An Internet Authorization Scheme Using Smart Card-Based Security Kernels

Noreddine Abghour, Yves Deswarte, Vincent Nicomette, David Powell

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

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 MAFTIA1 project, we are developing authorization schemes that can grant to each participant fair rights, 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 high-level operations involving several participants; if a high-level 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 *security kernel* 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 security kernels on off-the-shelf computers connected to the Internet, critical parts of the security kernel will be implemented on a Java Card.

In the following sections, the general authorization architecture and the security kernel are presented.

1. General authorization architecture

In [Nicomette & Deswarte 1997], we proposed a generic authorization scheme for distributed object systems. In this scheme, an application can be viewed at two levels of abstraction: high-level operations and method executions. A high-level operation corresponds to the coordinated execution of several object methods towards a common goal. For instance, printing file F3 on printer P4 is a high-level 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 high-level 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 high-level operation is to be granted or denied are given in [Nicomette & Deswarte 1996] and [Abghour *et al.* 2001]. If the request is authorized, capabilities are created by the authorization server for all the method executions needed to realize the high-level operation. These capabilities are simple method capabilities if they are used directly by the object requesting the execution of the high-level operation, i.e., used by this object to directly call another

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

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, itself, will invoke other object methods to participate in the high-level operation. In fact, the notion of high-level operation is recursive, and a voucher can contain either a method capability or the right to execute a high-level 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 privileges needed for performing 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 spooler to read the file to be printed. To delegate this read-right, with the proxy scheme, O must possess this read-right; so 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 high-level operations are managed by the authorization server, it is relatively easy to manage: the users and the security administrators have just to assign the rights to execute high-level 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 high-level 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 [Abghour *et al.* 2001], 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 any single fault or intrusion can be tolerated without degrading the service. The global architecture is given by Figure 1.

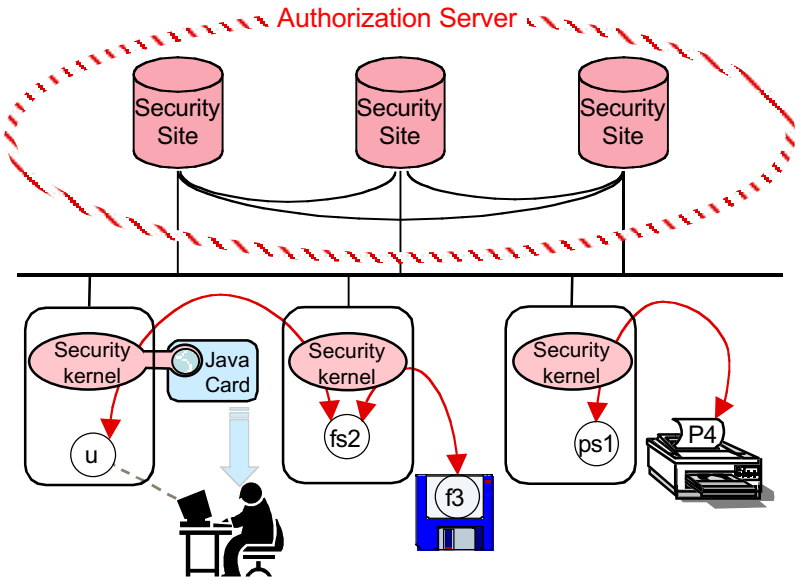

Figure. 1. Authorization architecture

2. Security kernel

There is a security kernel on each host participating in a MAFTIA-compliant application. The security kernel 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 security kernel is complicated since, due to the heterogeneity of connected hosts, it would be necessary to develop one version of the security kernel 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 security kernel, or the same version thereof. This is why we have chosen to implement them by using Java Cards.

2.1 Implementation

A security kernel in the context of a MAFTIA site is composed of a local Java Virtual Machine (JVM), a Java Card and a Dispatcher as shown in figure 2. The Dispatcher is a local object that always runs on each MAFTIA site and that is in charge of dispatching remote requests to the local objects. There is no specific requirement for the local JVM, it may be the JVM of a browser such as Netscape or Internet Explorer for instance. This JVM must simply be compliant with JDK1.2 and thus able to check a signature (for a signed class).

The Java Card 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.

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:

The MAFTIA public key, that we note PK_m (this key is associated to the MAFTIA private key SK_m, which is not stored in the Java Card);

A private/public key pair specific to the Java Card, that we note SK_j, PK_j;

The authorization server public key, PK_{as} (this key is associated to the private key of the authorization server, SK_{as}).

The role of these cryptographic keys is explained in the following paragraph.

2.2 A typical scenario

A distributed MAFTIA application is executed by means of method invocations between MAFTIA objects located on MAFTIA-compliant hosts. The objects are signed off-line by the global MAFTIA private key SK_m, and this signature is checked by the local JVM of the host (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).

Invocations are conveyed between objects by message passing. When an object is authorized to invoke a particular method of a particular object, it invokes the dispatcher of the corresponding site and gives it the capabilities received from the authorization server that authorize this access (see Step 1 of Figure 2).

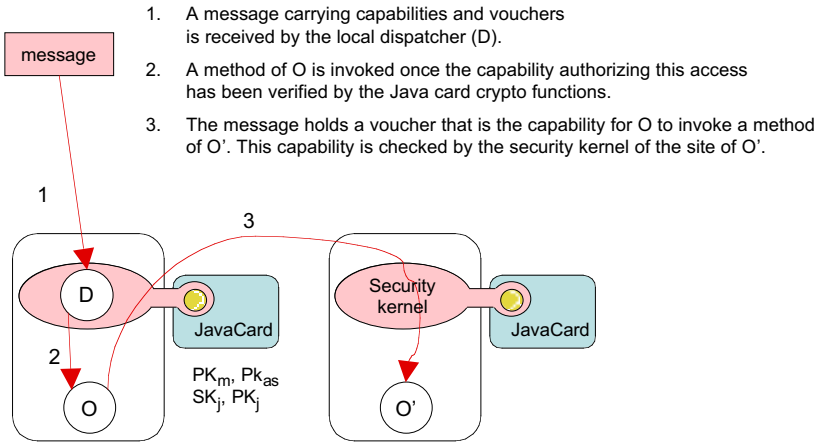

Figure. 2. Example of access control by the security kernel

Capabilities are ciphered by the public key PK_j of the local Java Card and signed by the private key SK_{sa} of the authorization server. The capability signature must first be verified using the authorization server's public key, 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 site is thus controlled by its local Java Card. This verification corresponds to step 2 of Figure 2.

The invocation message may contain vouchers. As explained before, a voucher may be a capability or the authorization to carry out a high-level operation in the system. A voucher is not used by the object requesting the operation but is delegated to another object involved in the realization of the current operation.

If the voucher is a capability, this capability is thus ciphered by the public key of the corresponding Java Card and thus, will be checked by the corresponding security kernel. Step 3 of Figure 2 presents an example of such a voucher.

If the voucher is the authorization to realize a high-level operation, it is transmitted to another object, which uses this voucher in the following way: the voucher is presented to the authorization server as a proof that the object is authorized to realize a particular operation. The authorization server simply checks the validity of this voucher and then delivers to the object the corresponding capability and vouchers.

3 Conclusion

The authorization scheme presented in this paper is flexible, easily managed (at the coarse-grain level of "high-level operations"), and efficient (fine grain access control at the object method invocation level, tamperproof security kernels 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 high-level operation.

References

[Abghour *et al.* 2001] N. Abghour, Y. Deswarte, V. Nicomette and D. Powell, *Specification of Authorisation Services*, MAFTIA Project IST-1999-11583 Contract Report, LAAS-CNRS, N°01.001, Jan. 2001, <http://www.maftia.org/deliverables/D27V13.pdf>.

[Nicomette & Deswarte 1996] V. Nicomette and Y. Deswarte, "Symbolic Rights and Vouchers for Access Control in Distributed Object Systems", in *Proc. 2nd Asian Computing Science Conference (ASIAN'96)*, (Singapour), LNCS n°1179, pp.193-203, Springer-Verlag, 1996.

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