An Intrusion-Tolerant Authorization Scheme for Internet Applications

Yves Deswarte, Noreddine Abghour, Vincent Nicomette, David Powell
LAAS-CNRS
7, Avenue du Colonel Roche
31077 Toulouse cedex 4 — France
{Yves.Deswarte, Noreddine.Abghour, Vincent.Nicomette, David.PowelL}@laas.fr

Abstract

This paper presents an authorization scheme for applications distributed on the Internet with two levels of access control: a global level, implemented through a fault- and intrusion-tolerant authorization server, and a local level implemented as a reference monitor located on both the local host Java Virtual Machine (JVM) and on a Java Card connected to this host.

1. Introduction

Today, most Internet applications are based on the client-server model. In this model, typically, the server distrusts clients, and grants each client access rights according to the client's identity. This enables the server to record a lot of personal information about clients: identity, usual IP address, postal address, credit card number, purchase habits, etc. Such a model is thus necessarily privacy intrusive.

Moreover, the client-server model is not rich enough to cope with complex transactions involving more than two participants. For example, an electronic commerce transaction requires usually the cooperation of a customer, a merchant, a credit card company, a bank, a delivery company, etc. Each of these participants has different interests, and thus distrusts the other participants.

Within the MAFTIA[1] project, we are developing authorization schemes that can grant fair rights to each participant, while distributing to each one only the information strictly needed to execute its own task, i.e., a proof that the task has to be executed and the parameters needed for this execution, without unnecessary information such as participant identities. These schemes are based on two levels of protection:

- An *authorization server* is in charge of granting or denying rights for composite operations involving several participants; if a composite operation is authorized, the authorization server distributes capabilities for all the elementary operations that are needed to carry it out.
- On each participating host, a *reference monitor* is responsible for fine-grain authorization, i.e., for controlling the access to all local resources and objects according to the capabilities that accompany each request. To enforce hack-proofing of such reference monitors on off-the-shelf computers connected to the Internet, critical parts of the reference monitor are implemented within a Java Card.

In the following sections, the general authorization architecture and the reference monitor are described, and finally, our approach is compared to related work.

2. General authorization architecture

In [1], we proposed a generic authorization scheme for distributed object systems. In this scheme, an application can be viewed at two levels of abstraction: composite operations and method executions. A composite operation corresponds to the coordinated execution of several object methods towards a common goal. For instance, printing file F3 on printer P4 is a composite operation involving the execution of a *printfile* method of the spooler object attached to P4, which itself has to request the execution of the *readfile* method of the file server object managing F3, etc.

A request to run a composite operation is authorized or denied by an authorization server, according to *symbolic rights* stored in an access control matrix managed by the authorization server. More details on how the authorization server checks if a composite operation is to be granted or

[1] MAFTIA (Malicious- and Accidental-Fault Tolerance for Internet Applications) is a project of the European IST Program (project IST-1999-11583). MAFTIA partners are the University of Newcastle upon Tyne (GB), prime contractor, DSTL (GB), IBM Zurich Research Lab. (CH), LAAS-CNRS (F), QinetiQ (GB), University of Lisbon (P) and University of Saarland (D). See <http://www.maftia.org/>.

denied are given in [2] and [3]. If the request is authorized, capabilities are created by the authorization server for all the method executions needed to perform the composite operation. These capabilities are simple method capabilities if they are used directly by the object requesting the execution of the composite operation, i.e., used by this object to directly call another object's methods. Alternatively, the capabilities may be indirect capabilities or *vouchers*, if they cannot be used by the calling object but must be delegated to another object that will invoke other object methods to participate in the composite operation. In fact, the notion of composite operation is recursive, and a voucher can contain either a method capability or the right to execute a composite operation.

This delegation scheme is more flexible than the usual "*proxy*" scheme, by which an object transmits to another object some of its access rights for this delegated object to execute operations on behalf of the delegating object. Our scheme is also closer to the "*least privilege principle*", since it helps to reduce the privilege needed to perform delegated operations. For instance, if an object O is authorized to print a file, it has to delegate a *read-right* to the spooler object, for the latter to be able to read the file to be printed. To delegate this read-right, with the proxy scheme, O must possess this read-right; and thus O could misuse this right by making copies of the file and distributing them. In this case, the read-right is a privilege much higher than a simple print-right. In our scheme, if O is authorized to print a file, O will receive a *voucher* for the spooler to read the file, and a capability to call the spooler. The voucher, by itself, cannot be used by O. With the capability, O can invoke the spooler and transmit the voucher to the spooler. The spooler can then use the voucher as a capability to read the file.

Since only composite operations are managed by the authorization server, the system security is relatively easy to manage: the users and the security administrators have just to assign the rights to execute composite operations, they do not have to consider all the elementary rights to invoke object methods. Moreover, since only one request has to be checked for each composite operation, the communication overhead can be reduced.

The authorization server is a trusted third party (TTP), which could be a single point of failure, both in case of accidental failure, or in case of successful intrusion (including by a malicious administrator). To prevent this, with the MAFTIA authorization architecture [3], the authorization server will be made fault- and intrusion-tolerant: an authorization server is made of diverse security sites, operated by independent persons, so that faults and intrusions can be tolerated without degrading the service, as long as only few security sites are affected.

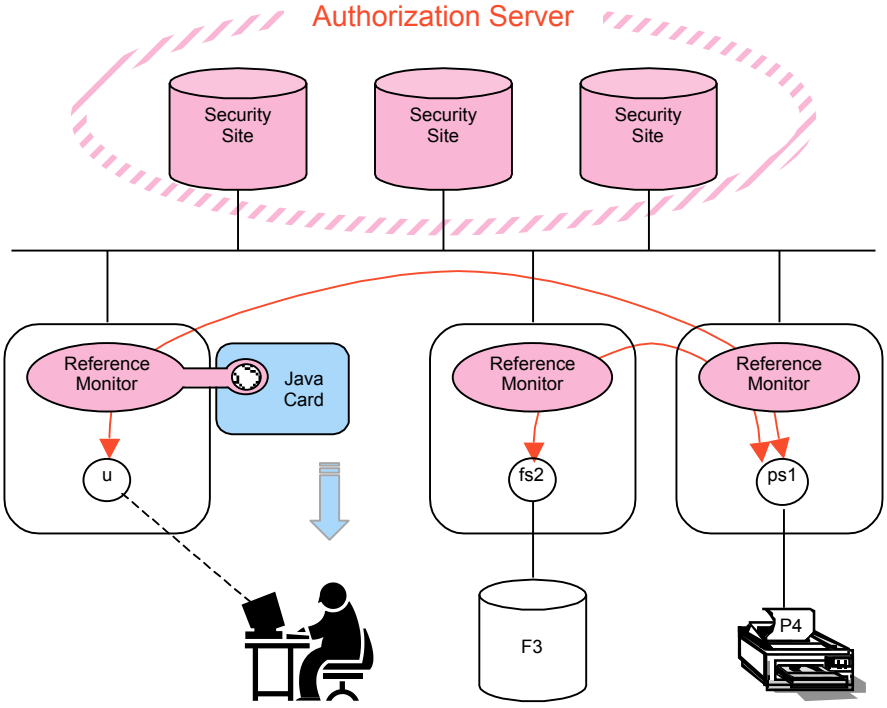

Figure 1. Authorization architecture

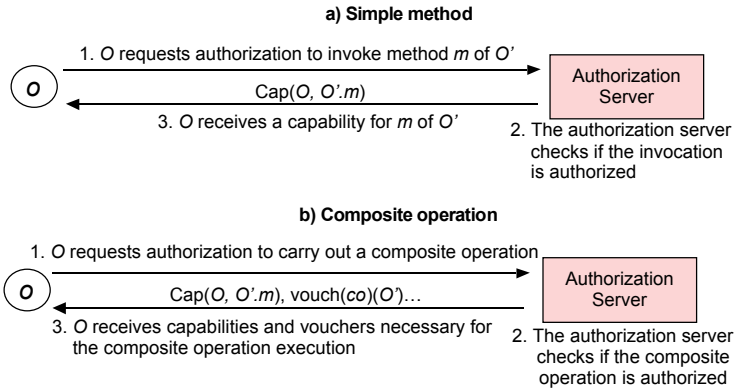

Figure 2. Protocol between a MAFTIA object and the authorization server.

In order to tolerate the failure of one or a small number of the sites composing the authorization server, several protocols are used:

- Mutual agreement: all non-faulty sites agree on the decision to grant or deny the authorization corresponding to a given request. This guarantees a correct decision as long as there is only a minority of faulty sites[2].
- *Threshold signature*: the capabilities and vouchers are globally signed by the authorization server, using a threshold signature scheme. Each of the sites composing the authorization server generates a signature share (depending on its own private key share) so that if at least t valid signature shares are available (t being the threshold), it is possible to combine these shares to generate a unique signature that can be verified with a global public key. This guarantees that if a capability (or a voucher) has a correct signature, the corresponding operation is indeed authorized (the signed capability cannot be forged, even by a cooperation of f faulty sites, as long as f is strictly less than the threshold t).

All these protocols are developed in other parts of the MAFTIA project [4, 5].

The global architecture is given by Figure 1. The dialogue between a MAFTIA object and the authorization server is typically as follows (see Figure 2.):

Object *O* asks the authorization server for the authorization to carry out an operation in the system. This operation may be the simple invocation of a particular method of a particular object *O'* or may be a "composite operation" that requires the collaboration between several objects in the system.

In the first case, if object *O* is authorized to carry out the operation, it receives a capability, ciphered by the public key of the host where *O'* is located, and then signed using the threshold scheme described above. This capability will be presented and checked by the reference monitor located on the site of the invoked object *O'*.

In the second case, the user may receive several capabilities and vouchers. Capabilities are directly used by object *O* to invoke particular methods of particular objects, and are ciphered and signed as in the first case. Vouchers are not used by object *O* but are forwarded by object *O* to other objects that are involved in the execution of the composite operation (e.g., a capability for *O'* to invoke a method *m* of an object *O"*, as a part of the composite operation). These vouchers will thus be transferred by object *O* to other objects, which will then execute their part of the composite operation thanks to these vouchers. A voucher may be a capability (in which case they are ciphered and signed as above), or the right to execute another composite operation (in which case the voucher is just signed).

3. Reference monitor

There is a reference monitor on each host participating in a MAFTIA-compliant application. The reference monitor is responsible for granting or denying local object method invocations, according to capabilities and vouchers distributed by the authorization server. In the context of wide-area networks (such as the Internet), the implementation of such a reference monitor is complicated since, due to the heterogeneity of connected hosts, it would be necessary to develop one version of the reference monitor for each kind of host. Moreover, since the hosts are not under the control of a global authority, there is no way to ensure that each host is running a genuine reference monitor, or the same version

[2] In practice, the number f of faulty sites may have to be much less than half the total number n of sites, depending on the fault assumptions. For instance, ($n > 3f$) must be guaranteed if Byzantine faults are to be taken into account.

thereof. This is why we have chosen to implement them by using Java Cards.

The reference monitor has the responsibility of deciding whether or not to authorize the invocation of particular methods on particular objects on the local host by checking that the corresponding capabilities are presented. These checks represent the central part of the authorization scheme, and thus have to be protected as strongly as possible. We have chosen to implement them on a Java Card, which we consider as sufficiently tamperproof. In particular, any software, even that within an operating system or a JVM, can be copied and modified by a malicious user who possesses all privileges on a local host. In particular, on Internet, any hacker can easily have these privileges on his own computer! With capability checks run on the Java Card, we can be sure that any remote request to execute a MAFTIA-application is genuine (if the capability is correct), and that a genuine MAFTIA request can only be executed on the host for which the capability is valid. The hacker's privileges on his host gives him no privilege outside that host.

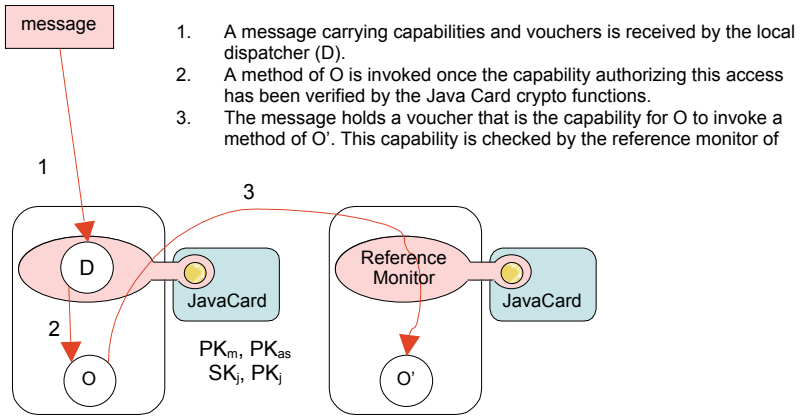

Figure 3. Example of a voucher corresponding to a capability

The capability checks carried out by the Java Card are based on strong cryptographic functions. Several cryptographic keys must be included in the Java Card:

- PK_m, the MAFTIA public key. This key is associated to the MAFTIA private key SK_m, which is not stored in the Java Card. The Java object classes are signed off-line by this key SK_m, and this signature is checked at load time by the local JVM of the host[3], using PK_m stored in the Java Card.
- SK_j, PK_j, a private/public key pair specific to the Java Card, thus specific to the host.
- PK_{as}, the authorization server public key. This key is associated to all the private key shares of all the sites composing the authorization server.

Each capability is ciphered by the authorization server, using the public key PK_j of the site where the invoked object is located. Then the capability is signed by the authorization server (with the threshold signature protocol presented in Section 2). Consequently, the capability signature must first be verified using the authorization server's public key PK_{as}, and then deciphered (by the cryptographic functions of the Java Card) using the private key SK_j, which is stored only in the Java Card. Each access to a method of an object on a MAFTIA host is thus controlled by its local Java Card. This verification corresponds to step 2 of Figure 3.

Other information can be stored in the Java Card, for instance for the authentication of the user, owner of the Java Card if the host is a personal workstation, or for the authentication of the administrator who has been assigned this Java Card if the host is a server. In the latter case, it may be possible to have several administrators for the same server, each administrator having his personal Java Card for this server, and all the server administrator Java Cards sharing the same pair SK_j, PK_j. More details on the Java Card implementation of the reference monitor can be found in [6].

4. Related work

The basic authorization scheme was developed in [7]. This work was the first attempt to introduce the voucher delegation scheme, and to demonstrate its ability to implement closely the least privilege principle.

[3] Since version 1.2, the Java Development Kit includes software that allows classes to be signed and the signatures to be checked at load time.

Other schemes have been recently introduced to provide more flexibility and more efficiency than the client-server model. In particular, [8] proposes to carry out access control in a distributed system by means of "communal laws". This paper addresses also the problem of revocation, which is not directly addressed in our scheme, even if an expiry time can be included in our capabilities and vouchers. However, it seems that the scheme presented in [8] may be difficult to implement.

The notion of "authorization server" is now relatively common when consistent access control has to be implemented in distributed systems. Even some public key infrastructure (PKI) implementations, such as SPKI [9], can be seen as a kind of authorization service. In the same way, the Kerberos V5 Ticket Granting Server [10] and SESAME Privilege Attribute Services [11], manage some authorization, but only at a coarse-grain level, for client-server interactions. Delta-4 [12] proposed also an authorization service, which has been implemented to control access to a persistent file storage service. Delta-4 was also the first attempt to implement fault- and intrusion-tolerant security services. Other recent authorization server implementations are the HP Praesidium [13] and Adage [14].

In the COCA project [15], a distributed intrusion-tolerant Certification Authority is developed, using dedicated multicast communication primitives based on different assumptions than those of MAFTIA. On most aspects, COCA's Certification Authority is similar to the generic CA developed within MAFTIA, but the authorization server is of course quite different in its functionality.

Concerning the use of smart cards for authorization, Au et al. [16] propose to use smart cards as portable, tamperproof storage for authorization tokens delivered by an authorization server and checked by an "authorization manager" (the equivalent of our reference monitor) on each application server. In their approach, the smart card is not used to implement the authorization manager of the application server, it is just used to store the authorization token. JCCap [17] proposed the use of capabilities to manage access controls between applications located on Java Cards, but their capabilities are defined statically by means of "views" during program development, rather than created dynamically by an authorization server. We consider that our approach is more flexible and closer to the least privilege principle.

5. Conclusion

The authorization scheme presented in this paper is flexible, easily managed (at the coarse-grain level of "composite operations") and efficient (fine grain access control at the object method invocation level, tamperproof reference monitors implemented with Java Cards). Moreover, it is not privacy intrusive, since personal information is disclosed to participants only on a "need-to-know" basis.

Since the implementation has just begun, no performance measurements are currently available. But since the authorization server is accessed only once for each composite operation, we hope that the induced overhead will be acceptable with respect to the gained security and privacy.

6. References

[1] V. Nicomette and Y. Deswarte, "An Authorization Scheme for Distributed Object Systems," *Proc. Int. Symposium on Security and Privacy*, Oakland, CA, USA, 1997, IEEE Computer Society Press, pp. 21-30.

[2] V. Nicomette and Y. Deswarte, "Symbolic Rights and Vouchers for Access Control in Distributed Object Systems," *Proc. 2nd Asian Computing Science Conference (ASIAN'96)*, Singapour, 1996, "*Concurrency and Parallelism, Progamming, Networking, and Security*", J. Jaffar and R. H. C. Yap, Eds., Springer-Verlag, LNCS n°1179, pp. 193-203.

[3] N. Abghour, Y. Deswarte, V. Nicomette, and D. Powell, *Specification of Authorisation Services*, LAAS-CNRS, Toulouse, MAFTIA Project IST 1999-11583 Deliverable D27, LAAS Report 01001, 33 pp., 23 January 2001, available at: <http://www.research.ec.org/maftia/deliverables>.

[4] J. Algesheimer, C. Cachin, K. Kursawe, F. Petzold, J. A. Poritz, V. Schoup, and M. Waidner, *MAFTIA: Specification of Dependable Trusted Third Parties*, IBM Research, Zurich Research Laboratory, Zurich (CH), MAFTIA Project IST 1999-11583 Deliverable D26, 98 pp., 22 January 2001, available at: <http://www.research.ec.org/maftia/deliverables>.

[5] C. Cachin, M. Correia, T. McCutcheon, N. F. Neves, B. Pfitzmann, B. Randell, M. Schunter, W. Simmonds, R. Stroud, P. Veríssimo, M. Waidner, and I. Welch, *Service and Protocol Architecture for the MAFTIA Middleware*, MAFTIA Project IST 1999-11583 Deliverable D23, 92 pp., 25 January 2001, available at: <http://www.research.ec.org/maftia/deliverables>.

[6] Y. Deswarte, N. Abghour, V. Nicomette, and D. Powell, "An Internet Authorization Scheme using Smartcard-based Security Kernels," *International Conference on Research in Smart Cards (e-Smart 2001)*, Cannes (France), 2001, "*Smart Card Programming and Security*", I. Attali and T. Jensen, Eds., Springer-Verlag, LNCS 2140, pp. 71-82.

[7] V. Nicomette, *Protection in Distributed Object Systems*, Thèse de Doctorat de l'Institut National Polytechnique, Toulouse, 1996, LAAS Report 96496, 177 pp. (in French).

[8] X. Ao, N. H. Minsky, and V. Ungureanu, "Formal Treatment of Certificate Revocation Under Communal

Access Control," *IEEE Symposium on Security and Privacy*, Oakland, CA, 2001, IEEE Computer Society Press, pp. 116-127.

[9] C. Ellison, *SPKI Requirements*, IETF RFC 2692, September 1999.

[10] B. C. Neuman and T. Tso, "Kerberos: an Authentication Service for Computer Networks," in *IEEE Communications*, vol. 32, 1994, pp. 33-38.

[11] T. Parker, "A Secure European System for Applications in a Multi-vendor Environment (The SESAME project)," *14th National Computer Security Conference*, Washington (DC, USA), 1991, NCSC and NIST, pp. 505-513.

[12] L. Blain and Y. Deswarte, "Intrusion-Tolerant Security Server for Delta-4," *ESPRIT 90 Conference*, Brussels (Belgium), 1990, CEC-DG-XIII, Ed., Kluwer Academic Publishers, pp. 355-370.

[13] HP, *HP Praesidium Authorization Server 3.1: Increasing Security Requirements in the Extended Enterprise*, pp., November 2 1998.

[14] M.-E. Zurko, R. Simon, and T. Sanfilipo, "A User-Centered, Modular Authorization Service Built on an RBAC Foundation," *IEEE Symposium on Security and Privacy*, Berkeley (CA, USA), 1999, pp. 57-71.

[15] L. Zhou, F. B. Schneider, and R. v. Renesse, *COCA: A Secure Distributed On-line Certification Authority*, Computer Science Department of Cornell University, Ithaca, NY (USA), Report ncstrl.cornell/TR2000-1828, 54 pp., 8 December 2000.

[16] R. Au, M. Looi, and P. Ashley, "Cross-Domain One-Shot Authorization using Smart Cards," *7th ACM Conference on Computer and Communications Security (CCS-2000)*, Athens, Greece, 2000, S. Jajodia and P. Samarati, Eds., ACM Press, pp. 220-226.

[17] D. Hagimont and J.-J. Vandewalle, "JCCap: Capability-Based Access Control for Java Card," *4th IFIP WG8.8 Working Conference on Smart Card Research and Advanced Applications (CARDIS-2000)*, Bristol, UK, 2000, J. Domingo-Ferrer, D. Chan, and A. Watson, Eds., Kluwer Academic Publishers, pp. 365-388.