The Security Architecture of the M&M Mobile Agent Framework

Paulo Marques[*], Nuno Santos[*], Luís Silva[*], João G. Silva[*]
CISUC, University of Coimbra, Portugal

ABSTRACT

In the Mobile Agent programming model, small threads of execution migrate from machine to machine, performing their operations locally. For being able to deploy such a model into real world applications, security is a vital concern. In the M&M project we have developed a system that departures from the traditional platform-based execution model for mobile agents. In M&M there are no agent platforms. Instead there is a component framework that allows the applications to become able of sending and receiving agents by themselves in a straightforward manner. In this paper we examine the security mechanisms available in M&M, and how integration with existing applications is done. One difficult aspect of this work is that all the features must work with the security mechanisms that already exist on the applications. This is so because the components are integrated from within into the applications, which already have security mechanisms in place. Currently, M&M provides features like fine-grain security permissions, encryption of agents and data, certificate distribution using LDAP and cryptographic primitives for agents. For validating the approach and solutions found, we have integrated the framework into several off-the-shelf web servers, having the security mechanisms running, with no problems.

Keywords: Mobile Agents, Security, Agent Platforms, Components, Integration

1. INTRODUCTION

Over the last few years there has been a large body of investigation on a new model for distributed systems programming. In this model, called "Mobile Agents", small threads of execution migrate from machine to machine, performing their operations locally[1]. Security is one of the key problems in mobile agent systems. The success or failure of the mobile agent paradigm is directly connected to the question on whether proper security mechanisms can be effectively implemented and used. This is especially important in this paradigm since users are not much predisposed to allow unknown, not trusted programs to run on their machines. The same applies to any company operating on an open environment like the Internet. There is the danger of resource stealing, denial-of-service attacks, data spying and many other problems[2,3].

During the last two years, we have been working on a project that tries to overcome some of the limitations found in terms of programmability and usability of the mobile agent paradigm[4]. In the M&M framework there are no agent platforms. Instead applications become agent-enabled by using simple JavaBeans components[5]. The applications can be developed using current object-oriented approaches and become able of sending and receiving agents by the simple drag-and-drop of binary software components. Different services are implemented in different components that can be selectively used when programming the application. In addition integration with ActiveX is provided, which enable applications written in other languages like Visual Basic, Visual C++ or Delphi to receive and send agents in a transparent way.

One of the key issues in our framework is that the agents arrive and departure directly from the applications. Thus, the security mechanisms associated with the framework must closely integrate in the security architecture already defined by the application. For instance, we have used our framework to agent-enable some off-the-shelf web servers. This

[*] {pmarques, nsantos, luis, jgabriel}@dei.uc.pt; phone +351 239790000; fax +351 239701266; Dep. Eng. Informática – Polo II, Universidade de Coimbra, 3030 Coimbra, PORTUGAL

evolves wrapping our components inside of a servlet[6] that is included on the web server. Because the web server is not modified and already has a security architecture up and running, our framework must closely integrate with the existing system, without modifying it.

This paper gives a two-fold discussion. First, the security requirements needed for distributed mobile agent systems are analyzed and related issues argued. Then, the M&M security architecture and integration approach is explained, as well its strengths and limitations.

Thus, the rest of this paper is organized as follows. Section 2 relates to the security issues in mobile agent systems. Section 3 discusses in detail the security features on the M&M framework. Section 4 concludes the paper.

2. ISSUES AND REQUIREMENTS

2.1. Open vs. Close Environments: Different requirements

When discussing security in distributed applications that use mobile agents, two different scenarios must be considered. The first scenario consists in deploying an application that uses mobile agents in a close environment. What this means is that it is possible to identify a central authority that controls the nodes of the network where the agents are deployed. For example, a network operator may deploy an agent platform on its network, for being able to deliver new services in an easy and flexible way. Although different users, with different permissions, may create the agents, the key point is that there is a central authority that is able to say who has what permissions. A completely different scenario is having the agents deployed in an open environment, where the agents are deployed on different nodes controlled by different people, possibly having very different goals. One classical example of this situation is an e-commerce situation on the Internet. Different sites may deploy an agent platform, allowing agents to migrate to their nodes and query about products and prices, and even perform transactions on behalf of their owners. Here the sites will be competing against each other for having the agents making the transactions on their places. There is no central authority, and each site may even attack the agents, stealing information from them, or making them do operations that they were not suppose to do. Probably the most important point is that in an open environment there may be competing nodes with different goals, which may be hostile towards the agent. This changes the requirements that the security infrastructure must be able to support.

Table 1: Different application domains run on different environments

Type of Environment	Application Areas
Closed (All nodes belong to the same authority)	Network management Telecommunication applications Software distribution and upgrading Parallel/Distributed processing Groupware
Open (Nodes may belong to different authorities)	Electronic commerce Information gathering and filtering Information dissimilation Distributed processing Messaging

Although it may appear that deploying applications on close environments is too restrictive, in fact there is a very large number of applications that is worth deploying in such setting. Table 1 summarizes some application areas that are notorious of open and closed environments.

Most of the applications in the area of network management, telecommunication applications, flexible software distribution/upgrading and groupware are based on the closed network model. All the hosts belong to the same authority and they will not attack or spy agents since there is no interest in it. The agents belong to the same entity and try to achieve a common goal in the closed distributed system they operate on. When considering applications for open

networks like the Internet, security assumes a whole new dimension. Firstly, hosts must be protected from attacks of malicious agents. Also, agents must be protected from attacks of other agents. Equally important is to assure users that when they deploy agents, their agents will not be tampered with or spied while they execute. If an agent carries a credit card number, this number should only be disclosed and used where the agent (and its owner) decides so. In the same way, if an agent is shopping for flight tickets, it should be able to execute without being coerced by malicious hosts to take offers it does not want to[3]. This can happen in several different ways[7]. The code of the agent may be tampered with, making it take an offer although it was not in its best interest to do so. The flow of execution of the agent may be changed so it takes the offer no matter what. Also, the data of the agent may be changed so it thinks that the offer being made is the best of all.

We will now expand a little more the view of open vs. closed environments into a more concrete a realistic setting, which we call "agent-accountable environments". This is the type of environment that the security framework of M&M supports.

2.2. Agent-accountable environments

In the current state of the technology, it is not currently feasible to deploy an agent system in a completely open environment. The problems associated with attacks to the agents by the executing environment are still far from solved. Although there are some promising approaches like computing with encrypted functions[8] and use of tamper-proof hardware[9,10], the state of the technology is nonetheless in its infancy. The computational overhead in using encrypted functions is too high, and not fully general in its capacities. Also, the currently available tamper-proof hardware is not powerful enough to attend to the needs of such a system, at least the most common off-the-shelf hardware.

Nevertheless, the need for security in mobile agent systems is fundamental. Currently mobile agent systems do exist, so in what kind of environments is it possible to deploy them? We define an "agent-accountable environment" as a set of cooperating organizations that deploy a mobile agent infrastructure for supporting their operations. These organizations have some sort of binding contract in which they compromise themselves not to attack any agent that execute on their premises. The key point is the assurance is that the infrastructure itself is not malicious, while the agents can be (either because their programmers/owners are, or because they are badly programmed). We say that the environment is "agent-accountable" because each agent is obligatory identified. Different agents execute with different permissions and access rights.

This type of environment is a mix between the closed environment, where all agents and infrastructure belong to the same authority, and the open environment where the agents and infrastructure belong to different entities. In this type of environment, the security framework must deal mostly with protection of the systems from the agents, confidentiality of execution and data, and accountability of the agents over their actions.

Let's consider an example of such a scenario. On the Internet, a large consortium that owns several companies that sell many different types of products decides to setup a mobile agent-based business-to-business (B2B) infrastructure. This particular company allows other companies with which it has business to migrate their agents into its infrastructure, so that they can lookup products and prices, launch alerts when certain products arrive, drop prices, and so on. This is certainly the kind of service that will be welcome by its costumers since allows them much more flexibility that the fixed logic of a static B2B e-commerce site. In the agent scenario, the costumers can deploy some costume-made logic for looking for things they are interested into. From the viewpoint of the large company, they are just setting up an infrastructure that allows them to interact more easily with their costumers. They do not intend to attack the costumers' agents or extract information from them. Nevertheless, they want to be sure that if a costumer misbehaves, it should be held accountable, and actually impeded of misbehaving. Also, some costumers should be granted more privileges than others. Now taking the customers' point-of-view, they must decide if they trust the major company or not. The company may attack their agents, or steal information from them. If a costumer feels that the risk is unacceptable, then it does not use the infrastructure, or deploys very limited agents. If the costumer trusts the company, then if can deploy more powerful, intelligent information rich agents.

Although it may seem that we are tying to escape the general security problem, we argue that this is the security scenario that already happens on the Internet today. We already decide whether we trust certain sites or not. If we do, for instance when we are shopping for a book, we give them our address and credit card number, and they keep it. We

are happy to live with that. If we do not trust them, we give them nothing. Thus, the situation is analogous to the mobile-agent one. What confuses things most in the mobile agent approaches is that we have the idea that the agents will migrate to arbitrary sites, which we do not know anything about. In fact, it would not be very intelligent to allow agents carrying extremely sensitive information or algorithms roam through arbitrary places on the Internet. That is not a very realistic scenario, even if more or less complete security assurances could be made.

Let's now consider the situation where multiple possible hostile parts are involved. In this type of scenario we believe that the dangers are much more of the hosts trying to manipulate the agents into buying something they would buy in some other party, than of stealing information. The point is that if a user knows that he is going to deploy an agent in a possible hostile environment, it will not give the agent much confidential data or algorithms. Also, the user will not allow the agent to migrate to arbitrary unknown hosts, which would be naive. Typically, the agent will migrate to sites that its user knows and probably already have the information that would be possible to steal from the agent. Our view is that at this level, the risks are not very high and acceptable. We believe that in this scenario the major risks are of the sites examining the state and code of the agent, gathering information about previous offers made by other sites, and manipulate that information for their benefit. For instance a host may delete a previous offer that was better than the one it can offer, or change another offer to a value larger than what it will offer, or even make an offer that is just slightly less than the lowest offer currently on the state of the agent.

When competing organizations are involved we take the position of Farmer[11], which states: *"An agent's critical decisions should be made on neutral (trusted) hosts"*. In fact, we believe that the extreme view that it is not possible to trust any host on the Internet is not realistic. Users can establish contracts with providers that give them trusted computing bases for performing operations in a safe way. For instance, Internet Service Providers (ISPs) and Application Service Provider (ASPs) already make such contracts today for deploying client/server software and web sites, many containing confidential and vital information. An example of such a futurist agent-based scenario could be a secure marketplace operated by an ASP where several bookstores and disk stores announce their offers, and the user agents interact with them.

Nevertheless, it is necessary to give the agents assurances that information gathered on a certain host may be safely stored and not tempered with. The agents must be able to gather information in unsafe places, being sure that these places cannot see what others offered or be able to delete and add arbitrary offers. This implies using trusted hosts for making their decisions. Algorithms for giving this kind of assurances do exist[12]. Thus, in a mobile agent framework, it is desirable to give the programmer a set of primitives that allow the easy implementation of such protocols, if not even an implementation of the protocols themselves.

To conclude, our opinion is that although it is important to have a framework that not only completely protects the hosts from the agents and holds them accountable, it is also important to protect the agents from attacks of the hosts, and allow them to execute with privacy and confidentiality. Nevertheless, this system is not currently possible of obtaining because of technological and theoretical issues, not yet completely resolved. Nevertheless, for many application domains it is possible to design a framework that allows applications based on mobile agents to be built allowing the agents to execute with strong security guaranties, and at the same time protecting the hosts from the agents. We designate these environments as "agent-accountable" environments, which have the following characteristics:

- The agents are well identified and can be hold accountable for their actions[*]
- The agents run under strict permissions according to their identity
- The actions of each agent are monitored and registered
- The agents do not carry (should not carry) critical vital information, with cannot be compromised to the host where the agent is going to run
- The agent should be able to carry private information stored by different hosts, not accessible to anyone else
- The agents makes critical decisions on trusted parties

[*] As we will see later, this does not necessarily means that the end-users of the agents are identified. What this means is that it is possible to uniquely identify and agent, and give it a set of strict permissions for executing – i.e. not all the agents are created equal. For instance, an agent running using a "guest" identifier as completely different permissions from a "sysadmin" agent.

We believe that are currently more types of applications that fell inside of this set of restrictions than applications that need the complete set of security features required for arbitrary open environments. The good news is that it is possible to build a framework that supports this scenario.

We will now examine in detail the security requirements needed from an agent framework in order to deploy applications in an agent-accountable environment.

2.3. Requirements

In order to build a secure system, first it is necessary to assess what kind of attacks the system may be victim of and whether it is best to tackle the problem, or withstand the consequences of the attack. For instance, a company may have a dial-up access router where a particular vulnerability is detected: in certain cases, it may be easy for a hacker to crash the access router. If the access router is not used a lot in the company, it may make more sense to once in a while suffer an attack than to spend a lot of money buying a new one. It is all a question of risk vs. value.

The attacks in a mobile agent system may be viewed in two dimensions: a) attacks against the infrastructure itself; b) attacks against the agents. The attacks may themselves be considered passive or active. Passive attacks relate to the stealing of information, and may even go undetected. Active attacks involve changing data structures, flows of execution, or compromise in any way the normal operation of the system.

Let's now consider what is required on an agent-accountable environment. First, the agent runtime must be protected from attacks of agents:

- Agents may try to execute in a runtime that they are not entitled to
  (e.g. agents from competing parties accessing each other's infrastructure)
- Agents may try to perform operations on a runtime that they are not authorized to
  (e.g. a guest agent trying to shutdown the agent platform)
- Agents may try to access information that they are not allowed to
  (e.g. a guest agent trying to obtain the private key of the platform where it is)
- Agents may try to use excessive resources like CPU time, threads, memory, disk and network bandwidth
  (e.g. a malicious guest agent trying to launch a denial-of-service attack using all available CPU cycles)
- The system must be protected from being overflowed by agents
  (e.g. too many agents trying to migrate to runtime at the same time)

Also, the external interfaces of the agent system must be protected. For instance, it should not be possible for an arbitrary party to call the external management interface of an agent platform, requesting that all running agents are killed.

By using proper authentication and authorization mechanisms, most of these types of attacks can be prevented. Cryptographic mechanisms, access control lists (ACLs), resource control mechanisms and logs play a fundamental role here[2].

Another important point it to protect the agents from attacks by other agents. For instance, on a badly designed system, a malicious agent could try to kill all other agents running in the environment, possibly resulting in great loss. For preventing these types of attacks, three basic mechanisms accomplish a great deal: a) use of secure languages that isolate the system resources and address spaces that agents can access; b) restrict communication between agents to high-level semantic messages; c) use of proxy objects that decouple the objects/messages from the actual flow of execution of the agents.

Finally, although it is not possible to completely protect the agents from attacks by the hosts, it is desirable that the agents have a set of cryptographic primitives that allows them to safely collect data from different hosts, maintaining the privacy of those offers.

3. SECURITY IN THE M&M SYSTEM

Before discussing the security features of the M&M framework, we will start by giving an overview of its architecture. The interested reader can consult additional documentation for obtaining more complete information about the project, its goals and implementation details[4,13].

3.1. M&M Overview

In typical available mobile agent systems, all the functionality is coded in a monolithic infrastructure called "platform". All the functionality is tightly coupled in a single application that provides all the resources and features needed by the agents. This approach has several disadvantages, from which we highlight:

- It is hard to extend. Creating new features, or adding new services requires that a new platform is built and deployed. For instance, it is not easy to add a new inter-agent communication mechanism to an off-the-shelf platform, or a persistence service that stores agents. This quite limits the ability of the programmers to develop applications that need a particular feature, or makes them code work-arounds that are error-prone and unnatural.
- Interacts badly with other middleware and with existing applications. In typical platforms the applications are the agents. Everything is coded around the agents. The agents are not just another part of the distributed applications, but its corner stone. Many times, if it is necessary to use other middleware, that is simply not possible because everything there is are agents, and there are strict limitations on what agents can and cannot do.
- It is hard to program. Programmers want to code their applications as they currently do, using best practice OO methodologies. They do not want to center all the development on the agents, or the agent platforms. That is exactly what the current systems force the programmer to do. The agents are not just middleware but are the central frontware.

The M&M framework tries to overcome these limitations and others by departuring from the classical platform-based model. In this framework there are no agent platforms. Instead applications become agent-enabled by using simple JavaBeans components. The applications can be developed using current object-oriented approaches and become able of sending and receiving agents by the simple drag-and-drop of binary software components. Different services are implemented in different components that can be selectively used when developing the application.

In the M&M framework there is a central component that implements the support for agent migration and management, and the basic infrastructure needed for extensibility. Other components can connect to this component offering higher-level functionality. The central component, called `MobilityComponent`, provides an event-based model that offers all the functionality needed for the extensibility of the system.

3.2. Security in Java, the Good and the Bad

The M&M framework, as well as most of the currently available mobile agent systems today, is written in Java[14,15]. This language is so used for building mobile agent systems because many of its characteristics make it quite easy to develop a basic mobile agent system, when compared with the work needed in other languages. Probably the two most important characteristics that allow this are the dynamic class loading facility and the object serialization features. Also, because Java is so ubiquitous today, having many powerful APIs readily available, Java makes the perfect candidate for coding this type of systems.

Nevertheless, using Java brings some hard problems in terms of security, which apply to our framework but are also applicable to the generality of the systems written in Java.

The first problem is that Java was never thought to be used as it currently is. Basically, when designing a mobile agent platform, we are asking that the platform acts almost as an operating system. Java has a strong notion of thread, but has no notion of process. All classes are loaded in the same JVM, and stay there. This brings a lot of problems. For instance, if it is necessary to kill an agent thread, and if it has a lock to a system monitor or semaphore, all threads (or agents) in

the system can deadlock. Actually, currently there is no standard and correct way of killing a thread in Java, which is fundamental for these systems.

Java was designed for a single-user environment where someone would run some code/application in the JVM. There isn't any resource control mechanism in Java that allows the system to control the resources each thread is using, and therefore each agent. What this means is that it isn't possible to detect the exact agent that is using too many resources, either because it is badly coded or because is performing a denial-of-service attack. At the same time, shutting down the whole system kills every agent, which is unacceptable.

Finally, although Java currently provides a fine-grain security architecture, the system itself has no notion of user. Authentication and authorization is based on who signs the code and where it came from but, there is no way to distinguish code to be run on the name of person A, from code to be run on the name of person B. Again, this is a consequence of Java having been though for a single-user environment. This has serious implications since in most cases it is required to associate the execution privileges of an agent with the user who starts the code and not with the person who signed it.

Other important problem has to do with middleware. One important trend nowadays is the use of software components. A programmer who needs a certain functionality just drags-and-drops a component from a component palette, configures some properties and events, and everything is done. Let's now suppose that two different components need to perform different security checks. In a traditional software engineering setting, if there are different security requisites in two different modules of a system, the programmer just modifies the Java security manager to accommodate both needs. But in this middleware case, another person coded the components. If one of the components needs to use a security manager and another one needs to use another, then there is a problem: Java only allows one security manager to be instantiated at one time. It is also not feasible to say that the programmer will recode the security manager to attend the needs of the components. Fortunately, since the Java 2 Platform[15] has been launched, people now seldom recode the security managers, since the new model is based on permissions. Nevertheless, the problem still exists, since if someone re-codes and uses a security manager such that it is not compatible with the Java 2 model, the middleware components will not work. Actually, these re-coding situations are not so uncommon as one might think.

To resume, the overall problem is that people are currently trying to make the JVM behave as a low-level processor, and the Java platform as an operating system. This clearly has consequences. Actually, these problems are not unique of mobile agent systems. Enterprise Java Beans servers, Servlet and JSP engines, even Integrated Software Development (IDE) environments suffer from these problems.

So, why do people keep coding mobile agent systems in Java?

- It is a lot more easy than to do it in other languages
- It makes it easy to integrate with other software and there are many off-the-shelf APIs available
- It increases the likelihood that other people will use the systems

The consequences are that some things are currently impossible of doing in Java systems, and others require difficult work-arounds. Clearly Java has been evolving into incorporating more and more operating system-like features, and we believe that this trend will continue. Nevertheless, the current state of things clearly limits what is and what is not possible to do in Java.

3.3. The Security Architecture of M&M

Let's now examine the security model of the M&M framework. In M&M the security responsibilities are divided between three modules: the basic `MobilityComponent`, the `SecurityComponent`, and a standard Java 2 `SecurityManager`.

As it was previously said, the `MobilityComponent` provides the basic infrastructure for agent migration and management, and an extensibility mechanism. Because this component is in the core, some of the security features of the framework must be implemented within it. For instance, allowing secure migration of the agents requires that the migration support of the component be responsible for that task. Any security functionalities that are not in the core of the system are delegated on the `SecurityComponent`. This component is responsible for tasks like public/private

key management, user/host authentication and authorization and so on. Finally, for most of the security features of the framework to work, a standard security manager must be instantiated. This allows the permission-based mechanism of Java to be used.

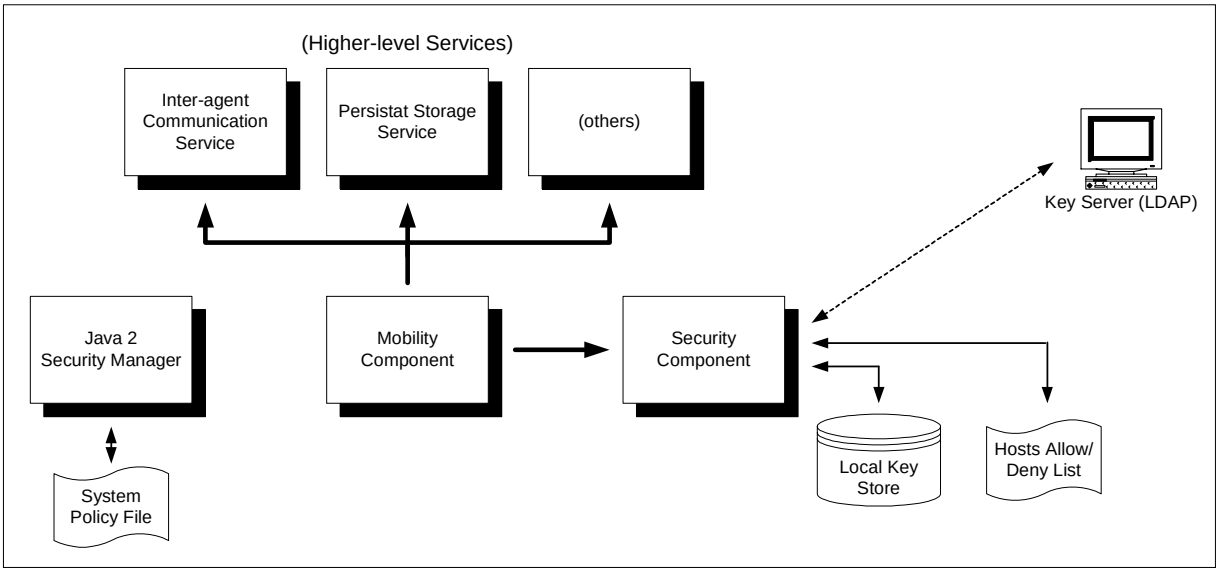

Figure 1: The security modules of the M&M framework

Because in M&M the components are running inside of existing applications, we were especially careful to make the framework work with any security manager that conforms to the Java 2 security delegation model. For using security, we only require that such a security manager is instantiated. We were also very careful to develop the several security features in a very orthogonal way so that if one of the features is not needed, its corresponding modules are not instantiated, thus not imposing any burden on the system. The programmer must only pay for the features that are actually used in the application he is developing. Figure 1 shows the main modules that are part of the security architecture of M&M[*].

Let's now take a closer look at the security features of the framework.

3.3.1. Authentication/Authorization

In M&M there are three types of principals: the agent programmer, the agent owners and the communicating hosts. Each of the principals has a private/public key pair, which is stored on a local key store. When a user creates an agent, it must give its password for the system to create the agent with an identity that is uniquely connected to that user[+]. When an agent is created, it is associated with an `AgentIndentity` object, which contain the following information:

- List of owners
- Name
- Hash signature of the code of the agent
- Creation time
- Expiration time

This object is signed with the private key of its owner so that it cannot be tampered with. It is important to note that only the local key store contains the private key of the user. Thus, the user can only create agents in the nodes where it

[*] Note that the inter-agent communication component and the persistent storage component are only shown as examples. They are not required for the security framework to work.

[+] Actually, it is possible to create an agent with more than one owner. This is useful in certain situations where the agent should have permissions of a different set of people.

is registered. Also, each key in the key store is protected by a password so that different users cannot create agents with tampered identities.

In terms of key distribution, when the security framework needs a certain public key it tries to find it in the local key store. If it fails and a key server is configured in the component, it securely accesses an LDAP server, which stores the public keys for the whole system. The public keys are cached locally. The private keys are stored manually on the local key stores, and are never automatically transferred between systems[*].

There are two different types of authentication. First, when a remote host whishes to transmit an agent, the host is authenticated. M&M also support a host allow/deny list that implements host-level mode of authorization. It is possible to configure the system to receive agents only from a certain set of hosts, or to deny that certain hosts send agents. The second level of authentication has to do with the agents. When an agent tries to migrate into a host, before the agent is sent, the system receives its identity. The `SecurityComponent` is then asked if the current agent can migrate. The component checks the global policy file, and allows or denies the arrival of the agent. After the agent has arrived, but before it starts executing, its code signature is checked, and the agent is associated with the proper permissions of its owner and also programmer, as specified in the global policy file.

This last part is tricky. Since Java has no notion of user, we had to use a work-around. The permission system of Java is based on the notion of code signer, and code source. The code signers are the persons who used their private keys to sign the code that will run on the system. The code source represents the location from where the code was downloaded. When a class loader loads a class, it creates a `ProtectionDomain` with the list of all the signers of the code, and also its source. When this code tries to perform a sensitive operation, the currently instantiated security manager uses the associated `ProtectionDomain` object to see if it has the appropriate permissions, against the policy file of the system. If it does, access is granted. If not, an exception is thrown. What we did was to modify our class loaders so that when the code of an agent is loaded, its protection domain will include not only the actual signers of the code (which may be none), but also a set of "virtual signers" which represent the owners of the agent. Thus, when an agent tries to perform a sensitive operation, its permissions are directly connected to the permissions that its owners are granted on the current system. At any given moment, any agent in the system is always properly authenticated, and has no permissions beyond the permissions associated with its owners and code signers.

Let's now consider how the agents view the world, when they are inside of M&M. There are two operations that the agents can perform. Firstly, they can make direct calls to the Java library (e.g. open a socket, open a file, get the current time), performing some operation. As it was discussed above, when an agent tries to perform a sensitive operation, the call is intercepted by the current security manager, which assures that only the authorized agents complete the call

The agents can also request the `MobilityComponent` for an instance of a service that is currently running on the system (e.g. access to the persistent storage service). When such a call is made, the `MobilityComponent` checks with the global policy file to see if the agent has access to the desired service, and if so, passes the request to its associated component. The component then creates an object instance that implements the interface of the service, which will be used by the agent. Nevertheless, it is not this object that is given to the agent. The agent may only have access to certain methods and capacities of the service (e.g. a guest agent may ask the management service which are the agents running on the system, but only an administrator agent may kill other agents). These access rights are also specified in the policy file of the system. What the framework does is to create a dynamic proxy object that sits between the actual object instance implementation of the service and the agent. The agent only has access to the methods that the current policy allows him to. It should be noted that a finer-grain of security is even possible. When an agent calls a method on the system, the service method can also check which was the agent it, and decide how to respond in face of that information.

3.3.2. Confidentiality and Privacy

In our system, the agents cannot obtain any references to the objects of the system, or other agents. The only thing they can do is to make standard Java API calls, request access to existing services, and call a very well defined API for agent

[*] To be completely correct, what is stored in the local key store, and on the LDAP servers are the certificates of the users, not simply the public/private key pair.

migration and basic management. All these calls are under strict access control. Thus, each agent is an island when it concerns other agents. There is no way for an agent (at least an ordinary agent, without permissions) to obtain a reference to another agent. The data of each agent is private, as well as its execution state.

The M&M framework also has several inter-agent communication services. Even in these services, the agents only exchange messages and objects, and each agent may stop its instance of the service at any moment. The only way for an agent to access directly another agent is by having it sending a message with a direct reference to him.

For protecting the integrity and confidentially of the agent during migration, the M&M framework uses SSL sockets[16].

3.3.3. Accountability

One key issue on a secure system is that its users must be held accountable for their actions. The M&M framework has extensive logging capabilities. The logging levels are specified in a file which is read when the system starts, and can be refreshed. Each log entry has a severity level, an origin, a timestamp and a message. There is two types of log entries: entries originated by the running system (the components of the framework); and entries originated because of actions of the agents. In the case of log entries concerning agents, the entry also maintains the identity of the agent.

One important point is that logging is quite a heavy operation since it implies writing to disk, which can happen frequently, depending on the level of activity in the system. Because of this, the exact level of required logging must be carefully configured, including which services must or must not perform logging, and at what level.

3.3.4. Installable Services and the Remote Management Interface

The framework also supports two important concepts: Remote Management and Installable Services. M&M provides a component that allows the existing components in the system to be configured and managed, as well as the agents running on the system. In order to access this interface, the user must also pass through a process of authentication and authorization where its credentials are checked against the policy file of the system. Only after that, the user can access the remote management object of the system.

Finally, fitting with the global component framework, M&M also supports the concept of installable services. It is possible for a user to install and configure a new service through the remote management interface. It is also possible for agents to transport services and request the dynamic installation of a service. Again, these operations are only accessible to principals with the proper credentials. One important point is that these services will be running under the security permissions of an arbitrary principal indicated at the time of the installation, and for which a correct password was supplied. This is so to avoid attacks in which an agent could install a service, which would run with the full permissions on the system, and then require a service instance that would allow it to control the whole framework.

3.3.5. Cryptographic primitives

One final feature of the security architecture of the framework is the cryptographic primitives service for agents. Basically, there is a component in the framework that allows the agents (or higher-level services) to use cryptographic primitives for implementing secure information gathering protocols, as the ones proposed by Karjoth[12] and others. Although the agents cannot execute in safety on a completely open environment, with these primitives they have the tools needed for making private, confidential information gathering operations, on agent-accountable environments.

3.4. Limitations

The M&M framework was developed with the agent-accountable environment in mind. The main limitations found currently on the framework have to do with the limitations found in Java.

The biggest issue has to do with resource control. Presently there is no way on the framework to assess how many resources each agent is using, or a way to control that usage. This is because Java has no means to do this. We believe that in time Java will incorporate such mechanisms in its architecture. Although there are some projects that address the

problem of resource control, typically they require the usage o custom-made JVM[17], which is very restrictive. An interesting approach is taken in the SOMA platform[18], where the Java debugging and profiling interface is used.

One important issue in our framework is the question of integration with the existing applications. If an application already has a security manager in place, which does not conform to the Java 2 platform standard, it is not possible to use our security framework. Although this is a potential problem, we believe that will become less and less common, as people migrate to the new permission-based model of Java. Actually, we have currently experimented with the framework on several off-the-shelf web servers[19], which had different security managers instantiated, with no problems.

Another limitation found has to do with logging. Although we can perform logging for every component service developed by us, or even a third-party, it is currently not possible to log the security sensitive calls made by the agents to the standard Java API. Although this is technically possible, all it is necessary is to change the security manager so that the checks are intercepted and logged, we find this solution not to be an option. This would imply that the applications would have to use our security manager (i.e. a costume security manager), which is completely against the philosophy of the Java 2 security model, and actually from our approach. We believe that in the future Sun will address this common problem.

Finally, although it is not a limitation, we find the use of "virtual code signers" to be an ugly hack. Currently Sun has an early-access extension that will implement the notion of user in a very similar way to ours. The name of the package is Java Authentication and Authorization Service[20].

4. CONCLUSION

This paper has presented a two-fold discussion. First, it was discussed the issues and requirements of security on mobile agent systems. We have distinguished between deploying agents in open and closed environments, and then identified a model called agent-accountable environment. We believe that such an environment is adequate for many types of applications that can be made with mobile agents, and that its security requirements are obtainable with the current available technology.

The second part of this paper was devoted to the exploration of the security features and architecture of the M&M framework. While some platforms provide some security features[21], and others even provide complete security architectures[18], we have developed a very comprehensive security model for agent-accountable environments and have included it in our system. The security features of M&M, implemented on a component-based architecture include:

- Strong authentication and authorization mechanisms
- Key distribution based on LDAP
- Strong isolation between agents, and the agents from system resources
- Accountability of agent actions
- Dynamic deployment of services under correct security permissions
- Cryptographic primitives for agents

Although the architecture of M&M was thought to fit the agent-accountable environment scenario, we believe that if practical cryptographic solutions are found for making it safe for agents to roam completely free an open-environment, these can be easily integrated into the modular component architecture of M&M.

Finally, throughout the paper we have discussed the implications of using Java for developing mobile agent systems, especially in terms of security.

ACKNOWLEDGMENTS

This investigation was partially supported by the Portuguese Research Agency – FCT, through the program PRAXIS XXI (scholarship number DB/18353/98), the M&M project (project reference POSI/33596/CHS1999), and by CISUC (R&D Unit 326/97).

REFERENCES

1. D. Chess, C. Harrison and A. Kershenbaum, "Mobile Agents: Are They a Good Idea?", *IBM Research Report RC 19887*, IBM Research Division, New York, USA, 1995
2. M. Greenberg, J. Byington and D. Harper, "Mobile Agents and Security," in *IEEE Communications Magazine*, IEEE Computer Society Press, vol. 36, no. 7, pp. 76-85, July 1998
3. W. Farmer, J. Guttman, and V. Swarup, "Security for Mobile Agents: Issues and Requirements," in *Proceedings of the 19th National Information Systems Security Conference*, pp. 591-597, Baltimore, USA, October 1996
4. P. Marques, P. Simões, L. Silva, F. Boavida and J. Silva, "Providing Applications with Mobile Agent Technology", in *Proceedings of the 2001 IEEE Open Architectures and Network Programming Conference (OpenArch'2001)*, IEEE Computer Society Press, pp. 129-136, Anchorage, Alaska, USA, April 2001
5. G. Hamilton, "JavaBeans Specification 1.01", Sun Microsystems, Mountain View, California, USA, July 1997, available at http://java.sun.com/products/javabeans/docs/beans.101.pdf
6. D. Coward, "Java Servlet Specification 2.3 (Proposed Final Draft 2)", Sun Microsystems, Palo Alto, California, USA, April 2001, available at http://java.sun.com/products/servlet/index.html
7. F. Hohl, "A Model of Attacks of Malicious Hosts Against Mobile Agents," in *Proceedings of the ECOOP Workshop on Distributed Object Security and 4th Workshop on Mobile Object Systems: Secure Internet Mobile Computations*, pp. 105-120, INRIA, France, 1998
8. T. Sander and C. Tschudin, "Towards Mobile Cryptography", in *Proceedings of the 1998 IEEE Symposium on Security and Privacy*", IEEE Computer Society Press, pp. 215-224, Oakland, California, May 1998
9. U. Wilhelm, S. Staamann and L. Buttyan, "Protecting the Itinerary of Mobile Agents", in *Proceedings of the ECOOP Workshop on Distributed Object Security and 4th Workshop on Mobile Object Systems: Secure Internet Mobile Computations*, pp. 135-145, INRIA, France, 1998
10. B. Yee, "A Sanctuary for Mobile Agents", in *Secure Internet Programming*, Jan Vitek and Christian Jensen (Eds.), LNCS 1603, Springer-Verlag, pp. 261-274, 1999
11. W. Farmer, J. Guttman and V. Swarup, "Security for Mobile Agents: Authentication and State Appraisal", *in Proceedings of the Fourth European Symposium on Research in Computer Security (ESORICS'96)*, LNCS 1146, Springer-Verlag, pp. 118-130, Rome, Italy, September 1996.
12. G. Karjoth, N. Asokan and C. Gülcü, "Protecting the Computation Results of Free-roaming Agents" in *Proceedings of the Second International Workshop on Mobile Agents (MA'98)*, LNCS 1477, Springer-Verlag, pp. 195-207, Germany, 1998.
13. P. Marques, L. Silva, J. Silva, "Going Beyond Mobile Agent Platforms: Component-Based Development of Mobile Agent Systems", in *Proceedings of the 4th International Conference on Software Engineering and Applications (SEA'2000)*, IASTED/ACTA Press, pg. 119-125, Las Vegas, USA, November 2000
14. "The Mobile Agent List," available at http://www.informatik.uni-stuttgart.de/ipvr/vs/projekte/mole/mal/mal.html
15. Sun Microsystems, "The Java 2 Platform", available at http://java.sun.com/j2se
16. Sun Microsystems, "Jave Secure Socket Extension (JSSE)", available at http://java.sun.com/products/jsse
17. N. Suri, J. Bradshaw, M. Breedy, P. Groth, G. Hill, R. Jeffers, "Strong Mobility and Fine-Grained Resource Control in NOMADS", in *Proceedings of the Second International Symposium on Agent Systems and Applications and Fourth International Symposium on Mobile Agents, ASA/MA 2000*, LNCS 1882, Springer-Verlag, pp. 2-15, Zurich, Switzerland, September 2000
18. P. Bellavista, A. Corradi and C. Stefanelli, "A Secure and Open Mobile Agent Programming Environment", in *Proceedings of the Fourth International Symposium on Autonomous Decentralized Systems (ISADS '99)*, IEEE Computer Society Press, pp. 238-245, Tokyo, Japan, March 21-23, 1999
19. P. Marques, R. Fonseca, P. Simões, L. Silva, J. Silva, "Integrating Mobile Agents into Off-the-Shelf Web Servers: The M&M Approach", to appear in the *Proceedings of the International Workshop on Internet Bots: Systems and Applications (INBOSA'2001)*, IEEE Computer Press, Munich, Germany, September 2001
20. Sun Microsystems, "Java Authentication and Authorization Service (JAAS)", available at http://developer.java.sun.com/developer/earlyAccess/jaas/index.html
21. T. Walsh, N. Paciorek and D. Wong, "Security and Reliability in Concordia," in *Proceedings of the 31st Annual Hawaii International Conference on System Sciences (HICSS31)*, IEEE Computer Society Press, pp. 44-53, Kona, Hawaii, January 1998