. Especially on the level of agents, this is a way to delegate tasks including the required rights without relying on a network of fully trustworthy agent systems and having some private key information in an agent (usually a prerequisite for an automated transfer of rights in other systems). The only assignment of rights happens at the initiation of the agent.

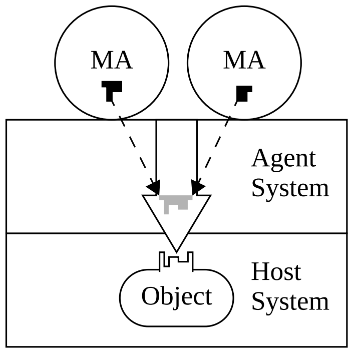

Fig. 4. Example of multiple authorization

This is a big advantage over other delegation concepts that actually transfer rights where there is no more control over the usage. In addition, revocation gets simpler. The authorizing agent itself can include some revocation logic to prevent abuse. But even if it is necessary to revoke an agent, because there is situation that has not been considered before, it is still easier as there is only a single agent at a single place to be contacted (except if just a single right was required on a single machine, then it would be the same effort). This allows to keep the security system relatively simple and easy to maintain while important parts can be individually protected for each task.

4.1. Authorization Agents

We call an agent that is able to provide additional authorization for a pending operation *Authorization Agent*. It must be especially equipped to decide which operations are allowed in the context of the task and which are to be denied. To make this decision the agent must migrate to the system involved as only local decisions on this system are fully trustworthy. If the agent ran on a different system, the local system would have to trust the other system. The agent must be very robust in that it is not possible to temper with the decision logic in any way. Ideally, the agent is instantiated on a machine fully trusted by its owner and is able to make correct decisions independently from any place it has been before. Therefore, it should not carry any collected data that might influence the decision. Only then, the agent can reliably fulfill its purpose to authorize an operation. Therefore, its only purpose should be to authorize operations.

The creation of this agent is the correspondent part for the necessary delegation of rights in other systems. However, before it authorizes an action, it can consider whatever is necessary to maintain a high level of security. For example, the Head of Department (HoD) in Fig. 5 is responsible for the software installation and delegates the task to the installer, that has insufficient rights to do the task. The created authorization agent must authorize operations of the installer, if necessary. Therefore, it examines the oper-

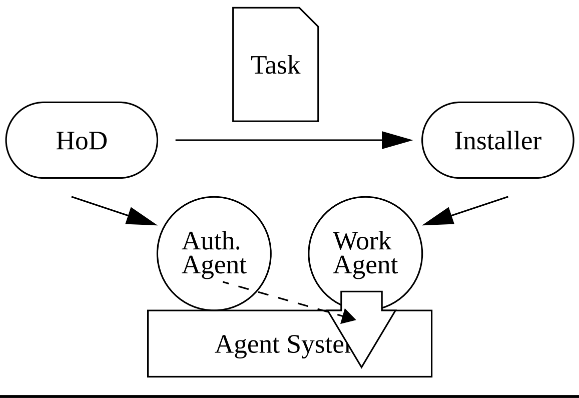

Fig. 5. Delegation of Task and Rights

ation and enforces high-level security constraints, e.g., it makes sure that the installer only writes to one directory intended for this task and that no files have been manipulated checking a file signature. This is usually impossible to achieve in traditional systems.

This conceptional separation of agents into work agents and authorization agents allows delegation of tasks even over several steps. The delegation process is split into two parts: the delegation of the task itself (the work to do) and the necessary authorization of the task (the rights required to do the work). The way how the work part is actually delegated is not in the scope of this paper. It requires a method to describe and communicate the task. This can happen verbally between people or by negotiation between agents.

The authorization agent is responsible to authorize the operations of the delegated task. If the task (or a part of it) is again delegated we have a *cascaded delegation* (see Fig. 6). Following the delegation trail, the authorization agent of HoD now actually authorizes the authorization agent of the head of the IT department (IT) that in turn authorizes the Installer agent. However, this is usually handled completely inside the agent system and is thus transparent to the HoD agent.

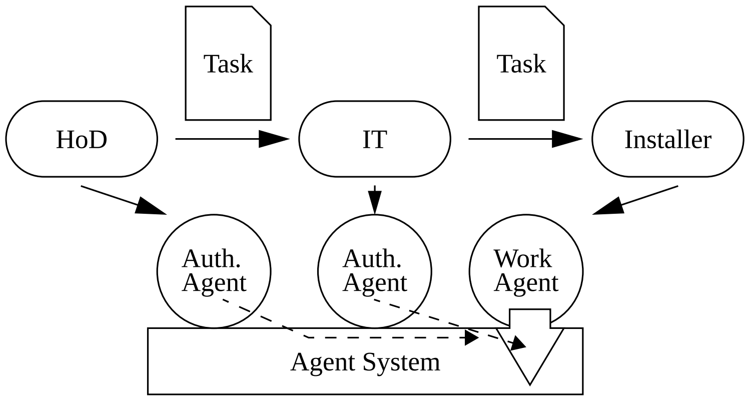

Fig. 6. Cascaded Delegation of Task and Rights

4.2. Example Scenario

To demonstrate this concept let us consider the software installation scenario of section 1. The software install agent written by the software installer is equipped with all necessary configuration and installation information. The installer signs the agent to approve its integrity and to assign his software installation right. The software installation right alone does not allow an installation on a particular machine but is a prerequisite. So, let us assume, there is a policy in the company that the head of the department (HoD) responsible for the machine must also authorize any installation on any of his machines.

Fig. 7 shows the control flow for this scenario. If the installer agent on the target system calls an installation operation (1) it is considered to have *incomplete authorization*. The authorization code of the agent is informed that for successful execution the authorization of the HoD is necessary (2). Therefore, the agent sends a message to the agent of the HoD to request it (3). The HoD's agent decides to send out a prepared agent to this machine to supervise the correct installation. This agent now checks the context of the operation call (4) and requests further approval by the head of IT (5) and the user of the machine (6). Both again have the same possibilities to send agents and to supervise and control the installation before they actually grant authorization. The agent of the head of IT makes sure that the extent of the installation happens according to the agreement with the HoD and that everything follows the installation policies of the IT department (7). The user of the machine saves some personal preferences during installation (8). Once all participants are satisfied and approve we have a *complete authorization*. The operation is finally executed and the installer can continue (9).

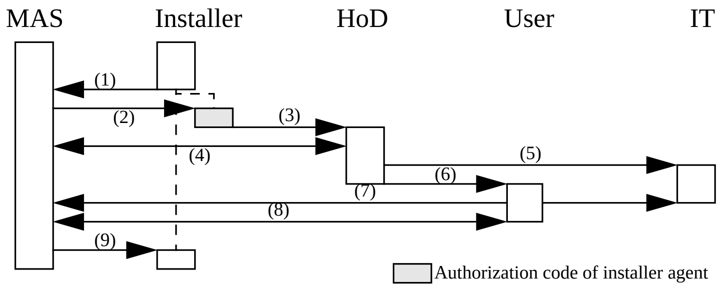

Fig. 7. Control flow in example scenario

5 A New Security Architecture

Multiple Authorization as outlined above has certainly an impact on the architecture of the management system, in particular on the security-related parts. The architecture must integrate mechanisms for delaying execution of an incompletely authorized operation and offer interfaces for others to supply the missing authorization including interfaces to inspect the delayed operation. Therefore, in this section, we discuss the requirements and consequences for the system following the life cycle of an operation subject to multiple authorization. In the scope of this paper we limit this discussion to the synchronous case when an operation call is delayed until the authorization has been given. Nevertheless, asynchronous processing is possible and should be considered for optimization of the overall cycle.

5.1. Access Control & Rights

When the *initiating agent* (Fig. 8) tries to access a restricted resource, e.g., calls a protected method of an object, the security system of the agent system intercepts the call and checks if this access is to be allowed or rejected, i.e., if the agent is authorized. Therefore, it authenticates the agent to identify the owner of the agent. The agent may as well carry further information about which of the owner's rights are actually active or in which role it acts. Based on the authentication the security system can determine the access rights of the agent. However, with multiple authorization there are now three possible answers: *grant*, *reject* or *incomplete*. If access is granted the operation is executed, if it is rejected an error is returned. In the third case, the agent is principally allowed to call the operation, but needs additional authorization by other parties (see Table 1).

Therefore, the access control list has an additional field *responsible* that contains possible prospects that may provide additional authorization. These are put on a list of pending authorizations of this operation call. To indicate this case, the security system now calls the *authorization code* of the agent. The authorization code is a special part of the agent that is responsible to complete any pending authorization, i.e., to find other agents that are able and willing to authorize this operation. As the agent works in the scope of a delegated task, it contacts the delegating agent and waits for completion of the operation. It passes an operation identifier of the security system to the delegating agent that uniquely determines the operation.

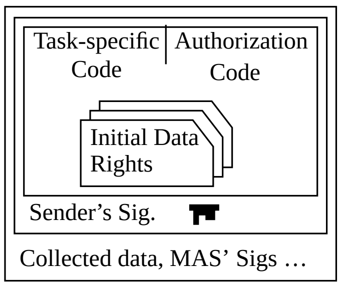

Fig. 8. Structure of initiating agent

Identity/Role	**Access**	**Responsible**
HoD	grant	
Installer	incomplete	HoD
other	reject	

Table 1: Access control list at target system

If the authorization agent of the delegating agent is already present at the local system because of previous calls it can register with the security system to directly catch all related operation calls to be authorized. Doing so, the initiating agent is not directly involved anymore and the authorization process is optimized.

5.2. Inspection Interface

The authorization agent contacted by the initiating agent must migrate to the agent system to make sure that the decision is not influenced by others. Only then, the local mobile agent system can trust in the decision of the agent. The agent must inspect the operation to be authorized before it makes its decision. Therefore, the security system offers a special interface (see Fig. 9) for authorization agents that allows them to inspect the called operation with its parameters as well as the context and status of the system it is running on. The agent can base its decision on any information that seems necessary (and it is allowed to access). In this sense, the authorization agent becomes part of the access control mechanism. Certainly, the security system allows access to this interface only to agents that are able to supply the required authorization.

When the initiating agent contacts the authorization agent it passes an identifier of the operation to be authorized. This identifier allows to access all information related with this operation. Therefore, the security system identifies the authorization agent and checks the list of pending authorizations for this operation to see whether this agent is allowed and requested to supply an authorization. If not, the access to the operation information is denied. Otherwise, the agent gets the operation information that consists of:

- the identity of the caller of the operation
- the object to be accessed
- the method to be called on this object
- all parameters passed to this method call
- the list of previously supplied authorizations by other agents

The agent may include other information available to make its decision, e.g., it may consult logging files to find out about previous actions of this agent or check the state of the system. The authorization agent can

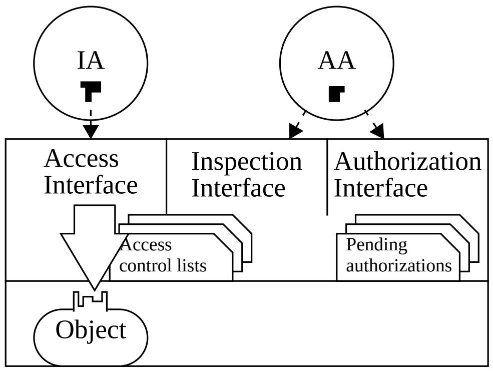

Fig. 9. New Security Architecture with Initiating and Authorization Agent

do (only limited by its own rights) whatever is necessary to be sure that the operation call can be executed without harm or damage.

5.3. Authorization Interfaces

Once the authorization agent has come to a decision it uses the authorization interface of the security system to give the answer. There are five possible answers that the agent of A can give: *authorized*, *delegated, delayed, rejected*, or *not responsible* (see Table 2). If the agent finds that the operation call is acceptable with no limitations, it authorizes the operation to the extent it (or his owner) is able to. If this was the last missing part of the authorization and there is no other authorization pending, the operation call is finally executed.

However, if the task has been delegated from A to B, and the operation call was not made by an agent of B himself, the agent must make its authorization dependent on the authorization of B. There are two alternatives to do this: *delegated to B* is used to indicate that A authorizes the operation under the condition that B does also (or more identities). In this case, it is up to the security system to ensure this. A is not consulted for this particular operation call anymore. *delayed for B*, on the other hand, means that A will not authorize the operation before B does. After B has granted authorization, A has again the choice what to do. It depends on the situation which of the two alternatives is appropriate.

The agent rejects the execution if it finds anything that is in violation to its decision policy. In this case, the operation call returns with an error. If the agent has been consulted but does actually know nothing about the operation it indicates that it is not responsible.

Authorization	**Action**
authorized	mark operation as authorized by A
delegated to B	mark operation as authorized by A, add B to pending authorizations
delayed for B	add B to pending authorizations
rejected	deny operation execution
not responsible	nothing

Table 2: Possible authorizations of an authorization agent

6 Conclusions & Further Work

The concept of delegation is very important in IT management. Each delegation of a task must include the right to do this task. All systems, components and functions required for this task must be accessible for the delegated person. However, it is usually impossible to control and supervise delegated management tasks. In most existing systems it is impossible to specify rights to a degree that enables the user to execute the task and still prevents any kind of abuse. Instead, the assigned rights create a new threat to the target system and sometimes even to the complete network.

We suggested the concept of a multiple authorization added to a management system based on mobile agents. With multiple authorization a single user is not able to execute protected operations but needs additional authorization by others. Before granting authorization these other participants are able to make checks on the operation to be executed and base their decisions on that. Multiple authorization, therefore, allows to maintain some shared responsibility for protected operations and to supervise actions of subordinates. As it is possible to bind authorization to particular operations under special circumstances, a very fine grained access control is achieved for the delegated task.

The agent technology provides a flexible mean to transport the necessary authorizations in the context of a delegated task. It allows automated checks and the enforcement of further security constraints before an operation is actually executed. We outlined the basic components and mechanisms of a new security architecture that includes this new concept.

Yet, to be clear, multiple authorization is no replacement for a proper security management and implementation of the security system. Assignment of rights to users and creation of security policies must still be done with great care. However, passing rights to others in the context of a delegated task is more flexible and more secure than other approaches.

We are working on a first implementation in our agent system MASA [5][9], a research prototype to investigate the benefits and drawbacks of mobile agents for IT management. Further research is necessary to evaluate different implementations and strategies for agents and how authorizations are handled inside the security system. In particular, with respect to security management it is important to analyze which way of specification of access rights is best to suit the needs of partial authorizations. To simplify the creation of authorization agents a development toolkit together with a set of templates is anticipated. This allows a fast creation of authorization agents when required for a delegation.

Acknowledgement

The author would like to thank the members of the Munich Network Management (MNM) Team for helpful discussions and valuable comments on earlier drafts of this paper. The MNM Team directed by Prof. Dr. Heinz-Gerd Hegering is a group of researchers at the University of Munich, the Munich University of Technology, and the Leibniz Supercomputing Center of the Bavarian Academy of Sciences. Its webserver is located at `http://wwwmnmteam.informatik.uni-muenchen.de/`

References

[1] A. Bieszczad, B. Pagurek, and T. White. *Mobile agents for network management.* IEEE Communication Surveys, 1(1), 1998.

[2] Compaq. *OpenVMS Guide to System Security.* OpenVMS VAX Version 7.2. Houston, Texas, USA. January 1999.

[3] C. Ellison, et al. *SPKI Certificate Theory.* RFC 2693. September 1999.

[4] G. Goldszmit and Y. Yemini. *Distributed Managment by Delegation.* In Proceedings of the 15th International Conference on Distributed Computing Systems, June 1995.

[5] B. Gruschke, et al. *Mobile Agent System Architecture - Eine Plattform für flexibles IT-Management.* (Mobile Agent System Architecture – A Platform for flexible IT Management. In German). Technical Report 9902, Ludwig-Maximilians-University Munich, Institute for Computer Science, Munich, August, 1999.

[6] H.-G. Hegering, S. Abeck, and B. Neumair. *Integrated Management of Networked Systems – Concepts, Architectures and their Operational Application.* Morgan Kaufmann Publishers, 2000.

[7] ITU-T Recommendation X.509, ISO/IEC 9594-8. *Information Technology – Open Systems Interconnection – The Directory: Public-Key and Attribute Certificate Frameworks.* 4th Edition, Draft V7. February, 2001.

[8] J.B.D. Joshi, W.G. Aref, A. Ghafoor, and E.H. Spafford. *Security Models for Web-based Applications.* Communications of the ACM, Vol. 44, No. 2, pp. 38–44, February 2001.

[9] B. Kempter, H. Reiser, H. Rölle, and G. Vogt. *Implementierung eines MASIF konformen Agentensystems – Die Mobile Agent System Architecture (MASA).* (Implementation of a MASIF conform Agentsystem – The Mobile Agent System Architecture (MASA). In German). PIK – Praxis der Informationsverarbeitung und Kommunikation, Vol. 24, No. 3, July-September 2001.

[10] B. Lampson, M. Abadi, M. Burrows, and E. Wobber. *Authentication in Distributed Systems: Theory and Practice.* ACM Transactions on Computer Systems, 10(4):265–310, November 1992.

[11] E. C. Lupu, D. A. Marriott, M. S. Sloman, and N. Yialelis. *A policy based role framework for access control.* In First ACM/NIST Role Based Access Control Workshop, December 1995.

[12] M.-A. Mountzia. *Flexible Agents in Integrated Network and Systems Management.* PhD Thesis, Munich University of Technology, December 1997.

[13] Object Management Group. *Mobile Agent Facility Specification*. Version 1.0. CORBAfacilities specification formal/00-01-02. January 2000.

[14] R. Pinheiro, A. Poylisher, and H. Caldwell. *Mobile Agents for Aggregation of Network Management Data.* In First International Symposium on Agent Systems and Applications and Third International Symposium on Mobile Agents (ASA/MA 99), pages 130–140, Palm Springs, California, October, 3–6 1999. IEEE.

[15] H. Reiser and G. Vogt. *Security Requirements for Management Systems using Mobile Agents.* In S. Tohme and M. Ulema, editors, Proceedings of the Fifth IEEE Symposium on Computers & Communications, pages 160–165, Antibes-Juan Les Pins, France, July 2000. IEEE.

[16] H. Reiser and G. Vogt. *Threat Analysis and Security Architecture of Mobile Agent based Management Systems.* In J. W. Hong and R. Weihmayer, editors, NOMS 2000 IEEE/IFIP Network Operations and Managment Symposium — The Networked Planet: Management Beyond 2000, page 979, Honolulu, Hawaii, USA, April, 10-14 2000. IEEE.

[17] R.K. Thomas and R.S. Sandhu. *Task-based Authorization Controls (TBAC): A Family of Models for Active and Enterprise-oriented Authorization Management.* In Proceedings of the IFIP WG11.3 Workshop on Database Security, Lake Tahoe, California, August 11-13, 1997.