Enforcing History-Based Security Policies in Mobile Agent Systems

Pedro Dias, Carlos Ribeiro, Paulo Ferreira
INESC, Distributed Systems Group
Rua Alves Redol, nº 9, 1000-029 Lisboa
[pdias, cnr, pjpf]@gsd.inesc-id.pt

Abstract

The mobile agent paradigm used in modern distributed systems has revealed some new forms of common security threats, such as abusive resource consumption or illegitimate information flow between different and non-cooperative entities. This problem is aggravated when an agent's host doesn't know anything about the agent's past activities, visited hosts and interactions with other agents. Thus, robust and efficient authorization platforms should be considered in order to avoid undesired actions from malicious agents.

We present an authorization platform designed for a mobile agent system, MobileTrans, which supports the definition and enforcement of history-based security policies, allowing hosts to decide on the authorization of an agent's action upon its past behaviour.

1. Introduction

The mobile agent paradigm [3] has introduced some new concerns on the security area, in particular, in information flow control and authorization.

A major issue on such agent-based applications is the definition of what operations the agent should be authorized to perform and what operations the agent should be prohibited from doing or obliged to do.

Consider the case where an agent travels between two different hosts belonging to two different bank institutions. Suppose that the agent is intended to pick up money from one bank account and then transfer it to the other one. After the first operation, the agent should be prohibited to spend the money in any other way, as it should only have permission to deposit the money in the other bank account.

This kind of scenarios can be accomplished with the use of history-based security policies, which are generally not addressed in the vast majority of mobile agent platforms.

In this paper we present a generic authorization platform designed for a mobile agent system, MobileTrans, which supports the definition and enforcement of history-based security policies. These policies are defined in SPL (Security Policy Language) [1], an authorization language, and are enforced by a security monitor. With such a platform we can define some useful history-based security policies applied to the mobile agent paradigm, such as Chinese Wall [2]. Thus, agent operations may be allowed or denied based on the agent's past behaviour.

This paper is organized as follows. In section 2 we describe relevant work related to authorization models for mobile agent systems. In section 3 we discuss MobileTrans security policies and present SPL. Next, in section 4, we describe the authorization's platform architecture. Finally, in section 5, we present some conclusions and consider future work that may enrich this platform.

2. Related Work

JVM [6], one of the first platforms that considered security support for mobile code, uses a hybrid approach, based on sandboxes and digital signatures. However, its expressiveness is relatively poor, since it doesn't support negative permissions or disallowances and does not support the definition of user groups and group hierarchies, which are necessary for definition of RBAC policies [11].

The JSEF framework [7] was developed with the objective of solving JVM's limitations, allowing the definition of a hierarchical security policy scheme and the definition of complex entities such as groups and roles. Although richer than JVM security policies, it doesn't offer any support for history-based security policies.

Deeds [4] supports an history-based access control mechanism that protects local resources from mobile code. Although flexible, in the sense that it may enforce many different history-based policies (handlers), Deeds is hard to manage because the programmer has to individually implement all the handlers. Besides, Deeds only focus on host protection, ignoring mobile code protection.

Ponder [8] provides a general-purpose deployment model for security and management policies. Its declarative language is able to specify some generic and complex security policies such as RBAC policies.

Although Ponder supports different types of policies, such as permissions, refrains, delegations and obligations, this platform doesn't address history-based security policies.

Aglets [3] presents a security architecture that protects both agents and contexts, which are execution environments for agents. This protection is based on security policies that both aglets and contexts may specify, although policies depending on history are not supported.

In [10] it is presented a model that controls agent mobility with specific mobility policies. Using Ponder obligation policies it allows, for instance, an agent to migrate to other host when the current's host CPU usage is above a given level. Although supporting obligation, this model doesn't support history-based policies.

JavaSeal [5] offers a security architecture that protects both execution environments and agents. However, this platform is extremely restrictive since agents are organized in a hierarchical tree and communicate through messages between neighbour agents in the tree. A message sent by a mobile agent to another, located in a remote node of the tree, may not arrive to the destination if an intermediate agent doesn't allow it; even when both sender and destination agents agree on cooperating.

Other systems, such as Ajanta [9], base their approach essentially in cryptographic mechanisms.

Although covering a lot of different techniques, all these platforms fail to provide a generic and modular support for history-based security policies.

3. Policy Definition

Although MobileTrans supports numerous types of policies, such as DAC, RBAC, or obligation policies, in this paper we will only address the definition and enforcement of history-based security policies.

The MobileTrans platform uses SPL for policy definition [1]. The SPL language is based on four essential blocks: entities, groups, rules and policies. The rules establish constraints through the relations between entities and groups, while policies result from the composition of multiple rules and groups. This language is therefore policy-oriented and constraint-based.

The SPL entities are typified objects with an explicit interface, through which their properties can be obtained and modified. These entities may represent not only internal authorization model objects, but also external platform resident objects. Although there are some internal entities like groups, rules or policies, the vast majority are external entities, such as mobile agents or files.

Each external entity has an associated type. That type is used to define its interface and subsequent properties. Figure 1 shows the definition of two useful types in MobileTrans: *object* and *mobileAgent*.

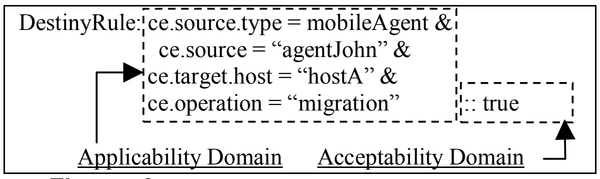
Another important SPL entity is the rule. Rules are entities that establish constraints to the authorized operations. An authorization policy may therefore be expressed in terms of a set of rules, which are three-value logical expressions. They may assume the following values: allow, deny and not apply. These values decide the acceptability of the events that are generated within the MobileTrans platform.

```
type object{
        string name;
        user owner;
        string homeHost;
        number timeOfCreation;
}
type mobileAgent extends object{
        boolean running;
        string group previousHosts;
}
```

Figure 1. Definition of types object and mobileAgent.

To enforce history-based security policies there are two crucial classes of events: the current event and the past events. The first one is the event that is being checked and over which approval is requested. The second type of events are already approved or refused events that constitute the knowledge basis for approving or refusing the current event.

A rule is, in SPL, composed by two logical expressions. The first one defines the applicability domain of the rule, while the second expression sets the acceptability domain. Figure 2 illustrates a MobileTrans rule written in SPL.



```
DestinyRule: ce.source.type = mobileAgent &
             ce.source = "agentJohn" &
             ce.target.host = "hostA" &
             ce.operation = "migration"
             :: true
```

Figure 2. Definition of a SPL rule. This rule is applicable to all migration requests to host A that are generated by the mobile agent agentJohn. The acceptability domain is always true, so the event is

A SPL policy is a set of rules and groups that determine the authorization and prohibition of a given event. From the complete set of rules, only those with a true applicability domain will have their acceptability domain checked.

Consider the implementation of a Chinese-Wall policy. In this situation suppose we have a single class of interest that contains multiple hosts. In this scenario any agent that has already been executed in a given host, will be denied access to any other host in that class of interest. Figure 3 shows this policy specification in MobileTrans using SPL.

This policy defines one parameter, InterestClass, as a group of hosts. This parameter will hold the hosts integrating the same class of interest. The policy behaviour is given by the ?AgentChineseWall rule. This rule looks in past events if there is any event of a migration operation to any other host in that interest class. If one such event is found then the current request is denied.

```
policy AgentChineseWall( host group  interestClass) {
?AgentChineseWall:
   Exist e IN PastEvents {
      ce.target IN InterestClass &
      e.target IN InterestClass &
      ce.operation = "Migration" &
      e.operation = "Migration" &
      ce.target != e.target    :: false
   };
}
```

Figure 3. Simple Chinese-Wall policy in MobileTrans.

Mobile agents must also be protected from interactions with other agents. Consider the simple scenario where mobile agent *Joe* offers two services, provided by methods *M1* and *M2*. *Joe* is very cautious and therefore does not allow some agents, such as agent *Bill*, to access the service provided by *M2* if it has already accessed service *M1*. Figure 4 shows how this policy could be specified in MobileTrans. In this policy we define an applicability domain that checks whether the current event refers to an invocation of *M2* made by *Bill*. If so, the acceptability domain assures that the current event is allowed only if *Bill* has never tried to invoke *M1* before.

```
Policy HistoryBasedServiceControl {
?HistoryBasedServiceControl:
   Not Exist e IN PastEvents{
      ce.source = "Bill" & ce.operation = "Invocation" &
      ce.operation.method = M2 ::
      e.source = "Bill" & e.operation = "Invocation" &
      e.operation.method = M1
   };
}
```

Figure 4. SPL security policy between two agents.

4. Architecture

An agent in MobileTrans is created and started by an application that interacts with the agent through a proxy, called home proxy. This proxy remains local to the application, no matter where the agent resides, offering, among other facilities, agent location independence to the application. To interact with its agents the application doesn't need to know where they reside. It only needs to interact with the corresponding proxy, which will then forward the requested operations to the appropriate agent. These operations may be either the agent's proper methods or platform services. The MobileTrans platform supports a large set of services, such as agent migration or replication.

Cooperating with an agent's home proxy there is a mobile proxy, located in the same host where the agent is being executed, that is able to manage the agent's execution flow. The mobile proxy acts as an extension of the home proxy in the remote host. Contrarily to the home proxy, the remote one is mobile and travels between hosts, staying permanently with the agent. The remote proxy creation is requested by the home proxy to the destination host when a migration is firstly requested. The proxy creation is based on the agent's code and associated security policies. The home proxy is in turn generated by an automatic tool provided by MobileTrans, the MobileCodeg (Mobile Code Generator). The programmer first implements the agent's code. Next, the user, possibly a different one from the programmer, may define a set of security policies, creating for that matter a policy file in SPL. With this file and the agent's code the programmer may then run the MobileCodeg to create the home proxy. The home proxy may then be used by any application.

4.1. Proxy's Architecture

The home proxy's architecture is composed by four modules: a reference to the agent's code, an event listener, an event history and an authorization monitor (Figure 5).

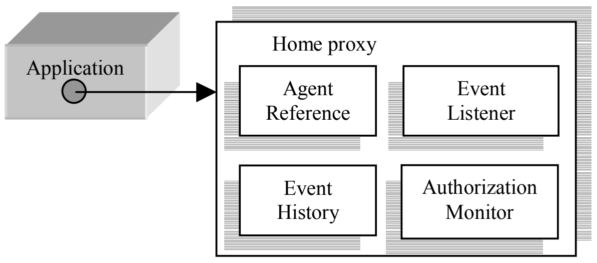


Figure 5. Home proxy internal architecture.

The agent's reference allows access to the agent's code and data that implement the agent behaviour. This reference is deleted when the agent migrates to a remote host, as a consequence of the transfer of the agent's code and data to that host. A new reference for the agent is then created at the mobile proxy in the new agent's host. This proxy, upon receiving the agent, starts an execution flow with one of its methods. The agent's execution is then moved from the original machine to the mobile proxy's machine.

The Event Listener is the responsible module for the events handling, such as migrations, file system accesses, or requests between agents. This module interacts with the authorization monitor, informing it that a new event that needs to be authorized. This module is also mobile since it stays always together with the agent's code and data.

The Event Listener collects four different kinds of events: platform events, operating system (OS) events, application requests and agent requests. The first ones are generated by our platform, MobileTrans, and are a result of agent operations, such as migrations or replications. OS events are generated by accesses to system resources, like local disk files or network ports. Application requests are invocations to agent methods that may control their execution flow, such as stop or resume. An agent request, on the other hand, is an invocation from an agent to another agent, which allows, therefore, agent cooperation.

The Event Listener may then create an event structure, filling in all the necessary event properties. The authorization monitor will later check these properties in order to decide about the current event acceptability.

Once verified the acceptability of a given event, it will be forwarded to the Event History, which checks if it is necessary to log it. For optimisation purposes there are events which are not logged. The Event History checks if the authorization monitor, to enforce its policies, will ever need this event. If the answer is negative, then the event is discarded and will not be logged, keeping the event history as small as possible. Consider, as an example, that an agent has a single policy and suppose that it only checks events related to migration operations. In this case there is no need to collect events involving files or network ports.

4.2. Migration Semantics

Although optimised, the Event History may become relatively large. For that reason MobileTrans supports two different ways of migrating an agent: complete migration and partial migration. In the first one, all modules at the home proxy are transferred to the destination host. In the partial migration only the agent itself (code and data) and the Event Listener are migrated to the remote host. The Authorization Monitor and the Event History remain local to the home proxy. Note that there is no advantage in separating the Authorization Monitor and the Event History since the frequency of accesses between them is considerably high. While the first one accesses the second in order to search for past events, the Event History accesses the Authorization Monitor to check if a given event will ever be needed, which allows the security platform to discard useless events.

5. Conclusions and future work

The MobileTrans is a mobile agent platform that supports the definition and enforcement of security policies. In this paper we have presented an agent architecture to support history-based security policies. Our architecture is modular and extensible to support new security semantics such as obligation or role-based policies. Its authorization model is very flexible. The agent's code and the agent's security policy may be independently developed, allowing two different users, with different concerns, to cooperate in the agent creation.

The mobile agent generation process is also very simple. The MobileCodeg generator automatically compiles the SPL policies and simultaneously creates the necessary proxies for the applications.

We have further considered two optimisations that increase the platform's efficiency: i) the event logging is filtered in order to prevent logs to overflow, ii) the migration process can be accomplished in two distinct ways allowing some structures to stay local to the application, even when the agent is migrated.

As future developments our main efforts will reside on guaranteeing events integrity.

References

[1] C. N. Ribeiro, A. Zúquete, P. Ferreira and P. Guedes, "SPL: An Access Control Language for Security Policies with Complex Constraints", in Network and Distributed System Security Symposium, NDSS' 01, February 2001.

[2] D. F. C. Brewer and M. J. Nash, "The Chinese Wall Security Policy", published at IEEE Symposium on Research in Security and Privacy, pages 206-214, May 1989.

[3] G. Karjoth, D. Lange and M. Oshima, "A Security Model for Aglets", IEEE Internet Computing, pp 68-77, July-August 1997.

[4] G. Edjlali, A. Acharya and V. Chaudhary. "History-based Access Control for Mobile Code", in Proceedings of the 5th Conference on Computer & Communications Security, 1998.

[5] J. Vitek and C. Bryce, "The JavaSeal Mobile Agent Kernel", in the Proceedings of the Joint Symposium on Agent Systems and Applications and the Third Symposium on Mobile Agents, 1999.

[6] L. Gong, M. Mueller, H. Prafullchandra and R. Schemers, "Going Beyond the Sandbox: An Overview of the New Security Architecture in the Java Development Kit 1.2", in Usenix Symposium on Internet Technologies and Systems, 1997.

[7] M. Hauswirth, C. Kerer and R. Kurmanowytsch, "A Secure Execution Framework for Java", in Proceedings of the 7th ACM Conference on Computer and Communications Security, pages 43-52, 2000.

[8] N. Dulay, E. Lupu, M. Sloman, N. Damianou, "A Policy Deployment Model for the Ponder Language", in Proceedings of the IEEE/IFIP International Symposium on Integrated Network Management, 2001.

[9] N. M. Karnik and A. R. Tripathi. "A Security Architecture for Mobile Agents in Ajanta", in Proceedings of the International Conference on Distributed Computing Systems, 2000.

[10] R. Montanari and G. Tonti, "A Policy-Based Infrastructure for the Dynamic Control of Agent Mobility", in Proceedings of the 3rd IEEE International Workshop on Policies for Distributed Systems and Networks, June 2002.

[11] R. S. Sandhu, E. J. Coyne, H. L. Feinstein and C. E. Youman, "Role-Based Access Control Models", in IEEE Computer, volume 29, Nº 2, pages 38-47, February 1996.