A Flexible Policy Architecture for Mobile Agents

Suat Ugurlu and Nadia Erdogan

Istanbul Technical University, Computer Engineering Department,
Ayazaga, 34390 Istanbul, Turkey
suat@suatugurlu.com, erdogan@cs.itu.edu.tr

Abstract. Recent advances in distributed computing has lead software agents to be mobile and/or composed of distributed resources. In order to perform certain tasks, mobile agents may require access to resources available on remote systems. Although appealing in terms of system design and extensibility, mobile agents are a security risk and require strong access control. Further, the mobile code environment is fluid where resources located on a host may change rapidly, necessitating an extensible security model. This makes difficult to dynamically change agent ability and host security strategies in order to adapt to evolving conditions of the execution environment. In this paper, we present the design and implementation of a policy-based secure mobile agent platform (SECMAP). The platform makes use of agent and host policies for security and flexibility concerns. Its main strength is that it allows security policies to be specified or modified dynamically at runtime, resulting in high adaptability of agents and hosts to varying system state and requirements.

1 Introduction

Intelligent agents and multi-agent systems bring in a new approach to the design and implementation of complex distributed systems. Several multi-agent systems have been implemented either as commercial products or in various research projects, with varying success [1]-[7]. Reasons for the growing recognition of agent technology are the innovative solutions it provides to problems of more traditionally designed distributed systems through mobility of code, machine based intelligence, and improved network and data-management possibilities. Using mobile agent technologies provides potential benefits to applications, however, an agent's ability to move introduces significant security risks. Both mobile agents during their life times and hosts executing mobile agents are under security threats [8], [9].

Mobile code environments, however, have two important characteristics. They are dynamic - mobile programs come and go rapidly, and the resources present on a host may change. They are also unpredictable - administrators might not know ahead of time the source, behavior, or requirements of the programs that migrate to their host. There is no fixed set of resources that a host administers. Further, because the different components of resources and mobile programs may

J. Wiedermann et al. (Eds.): SOFSEM 2006, LNCS 3831, pp. 538–547, 2006.

require different levels of protection, security models must support fine-grained access control. This paper describes a new mobile agent platform, Secure Mobile Agent Platform (SECMAP) and its policy architecture. Unlike other agent systems, SECMAP proposes a new agent model, the shielded agent model, for security purposes. A shielded agent is a highly encapsulated software component that ensures complete isolation against unauthorized access of any type. SECMAP presents a policy-driven framework to support adaptive and dynamic behavior of agents, providing a secure environment through host and agent policies. SECMAP allows dynamic manipulation of policy content, which results in an adaptive and flexible framework that eliminates the reprogramming of the agents on changing conditions.

2 SECMAP Architecture

A brief overview of SECMAP architecture[8] is necessary before the description of the policy architecture. We have used Java for the implementation of the execution environment because it offers several features that ease the development process. The main component of the architecture is a Secure Mobile Agent Server (SMAS) that is responsible of all agent related tasks such as creation, activation, communication, migration and execution of policies. The system comprises of several SMAS executing on each node which acts as a host for agents. A SMAS may operate in three modes according to the functionality it exhibits. It can be configured to execute in any of the three modes on a host through a user interface. A SMAS on a node can also operate in all three modes at the same time.

Standard Mode(S-SMAS): S-SMAS provides standard agent services such as agent creation, activation, inactivation, destruction, communication, and migration. It also includes a policy engine that checks agent activity and resource utilization according to the rules that are present in host and agent policy file. In addition, S-SMAS maintains a list of all active agents resident on the host and notifies the Master Browser SMAS anytime an agent changes state. Keeping logs of all agent activities is another important task S-SMAS carries out.

Master Browser Mode (MB-SMAS): When agents are mobile, location mappings change over time, therefore agent communication first requires a reference to the recipient agent to be obtained. In addition to supporting all functionalities of S-SMAS, MB-SMAS also maintains a name-location directory of all currently active agents in the system. This list consists of information that identifies the host where an agent runs and is kept up to date as information on the identities and status (active/inactive) of agents from other SMAS is received.

Security Manager Mode (SM-SMAS): In addition to supporting all functionalities of S-SMAS, SM-SMAS performs authentication of all SMAS engines and maintains security information such as DES keys and certificates. Any SMAS engine in the system has to be authenticated before it can start up as a trusted server. SM-SMAS holds an IP address and key pair for each

of SMAS engine that wants to be authenticated. If the supplied key and the IP address of the requesting SMAS engine is correct then it is authenticated. The authenticated SMAS engine gets a ticket from the SM-SMAS and uses this ticket when communicating with other SMAS engines. A SMAS that receives a request from another SMAS refers to SM-SMAS to verify the validity of its ticket before proceeding with the necessary actions to fulfill the request.

SECMAP provides a secure communication framework for mobile agents [9]. Agent communication is secured by transferring encrypted message content by SSL protocol and is managed in a location transparent way. SECMAP also supports weak migration of agents between remote hosts.

2.1 SECMAP Agents

SECMAP requires agents to conform to a software architectural style, which is identified by a basic agent template given below. The agent programmer is provided a flexible development environment with an interface for writing mobile agent applications. He determines agent behavior according to the agent template given and is expected to write code that reflects the agent's behavior for each of the public methods. For example, code for the *OnCreate()* method should specify initial actions to be carried out while the agent is being created, or code for the *OnMessageArrive()* method should define agent reaction to message arrival.

```
public class Main extends Agent{
public void OnMessageArrive(){...}
public void OnCreate(){ ... }
public void OnActivate(){...}
public void OnInactivate(){... }
public void OnTransfer(){... }
public void OnEnd(){... }}
```

An instance of class *AgentIdentity* is defined for the agent on an initial creation. All agents in the system are referenced through their unique identities, which consist of three parts. The first part, a random string of 128 bytes length, is the unique identification number and, once assigned, never changes throughout the life time of the agent. The second part is the name which the agent has announced for itself and wishes to be recognized with. While the first two parts are static, the third part of the identity has a dynamic nature: it carries location information, that is, the address of the SMAS on which the agent is currently resident, and varies as the agent moves among different nodes. This dynamic approach to agent identity facilitates efficient message passing.

3 Security Policies

SECMAP provides a highly configurable security environment by supporting policy-driven integration of mobile agent activity with system level services and

resources. Policies define how allocation of resources is to be carried out and how security should be ensured. A policy is represented by a number of rules, where each rule is triggered by an event and consists of an action if a condition is evaluated to true. Thus, a security policy specifies the conditions under which a request is to be granted. If a request does not violate a policy rule, it is allowed to proceed; if it does violate a policy rule, it is blocked. The system-wide security policies are defined by agent developers as well as by system administrators.

A mobile agent is expected to adapt itself to environmental changes immediately. Such dynamic behavior in mobile agent systems requires mechanism where agent reprogramming is not needed. Hosts also need to easily reconfigure their resources in order to provide more flexible environments for the mobile agents. SECMAP's approach to achieve such flexibility is by means of dynamic policies that allow the agent programmer to change his agent's abilities without reprogramming the agent and the system administrator to reconfigure the execution environment on changing conditions. Thus, SECMAP employs policies mainly for two reasons: security and dynamism. The platform supports the specification of two kinds of security policies.

Host Policy: Host policies are concerned with the security of the host and its execution environment. They ensure that the local resources of the host are protected from unauthorized actions by the agent, by either granting or denying agent requests according to local policies.

Agent Policy: Agent policies are specified by the creator of the agent and define the capabilities of the agent to carry out requests on remote hosts. Those access privileges may be dynamically updated on changes in policy content. Agent policies also serve to protect the agent against malicious hosts or other agents through restrictions on communication, migration, etc.

Java has a default *Security Manager* which is initially disabled for applications. The Security Manager is a single Java object that performs runtime checks on potentially unsafe method calls. Code in the Java library consults the Security Manager whenever an operation of this nature is attempted. Decisions made by the Security Manager take into account the origin of the requesting class. The Security Manager makes the final decision as to whether a particular operation is to be permitted or rejected. In case of a reject decision, the operation is prevented to proceed by generating a *Security Exception*.

SECMAP agent servers utilize a strong custom policy engine that is derived from Java's default security manager. It replaces Java's default policy manager with an infrastructure that presents a flexible configuration interface for policies to be defined and assigned to agents and hosts. Opposite to Java's static policy definitions, the infrastructure allows policy rules to be inspected and manually modified at runtime so that policies can be dynamically adjusted to new, changing requirements and circumstances.

A SECMAP agent can issue two kinds of calls; SMAS calls or JAVA API calls, as shown in Figure 1. Through SMAS calls, the agent announces its requests to migrate, to communicate (send or receive messages), or to publish itself through the agent interface. Both kinds of calls are intercepted by the *Policy Engine* to

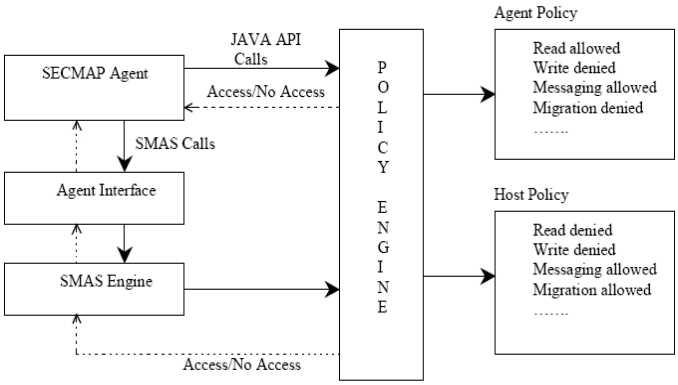

Fig. 1. The operation of the Policy Engine

check policy violations, and they are either allowed to proceed, or are blocked, according to agent and host policy definitions. Even though we are not able to catch every Java API call, we do intercept every call that Java's default security manager supports. These are;

- File system functions (read, write, delete)
- Networking functions (socket creation for listening and opening connection)
- Class loader functions
- Functions accessing system resources (print queues, clipboards, event queues, system properties, and windows)

The operation of the Policy Engine is as follows:

- An agent or the SECMAP platform itself makes a call to a potentially unsafe operation in the Java API, or an agent makes a communication or migration call.
- The call is intercepted by the Policy Engine to be checked for any policy violations.
- The Policy Engine determines the source of the call, if the call is issued by SECMAP, the operation is permitted to proceed, if the call is issued by an agent, the Policy Engine finds out its agent identity in order to refer to its specific policy definitions. In case the operation is permitted by the agent policy, the Policy Engine next refers to the host policy. If host policy permits the operation as well, then the Policy Engine allows the call to continue with a no-error return. On the other hand, if either agent policy or host policy, or both, do not permit the operation, Policy Engine returns a SecurityException, thus blocking the call.

3.1 Agent Policies

Each SECMAP agent is assigned an agent policy which includes "creator granted capabilities", when the agent is first deployed into the system by the agent

programmer. The agent policy simply defines the types of actions that the agent can perform in its execution environment (Migration, messaging, writing to disk, reading from disk, etc.) Agent policies are maintained as encrypted XML files and carried with the agent itself as it moves between nodes. SECMAP keeps all of agent class files in a single zipped encrypted file and stores it together with the agent policy and data file in a secure place in the host disk. When an agent is to be activated, SMAS first loads the agent's classes and its final state information from its code and state files. Next, it creates an "Agent Policy" object for the agent. The *AgentShield* object that isolates the agent from its environment associates this policy object with the agent and updates the policy values from the agent's policy file before activating the agent. If agent policy is modified at runtime by the agent programmer, SMAS updates the agent policy file on host disk as well.

The platform presents a flexible graphical interface window for the agent owner to monitor his agents on the network and to manually change their policies if necessary to adapt to changing conditions at any time during execution. An agent policy can only be changed by the agent owner.

3.2 Host Policies

Host policies mainly serve security reasons by denying unauthorized agent access to host resources. Security and flexibility generally are not tolerant to each other; however, SECMAP policy architecture presents a flexible environment for mobile agents while it can still protect the host from intrusted agents that come from unknown sources. Host policy rules are defined considering two criteria: the agent owner and the agent source.

Agent Owner: Agent owner is the location where the agent is created and deployed into the system. It simply consists of the IP address of the host where the agent was created. The agent owner information is carried with the agent and does not change throughout its life time.

Agent Source: Agent source is the location from where the agent has migrated. It consists of the IP address of the host where the agent was running previously. This information is not carried with the agent since, on a migration request; the target agent server is provided with the information where agent is coming from and where it wants to move. The host administrator can define different host policy rules for different agent owners and agent sources.

As can be seen in the Figure 2, the administrator can assign different policies for different agent owners so that some agents whose owners are trusted will possess more rights than others whose owners are not trusted as much. It is possible to restrict all or particular actions of agents whose owners are not trusted. It is also possible to define rules that will enable the host to reject migration requests of agents from a specific agent owner or source. Sending to and receiving messages from intrusted agent sources can also be similarly restricted.

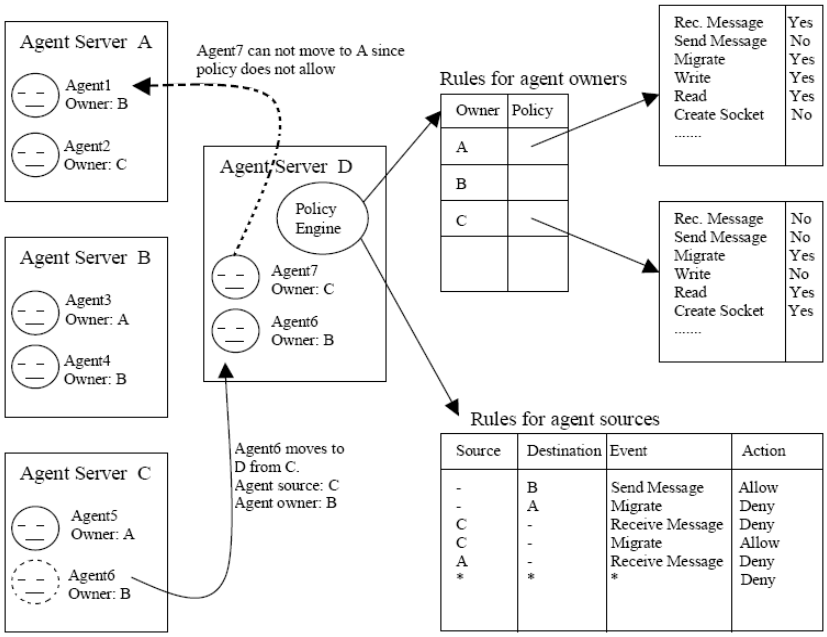

Fig. 2. A snap shot of agents and host policy rules at runtime

The platform provides the host administrator a flexible graphical window interface to specify host policies. Rules for both agent owners and for agent sources are checked for inconsistencies. For example if there is a rule stating that agents can not send messages to Agent Server B, then a new rule stating agents can send messages to Agent Server B will not be allowed to be defined.

Since there may be a large number of agent owners and agent sources in a network, it is not convenient for the administrator or agent programmer to define policy rules for each one of them separately. For example, the administrator may wish to grant certain rights to particular agent owners or sources while denying the rest, or he may wish to deny particular agent owners or sources while permitting the others. To ease rule specification in such conditions, there is a default policy rule at the end of the host policy that determines the course of action in case there is not a matching rule for a specific owner or source. This default rule is represented by the symbol "*" in Figure 2.

3.3 Performance Evaluation

While policies provide both security and execution time flexibility, one drawback may be a performance overhead being introduced. We have carried out tests in order to analyze the performance affect of policy usage in SECMAP. We've chosen "disk write" action of an agent as a sample operation. In terms of policy execution, the type of operation has no influence on the results obtained as the

agent server's policy engine proceeds in exactly the same way for each type of operation. The tests are carried out on a PC with celeron 2.4 GHz CPU, 512 MB RAM.

We first carried out the test with all policy features disabled and computed the time required when access requests are not checked against policy rules. Next, we enabled the policy features and repeated the test with different numbers of policy rules, to see the influence of varying numbers of policy rules in the host policy on performance. Table 1 shows the result, the elapsed time measured for each test. The agent code executed for the tests is also given below.

```
1-  byte[] text = "Hello I am the helloworld agent".getBytes();
2-  FileOutputStream fo = null;
3-      for (int j = 1; j <= 10; j++) {
4-          long startTime = System.currentTimeMillis();
5-          for (int i = 1; i <= 50000; i++) {
6-              File a = new File("agentdata.txt");
7-              a.createNewFile(); //Policy engine interruption
8-              fo = new FileOutputStream(a);
9-              fo.write(text);
10-             fo.close(); }
11-         long endTime = System.currentTimeMillis();
12-         System.out.println("Elapsed Time = ");
13-         System.out.println(endTime - startTime);}
14-     }
```

When we analyze the results, we see that the use of policies brings a performance overhead of about %13 ((35823-31691)/31691) for operations which require a security check. This decrease in performance may be overlooked when the benefits of policy usage is considered. We also see that the number of rules in host policy has no effect on the performance attained. This is because policy rules are kept in a hash table in the memory and are searched for in a very time-effective manner.

Table 1. The elapsed time measured for each test

Test	Policy Disabled	Policy Enabled Number of Rules:1	Policy Enabled #Rules:10	Policy Enabled #Rules:30
1	31875ms	35562ms	35828ms	35032ms
2	30735ms	35735ms	33906ms	35500ms
3	29891ms	35781ms	36219ms	37171ms
4	34579ms	35469ms	37640ms	35110ms
5	29985ms	35813ms	34485ms	35937ms
6	28750ms	36531ms	36578ms	36500ms
7	36421ms	35281ms	35203ms	33625ms
8	32829ms	35250ms	37500ms	37906ms
9	31047ms	36672ms	35687ms	35016ms
10	30797ms	36140ms	35969ms	36781ms
Avg	31691ms	35823ms	35902ms	35858ms

4 Related Work

Several mobile agent systems have been proposed and developed up to now. They all have their software agent specific features. Although most of them have enough features for mobile agents to communicate with each other and migrate to remote hosts, a flexible policy-based management is not available in any. Because mobile agents require a dynamic environment where conditions and requirements may change rapidly, necessary changes should be done without reprogramming agents. SECMAP's policy architecture gives this opportunity to agent programmers and administrators. Gallery [10] introduces a framework to authorize mobile agents and determines whether or not a mobile agent should be executed on a particular platform. SECMAP includes both authentication and authorization mechanism while it is possible to give an agent detailed access rights. The work in[11] implements a policy-based solution to control only mobile agent mobility. In [12], the researchers have developed an authorization platform that supports definition and enforcement of history-based security policies, allowing hosts to decide on the authorization of an agent's action upon its past behaviour. As a whole, we see that work has focused on policy-based solutions for the problems in mobile environments with different approaches. A contribution of SECMAP is that, it not only supports mobile agent services in a secure way, but it also presents a policy-based management.

5 Conclusions and Future Work

This paper describes the policy architecure of a secure mobile agent platform. SECMAP provides an isolated, secure execution environment for mobile agents. It also presents a policy based management framework to protect system-level resources and agents against unauthorized access, as well. The policy architecture allows for dynamic manipulation of policy content, which results in an adaptive and flexible framework that eliminates the reprogramming of the agents on changing conditions.

Future work will concentrate on definition of policy rules that specify further details on rights granted, possibly narrowing the scope of rules. For example, it will be possible to restrict disk access rights to specific files. The only unsafe action for the host that an agent may perform and which we do not control with security policies is memory and CPU utilization functions. An agent may consume host memory and CPU time, running in an endless loop, and currently SECMAP doesn't have the capability to realize and stop this kind of action. We have already worked on some methods to measure the amount of memory that an agent is using and have obtained some good results. However, this kind of low level checks ended with different results in different JVM. After completing our work successfully, we plan to restrict the memory usage of agents using policies.

References

1. Voyager, `http://www.recursionsw.com/products/voyager/voyager.asp`
2. Aglets, `http://www.trl.ibm.com/aglets/`

3. http://www.genmagic.com/technology/odyssey.html
4. Stanford Univ., JATLite, http://www-cdr.stanford.edu /ProcessLink/ papers/ jat
5. http://www.cs.dartmouth.edu/ dfk/papers/gray:security-book.ps.gz
6. Bryce, C., and Vitek, J.: The JavaSeal Mobile Agent Kernel. Autonomous Agents and Multi-Agent Systems **4** (2001) 359–384
7. Concordia, http://www.merl.com/projects/concordia
8. Ugurlu, S., and Erdogan N.: An Overview of the SECMAP Secure Mobile Agent Platform. AAMAS05 – 2nd International Workshop on Safety and Security in Multiagent Systems (SASEMAS '05), Utrecht, The Netherlands (July 2005)
9. Ugurlu, S., and Erdogan, N.: A Secure Messaging Architecture for Mobile Agents. To appear in the LNCS proceedings of The 20th International Symposium on Computer and Information Sciences, Istanbul, Turkey (October 2005)
10. Gallery, E.: Towards a Policy Based Framework for Mobile Agent Authorisation in Mobile Systems. Mobile VCE Research Group, Royal Holloway, University of London (2003)
11. Montanari, R., Lupu, E., Stefanelli, C.: Policy-Based Dynamic Reconfiguration of Mobile-Code Applications. ISSN: 0018-9162, Computer **37** 7 (July 2004) 73
12. Dias, P., Riberio, C., Ferreira, P.: Enforcing History-Based Security Policies in Mobile Agent Systems. Fourth IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY'03) (2003) 231