Security Issues in Mobile Agents

- Kedar Mohare
Department Of Computer Science And Engineering
University Of Texas At Arlington
kbm6471@omega.uta.edu

Abstract

Mobile agents offer a new paradigm for distributed computation, but their potential benefits must be weighted against the very security threats they pose. These threats originate not just in malicious agents but malicious hosts as well. Thus security is a very crucial issue in dealing with mobile agent systems without which their implementation in real life applications will be rendered useless. This paper highlights the problems encountered in a mobile agent system from a host's point of view as well as from a mobile agents point of view, as well as the various approaches to dealing with those problems. The paper concludes with the implementation of two mobile agent systems: Aglets & D'Agents and describes their approaches to integrate security into the system.

1. Introduction

A mobile agent is a program that moves from machine to machine and executes on each. Neither the agent nor the machines are necessarily trustworthy. The agent might try to access or destroy privileged information or consume more that its share of some resources. The machines might try to pull out sensitive information out of the agent or change the behavior of the agent by removing, modifying or adding its data code. A mobile agent system that does not detect and prevent such malicious actions can never be used in real applications. **Security is the most critical issue in a mobile agent system [3].**

The second section deals with the security issues involving a mobile agent system. The third and the fourth section describe the various general approaches to protecting the hosts from the mobile agents and the mobile agents from the malicious hosts respectively. The next two sections provide two practical implementations of the mobile agent systems namely, AGLETS & D'AGENTS. We conclude the paper with a discussion and summary of the various topics discussed in the paper and practical feasibility of these schemes.

2. Security issues in mobile agent systems:

A mobile agent system has the following four interrelated problems: **protect the machine from visiting agents, protect other agents from the visiting agents, protect the agent from the host machine and finally protect a group of machines from the agents.** The following discussion will demonstrate the threats encountered in mobile agent system with the aid of two examples in which mobile agents are used to perform some useful work.

2.1 Competing airline carriers [3]

Consider a mobile agent that visits the websites of several airlines looking for a flight plan that meets customer's requirements. We focus on four hosts: a customer host, a travel agency host and two server host owned by competing airlines viz. America Airlines and United Airlines. A travel agent programs the mobile agent. A customer dispatches the agent to the United Airlines server where the agent queries the flight database. With the results stored in this environment, the agent then migrates to the American airlines server where it again queries the flight database. The agent has flight and fare information and decides on a flight plan, migrates to the appropriate airline host and reserves the flight. Finally the agent returns to the customer with the results.

There are a number of attacks that they may attempt. For instance, the second airline server may be able to corrupt the flight information of the first airline as stored in the in the environment of the agent. It could surreptitiously raise its competitor's fares or it could advance the program counter into its preferred branch of conditional code. Thus the agent cannot decide its flight plan on an airline host since

the host had the ability to manipulate the decision. Instead the agent will have to migrate to a neutral host like the customer host or the travel agent host and then migrate to the selected airline to complete the transaction. A second kind of attack is also possible. The first airline may cheat the second airline for instance, when the second airline has a cheaper fare available. The first airline surreptitiously increases the number of reservations from 2 to 100. The agent will then proceed to reserve 100 seats of the second airlines cheaper rate. Later legitimate customers will have to book their flights on the first airline, as the second believes that its flight is full.

2.2 Distributed Intrusion Detection [3]

Suppose we have a mobile agent system architecture where the network is partitioned into one or more network domains. Each domain has a trusted computer running an interpreter that is trusted by all agents in that domain. All other interpreters run on untrusted computers that the intrusion detection system is trying to protect, hence these interpreters cannot be trusted. Consider a data collection agent that is dispatched by trusted interpreters and migrates to an untrusted host, collects process information from that host and migrates back to original interpreter to deposit the collected information. If the network addresses of the two interpreters are stored as state variables of the agent, the second interpreter can switch the two addresses, reset the program counter and return the agent to the first interpreter. The agent will now collect information from the first interpreter and return the information to the second interpreter thus providing valuable information to an attacker. This illustrates that an agent can become malicious by virtue of its state becoming corrupted.

2.3 Types of Attacks In a Mobile Agent System:

There are two categories involving misuse of mobile agents:
(1) Misuse of hosts by mobile agents and (2) misuse of mobile agents by hosts and other mobile agents.
These types of attacks are often categorized as follows: **damage, denial of service, breach of privacy, harassment, social engineering, event triggered attacks and compound attacks.**

The table below summarizes the various threats /attacks involved in a mobile agent system:

Types of Attacks	Threats to host	Threats to mobile agents
Damage	Destroying/changing services by erasing disk	Erasing its state leaving it in an unstable state
Denial of Service	Overload service by consuming network connection, overflowing buffers	Failure to give net access, leaving the mobile agents stranded
Breach Of Privacy or Theft	Access and steal private information on host	Copying of binary image of the mobile agent
Harassment	Repeated display of unwanted information	Tracking mobile agent to discover information.
Social Engineering	Stealing user passwords posing as system administrator.	Misdirecting hosts to unwanted destinations

Table . 1 Threats in a mobile agent system

Damage:

Destruction or subversion of a hosts files, configuration or hardware, or of a mobile agent or its mission [1].

To hosts: A mobile agent can destroy or change resources or services by reconfiguring, modifying, or erasing them from the disk. When a mobile agent deliberately damages a mobile agent system, it inadvertently destroys all the other mobile agents executing there at that time. Examples of destruction include deleting randomly or writing into files or ordering an unscheduled hardware upgrade to a host. An example of subversion is modification of the system configuration to change the security policy.

To mobile agents: A host can destroy a mobile agent by erasing it and in the process losing anything it has gathered and possibly leaving it in an unstable state. A mobile agent can be subverted and caused to radically change its function [6].

Denial of Service:

Partially or completely impeding one or more computer services, or a mobile agents access to some resources or services [1].

To hosts: An executing mobile agent overloads a resource or service such as by constantly consuming a network connection, or a mobile agent blocks another process by overloading its buffers to create deadlocks.

To mobile agents: An execution layer may fail to provide access to system resources. For example if a host fails to give a mobile agent system network access, present mobile agents will be stranded.

Breach Of Privacy or Theft:

Illegal or undesirable access or removal of data from a host or a mobile agent [1].

To hosts: A mobile agent accesses and steals private information, for example secretly recording the input of the computers microphone to an unauthorized site.

To mobile agents: At any point when visiting host, a mobile agents is at the risk of having its binary image copied unless it is encrypted. Since it has to be decrypted to execute, execution provides a potential point of failure.

Harassment:

Annoying people with repeated attacks [1].

To hosts: A mobile agent programmed to attempt to offend people can display unwanted pictures, or write to the screen at intervals that cause the display to flicker.

To mobile agents: A mobile agent can be attacked in ways that will bother its sender. Examples include delaying the mobile agent, or tracking mobile agent to discover information about it or its sender.

Social Engineering:

Manipulation of people, hosts, or mobile agents by using misinformation coercion and other means [1].

To hosts: A mobile agent can play a role in social engineering. For instance it could request users passwords under false authority of the systems administrator. Next the mobile agent could go to 100 employees computer with the appearance of stemming from the system administrator. Each of them would then concurrently ask the employees for their passwords, and then disappear with revealed passwords before a word could be passed that a fraudulent mobile agents has been detected [6].

To mobile agents: Mobile agents can be given misinformation with the objective of manipulating them to their senders. Hosts can misdirect mobile agents to unwanted destinations.

Event triggered attack:

Initiation of any of the previously described attacks based on an external event [1].

To hosts: An event-triggered attack "goes off" when code, concealed within an apparently benign mobile agent is triggered by special event, such as time location or on arrival of a specific person.

To mobile agents: Mobile agent can be attacked for something they are carrying or because they are from a specific sender.

Compound attacks:

A compound attack is composed of multiple attacks than can achieve more than single attack [1].

To hosts: Using cooperation techniques from agent research, mobile agents can collaborate with each other in order to commit a series of attacks.

To mobile agents: Hosts or mobile agents can track or stalk a mobile agent with the intent of stealing from it, discovering its source or destination, and so on. Multiple collaborating hosts can track the mobile agent until it has something of interest and then attack it.

The biggest threat is a compound attack strategically aimed towards a goal. The scenario shown in the figure includes network hosts A, B, C, D, E with the two collaborating malicious mobile agents posing as network management agents. They are temporarily located on B & C and have been granted minimum access to network resources.

1. ‘A’ broadcasts an auto-configuration request. The recipients of this broadcast include the auto-configuration server on B and the two malicious mobile agents.
2. The malicious agent on B interferes with the auto-configuration server possibly by flooding the local network interface with traffic.
3. The malicious agent on C sends a normal auto-configuration reply to A, except that it contains the wrong server address, A accepts the malicious agents configuration reply.
4. After a short while the malicious agent on B stops interference, allowing the auto-configuration server to reply. The reply is ignored by A, which is no longer listening on that port.

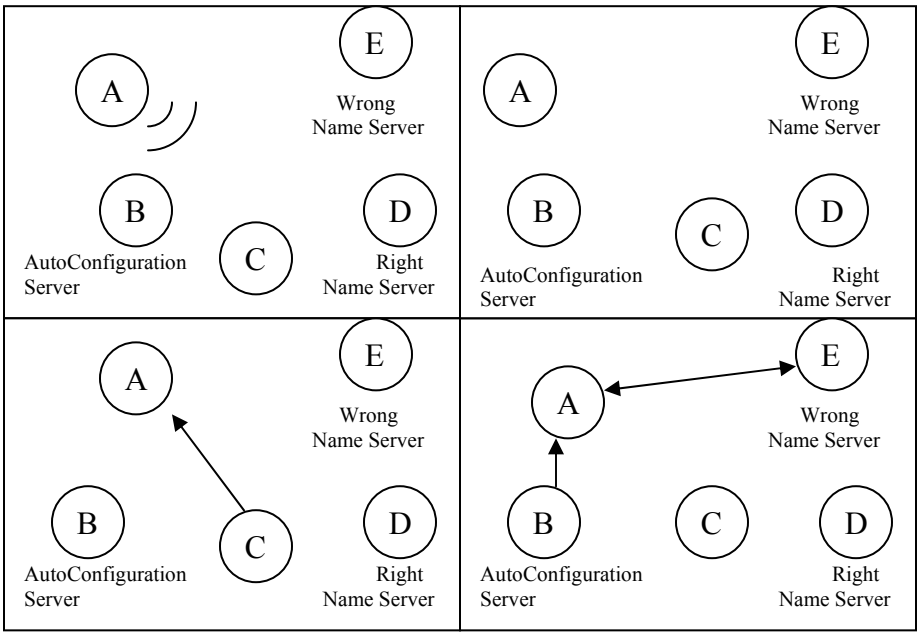


Fig. 1 [1] Compound attack Scenario

A is now receiving subtly subversive name service, which facilitates further security violations. The malicious mobile agents can depart to search for other vulnerable machines. They leave little or no trace. The malicious mobile agents used an event-triggered attack, coordinating activities based on an external event: **auto-configuration request.** This requires no overt communication and is hard to detect. Once access to local network resources is granted, there is little or no way of detecting their nefarious activities [1].

3. Security techniques protecting hosts.

The following technique are designed to protect hosts: **authenticating credentials, access level monitoring and control, code verification, time limits, range limits, duplication limits and audit logging [5]**.

Authenticating credentials: A mobile agent is digitally signed by one or more parties using a public key signature algorithm. Depending on the algorithm, this process either generates a certificate, which is then appended to the mobile agent’s binary image for use as a credential or encrypts the mobile agents binary image for privacy during transportation. In both cases digital signatures can be used to verify the identity of the mobile agent’s author and of its sender, where and when it was sent, and that it has not been tampered with in transit. Authenticating credentials do not guarantee that the mobile agent will be harmless or even useful. Forgery, theft of cryptographic keys or poor implementation can

compromise cryptographic techniques. The recipient is biasing his or her trust in mobile agents integrity based on another person's word, not on the knowledge of the mobile agent [3].

Access level monitoring and control: A reference monitor is used to restrict the information, system resources and service that mobile agents are allowed to access and use. The reference monitor grants permissions to mobile agents based on their level of authorization as shown by their authenticating credentials [3]. The reference monitor consults a security policy to determine if access is to be granted. Access level monitoring and control places restrictions directly on what a mobile agent can do.

Code Verification: The binary image of the mobile agent is scanned to determine if the mobile agent is a valid program. A code verification program, which finds illegal instructions in a mobile agents code will not invoke the execution layer. This technique is relevant for execution layers that are vulnerable to subversion and sabotage by mobile agents executing within them [4].

Limitation techniques:

The following three techniques are a natural extension of the need to control the persistent survivability of mobile agents.

Time Limits: The amount of time, elapsed or absolute, that a mobile agent system allows a mobile agent to run in an execution layer. A mobile agent that has been roaming the network past its relevant life span can be destroyed or returned to its origin because of an encryption protected timestamp embedded in its corpus.

Range Limits: The number of destinations or network hops that a mobile agent system will allow a mobile agent to be transmitted. Travel may be restricted to hosts listed on the mobile agents itinerary, such as only hosts within an organization.

Duplication Limits: The number of times that a mobile agent system will allow a mobile agent to be transmitted to transmit another mobile agent. For instance a mobile host that is duplicating recursively can easily swamp the hosts on a network.

The above three techniques are effective techniques to control mobile agents and prevent them from running wild because they directly control what a mobile agent does.

Audit Logging:

Recording all mobile agent activities. An audit trail is kept so that after abuse is detected the responsible party can be identified and called to account. Audit logging is done at each stage of the cycle,namely, arrival, execution and departure.

4. Security techniques protecting Mobile Agents:

There are two categories of mobile agent protection techniques: **fault tolerance** and **encryption [1]**. The techniques based on fault tolerance aim to make the mobile agent robust in an unpredictable environment, thus protecting them from malfunctioning hosts, intermittent network connectivity etc. The techniques based on encryption hide the mobile agent, code or sensitive data so that it cannot be recognized and will be thus less likely to be destroyed, stolen or otherwise misused.

4.1 Techniques based on fault tolerance:

Replication and voting: This technique ensures that mobile agent gets to its destination intact. A mobile agent traveling through a network replicates at each node with the nodes letting only intact agents to pass through. A shared secret carried by mobile agents is one way to determine if they are intact [1]. This technique can be used to ensure that a critical message is delivered through a network of potentially faulty or malicious hosts.

Persistence: This technique involves temporary storage of a running mobile agent and its execution state against host failure. When a mobile agent arrives on a host it uses a persistent storage facility in the mobile agent system (hard disk) to store its binary image for the duration of its visit. If the host crashes, destroying the executing mobile agent, the copy persists in the storage. When the host returns to service, it restarts the agents in the persistent storage.

Redirection: Mobile agents finding new paths around damaged hosts or network to complete a mission. When a mobile agent attempts to travel to a destination and fails to open the communication channel, it can be programmed to try other channels to attempt to reach its destination.

4.2 Techniques based on Encryption:

Sliding Encryption: A mobile agent uses this technique to encrypt the acquired data using the public key. Decryption can only be performed with the corresponding private key. The mobile agent uses sliding encryption to hide what it is carrying so that potentially malicious hosts that it visits cannot steal any data.

Trail Obscuring: Changing a mobile agents binary image to make it hard to identify by pattern matching. Mobile agent attempts to obscure its path through the network by constantly changing its binary image so that it cannot be identified as the same agent by different hosts, which are colluding in an attempt to track the mobile agents. It may also aid in surviving malicious hosts to stop specific behavior that can be identified by analyzing the mobile agents path [4].

Code Obfuscation: A method to obscure mobile agents code is to make it harder to reverse engineer. One promising method is called **black box security**. When an encrypted mobile agent is sent by a mobile agent system, a special execution layer is sent with it to run on each host the mobile agent visits and to execute the encrypted mobile agent while it is encrypted. The host never has direct access to the code or data of the mobile agent. This defers theft or subversion

Encrypted data manipulation: In this technique the binary data of a mobile agent is encrypted in such a way that it allows it to be manipulated while still encrypted. This allows the mobile agent to carry data that cannot be read by a host. The data is decrypted when the mobile agent returns to its sender. This technique defers theft or nondestructive tampering, and allows the mobile agent's code to be inspected.

5. Implementation of Mobile Agent Systems:

We will now discuss two mobile agent programming systems **IBM Aglets** and **D'Agents**.

5.1 IBM AGLETS:

The IBM Aglets Workbench lets users create **aglets**, mobile agents based on the Java programming language. The AWB consists of a development kit for aglets and a platform for their execution. It is based on the aglet object model, whose major elements are aglets, contexts, and messages. The Aglet Transfer Protocol (ATP) and the Aglet API (A-API) are AWB components that define how to transport aglets and how to interface to aglets and contexts. The Java Aglet API defines the methods necessary for aglet creation and manipulation. The key principals found in this model are aglet, context, proxy, message, itinerary, and identifier [14]. An **aglet** is a mobile Java object that visits aglet-enabled hosts in a computer network. It is autonomous, since it runs in its own thread of execution after arriving at a host, and reactive, because of its ability to respond to incoming messages. A **context** is an aglet's workplace. It is a stationary object that provides a means for maintaining and managing running aglets in a uniform execution environment where the host system is secured against malicious aglets. One node in a computer network may host multiple contexts. The context master is responsible for safety and security of the context and its underlying machine [14]. A master defines the security policy for the context under its control and is also responsible for guaranteeing that no aglet interferes with any other aglet active in the same context. A **proxy** is a representative of an aglet. It serves as a shield for the aglet, protecting it from direct access to its public methods. The proxy also provides location transparency for the aglet. That is, it can hide the real location of the aglet. A **message** is an object exchanged between aglets. It allows for synchronous as well as asynchronous message passing between aglets. Message passing can be used by aglets to collaborate and exchange information in a loosely coupled fashion. A message manager allows concurrency control of incoming messages. An **itinerary** is an aglet's travel plan. It provides a convenient abstraction for non-trivial travel patterns and routing. An **identifier** is bound to each aglet. This identifier is globally unique and immutable throughout the lifetime of the aglet. A **domain** is contexts organized in groups. A single context providing all services might be very expensive, so grouping contexts can be efficient and cost effective.

Table below points out the principals defined in the aglets security model.

Aglet	Instantiation of Aglet program Itself
Aglet Manufacturer	Author of an aglet program. (human, company)
Aglet Owner	Individual that launched the aglet
Context	Interpreter that executes Aglets
Context Manufacturer	Author of a context program
Context Master	Owner/administrator/operator of the context program
Domain	A group of contexts owned by the same authority
Domain Authority	Owner/administrator/operator of the domain

Table. 2 Principals in an Aglet Security Model

Security Policies:

All the principals stated above may define security policies. A secure aglet system should implement the overall effect of security policies involved. The hierarchy of the security policies defined by the different principals is:

Aglet Manufacturer<Aglet Owner<Context Master<Domain Authority [10]

This indicates that the domain authority sets the basic policies on the execution of aglets with in a given context.

Aglet Mobility:

Before the owner can launch an aglet, the context authenticates the owner as a registered user. Within the creation request, the aglet owner defines security preferences to be applied on the aglet. When the context instantiates the aglet from the corresponding Java class, it might include information about the manufacturer, owner, and the aglet's original context [14]. This information, together with the aglet code and the owner's security preferences, forms the static part of the aglet, and will be signed by the context. Any receiving context can verify the integrity of the static part of the aglet. Aglets move when they are either dispatched to or retracted from a remote location.

Access to local resources:

When an aglet enters a context, the context receives a reference to it, and the aglet resumes execution in the new context. An aglet uses the context to gain information about its new environment, in particular about other aglets currently active in the context in order to interact with some of them. Contexts have to protect themselves against aglets and aglets must be precluded from interfering with each other. The aglet context establishes a reference monitor, which gives an aglet access to a resource only if it complies with the access control policy instated by the context master. Thus a context establishes a domain of logically related services under a common security policy governing all aglets in that context. The master of the context configures authorization policies for incoming aglets [14].

Security Architecture:

The security architecture implements the security model by providing a set of components and their interfaces. The two components of the aglets security architecture: the policy database of the context master and the preferences of the aglet owner, are discussed later [14]. Because both context master and aglet owner have their own specific interests concerning what an aglet should be able to do, both may want to restrict its capabilities. Such restrictions might apply to either accessing the local resources of a context or offering services to other aglets. The policy database and security preferences therefore constitute powerful elements in introducing security into the Aglets Workbench. Any useful mobile agent system must implement general and flexible security policies. This model simplifies the administration of these policies by introducing the notion of roles, namely, the manufacturer, owner, master, and authority principals. In the following, is presented a language for defining policies using the concepts presented in our security model, and show how a context master and an aglet owner can use it to define their policies.

Authorization Language [10]

For illustration, we define the following principals

(1)Manufacturer: Canon, Nikon, (2)Owner: Yashica, Minolta ,(3)Master: Pentax, Vivitar, (4)Authority: Ricoh, (5)Context: Tamron, Fuji, (6)Aglet: Zenith, Kodak.

We use these names to simplify our discussion, but the real value of Fuji might be something like **atp://www.trl. ibm.co.jp** and its product name might be **ibm.aglets.tahiti**. The product name of aglet Kodak might actually **be ibm.aglets.samples.Writer [10]**.

Basic principals address single aglets or groups of aglets. **Composite principals** offer a convenient way to combine privileges that must be granted to multiple principals into a single access right. Such a grouping feature considerably simplifies the security administration. Membership in a group supports the combination of various principals that should have the same access rights. For example, the following specifications combine principals of the same type into named groups and use these group names later in rule definitions:

The following table[10] depicts some of the examples of basic principals and composite principals.

Basic Principals	Composite Principals
aglet=Kodak: denotes the aglet Kodak.	**GROUP AssociationOfManufacturers=Canon, Nikon [10].** This rule indicates that group "Association Of Manufacturers" consists of Canon and Nikon.
owner=Minolta: denotes all aglets launched by Minolta	**Kodak IS_MEMBER_OF Titans** This rule adds Kodak to group Titans.
context= Fuji: denotes all aglets arriving directly from Fuji	
manufacturer=Canon: denotes all aglets written by Canon.	

Table. 3 Basic and Composite Principals

Privileges

Privileges define the capabilities of executing code by setting access restrictions and limits on resource consumption. A privilege is a resource, such as a local file, together with appropriate permissions such as read, write, or execute in the case of the local file. The following table presents the resources and privileges.

Resources	Privileges
File: files in the local file system	**Context AGLET retract:** the aglet can retract any aglet in the context.
Net: network access	**Context AGLET owner=Leda retract:** the requester of method retractAglet can retract the specified aglet if the owner of the retracted aglet is Leda
QoP : quality of protection	**Context PROPERTY origin get:** the aglet can retrieve property origin of the context
System: any kind of system resources, such as memory and CPUs	**Context MESSAGE subscribe:** the aglet can subscribe to messages.

Table . 4 Resources And Privileges.

Resources are structured hierarchically. Thus, permissions can be given to a set of resources or even to a complete resource type, like universal file access. Net access is a more elaborate resource. The

authorization language lets distinguish among different protocols (for example, TCP and HTTP) and select ports or port ranges to build resources [10], viz., **Net**: any kind of networking, **Net TCP:** any kind of TCP connections, **Net TCP host:** TCP connections to host, **Net TCP host port:** TCP connections to host but only on port.

Each resource also has a corresponding set of permissions. The permissions for networks are send, receive, connect, and accept. The services provided by the aglet context are also subject to control. The context provides methods to create, retract, and activate aglets; to send or receive messages to/from other aglets; to obtain aglet proxies, the hosting URL, audio clips, and images; and to get or set the properties of the context.

Context Master Policy Database

The context master defines the security policy for aglet contexts under its control. This policy defines the actions an aglet can take. In the policy database, the context master combines principals that denote aglet groups and privileges into rules into rule. The syntactic form of a rule is

<label>:<principal> -> <privileges> [10]

Aglet Owner Preferences

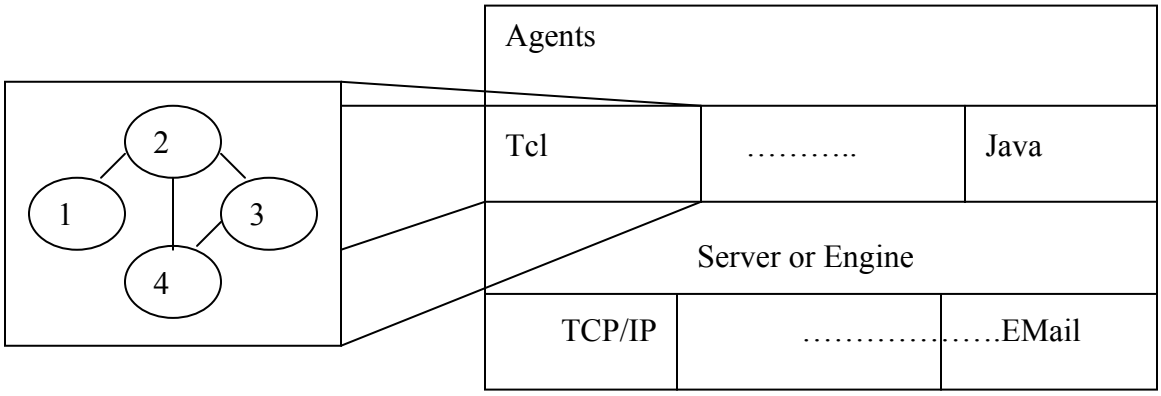
The aglet owner has the opportunity to establish a set of security preferences that will be honored by the contexts the aglet might visit. Preference combines context groups and privileges into rules. The syntactic form of a rule is:

<label>:<context_group_definition> -> <privileges> [10]

5.2 D'Agents:

D'Agents is a mobile agent system whose agents can be written in Tcl, Java and Scheme. Like all mobile agent systems the main component of the D'Agents, is a server that runs on each machine. When an agent wants to migrate to a new machine it calls a single function, which automatically captures the complete state of the agent and sends this state information to the server on the destination machine. The destination server starts up an appropriate execution environment, loads the state information into this execution environment, and restarts the agent from the exact point at which it left off.

The figure below shows the D'Agents architecture[2].



1 --- Security, 2 ---Interpreter, 3----State Capture, 4---- server API

Fig. 2 The architecture of the D'Agents system [2].

The core system has four levels. The lowest level is an interface to each available transport mechanism. The next level is the server that runs on each machine. It keeps track of the agents running on each

machine, provides low level inter-agent communication facilities, receives and authenticates agents that are arriving from another host, and restarts an authenticated agent in an appropriate execution environment. The third level of architectures consists of the execution environments one for each supported agent language[2]. The last level of the architecture is the agent himself or herself, which execute in the interpreters and use the facilities provided by the server to migrate from machine to machine and to communicate with other agents [2].

Protecting the Machine:

Protecting the machine involves two tasks, **Authentication and Authorization and enforcement.** D'Agents like other mobile agent systems handle these tasks with public key cryptography and secure execution environments that perform authorization checks before each resource access.

Authentication:

Each D'Agent server distinguishes between two types of agents: owned and anonymous. An owned agent is an agent whose owner could be authenticated and is on the server's list of authorized servers. An anonymous agent is an agent whose owner could not be authenticated and is not on the server's list of authenticated servers [2]. Protecting an agent from a malicious machine is the most difficult security problem. Unless "trusted hardware" is available on each agent server, there is no way to prevent a malicious machine from examining or modifying any part of the agent that visits it. Thus the real problem is to prevent the machine from using the stolen information in a meaningful way and to detect tampering as soon as possible.

RSA public key cryptography is used to authenticate the agent's owner. Each owner and machine in D'Agents has a public-private key pair. The server can authenticate the owner if (1) the agent is digitally signed with the owner's public key or (2) the agent is digitally signed with the sending machine key, the server trusts the sending machine, and the sending machine was able to authentication the owner itself. D'Agents uses Pretty Good Privacy (PGP) for its digital signatures and encryption [2].

When an agent registers with its home server using the **begin** command, the registration request is digitally signed with the owner's private key, optionally encrypted with the destination server's public key, and sent to the server. The server verifies the digital signature, checks whether the owner is allowed to register an agent on its machine, and then accepts or rejects the request. If the agent and the server are on different machines, all further requests that the agent makes of the server must be protected to prevent tampering and masquerade attacks.

When an agent migrates for the first time with the **jump** command, the state image is digitally signed with the owner's private key, optionally encrypted with the destination server's public key, and sent to the destination server [2]. The server verifies the digital signature, checks whether the owner is allowed to send agents to its machine, and accepts or rejects the incoming agent. Once the agent has migrated, the owner's private key is no longer available. Therefore, for all subsequent migrations, the agent is digitally signed with the private key of the sending server. If the destination server trusts the sending server, and the sending server was able to authenticate the owner itself, the destination server considers the owner authenticated and gives the agent the full set of resource limits for that owner. If the destination server does not trust the sending server, or the sending server could not authenticate the owner itself, the destination server considers the agent to have no owner and will either (1) accept the agent as an anonymous agent or (2) reject the agent if it is not allowed to accept anonymous agents.

Authorization and enforcement:

Once the identity of an agent's owner has been determined, the system must assign access restrictions to the agent and ensure that the agent does not violate these restrictions. In other words, the system must guard access to all available resources. We divide resources into two types viz, **Indirect** resources can be accessed only through another agent. **Built-in** resources are directly accessible through language primitives for reasons of efficiency or convenience or simply by definition.

For indirect resources, the agent that controls the resource enforces its own access restrictions, rejecting or allowing requests from other agents based on the security vector attached to the incoming communication. Typically, the resource agent would simply check each request against an access list.

For built-in resources, the agent servers enforce several absolute access policies. For example, an agent can **terminate** another agent only if its owner is the system administrator or if it has the same owner as the other agent. The **name** operation reserves certain symbolic names for certain agent owners, preventing an arbitrary agent from masquerading as a service agent. For built-in resources, security is maintained using the language- specific security module and a set of language-independent **resource-manager** agents. When an agent requests access to a built-in resource, the security module forwards the request to the appropriate resource manager. The resource manager, which is just a stationary agent, implements a security policy that determines whether the access request should be approved or denied. The security module then enforces the decision. This approach provides a clean separation between security policy and mechanism, with the same resource managers making security decisions for all agents, regardless of their implementation language.

Protecting the Agents:

We consider two broad categories of tampering attacks [2]:

Normal Routing: The malicious machine allows the agent to continue with its normal itinerary, but holds the agent longer than necessary, charges the agent extra money, or modifies the agents code or state.

Rerouting: The malicious machine reroutes the agent to a machine that it would not have visited under normal circumstances, or prevents the agent from migrating at all and pretends that it is the next machine on the agents normal itinerary.

The partial solutions considered are as follows:

Partitioning: An agent can migrate through trusted machines only, such as a set of proxy sites under the control of a trusted Internet service provider. Then it either interacts with the untrusted resources from across the network using standard RPC, or sends out the child agents that contain no sensitive data and will not migrate again, instead just returning the result.

Components: Perhaps the most powerful idea is to divide each agent into components. Components can be added to agents as it migrates, and each component can be encrypted and signed with different keys. The agent's static code and the variables whose values never change make up one component, and would be signed with the owner's key before the agent left the home machine. If malicious machine modifies the code or variables, the digital signature becomes invalid and the next machine in the migration sequence will immediately detect the modification. In addition if an agent obtains critical information from a service, it can put this information into its own component. Similarly any code or data that is not needed unit the agent reaches a particular machine can be encrypted with that machines key. Finally, components make it easier for an agent to use the partitioning approach; an agent can leave a particular component behind on a trusted machine or can create and send a child agent that includes only certain components.

Self-Authentication: Although it is impossible to detect all malicious modifications to the state information of the agent, it is possible to construct an authentication routine that will examine the state information for any obvious inconsistencies or impossibilities. The authentication routine could examine the current set of components. Each agent server would execute the authentication routine, terminating the agent if the routine finds any inconsistencies. The authentication routine would run as anonymous and would only have the authority to examine the state image.

Migration History: It is possible to embed a tamper proof migration history inside the moving agent. This movement history allows the detection of some rerouting attacks, particularly if an agent is following a fixed itinerary and in combination with additional digital signatures, makes it impossible for a malicious machine to drop an entire component from the agent.

Audit logs: Machines should keep logs of important agent events so that an aggrieved agent or owner can request an audit from an authorized third party. The auditor would seek to identify the machine responsible for theft or modification and penalize that machine appropriately.

Encrypted Algorithms: Recent work involves encrypting a program and it's input in such a way that (1**)** the encrypted program is directly executable; (2) the encrypted program performs the same task as the original program. (3) the output from the encrypted program is also encrypted and can only be decrypted by the program encrypter [2].

6. Discussion

So far we have discussed in general, the various security techniques proposed to protect the hosts as well as the agents in mobile agent system. We mentioned authenticating credentials, access level monitoring and control, code verification, limitation techniques and audit logging as the techniques to protect the hosts against the agents. Authentication credentials do not guarantee that mobile agent will not be a harmful threat. Theft of cryptographic keys, poor implementation techniques and forgery can limit the degree of security provided. Some of the mobile agent code can be encrypted and will be hard to verify. This can be a potential problem if not taken care of. As far as the limitation techniques go, the time limits, duplication limits impose a considerable overhead. It might result in unnecessary duplication and other problems, which offset the benefits provided by the mobile agent system in general. Though audit logging is an effective techinque, it is not possible that all activities be logged and hence is not 100% effective.

We also talked about techniques based on fault tolerance and encryption to protect the mobile agent against the hosts. Although persistence makes mobile agents fault tolerant of host failures to some extend, it can cause an unwanted duplicate mobile agent, if the host is down for an extended time and the mobile agent was replaced assuming that it was destroyed. The technique of time limit and persistence often contradict each other. When a host crashes, time is wasted and this may cause the mobile agent to time out before it can complete its work. Thus it will not be able to complete its work in the given time limit and would remain in an incomplete state.

IBM aglets provide a very powerful tool to specify the host's policy and the aglet owner preferences. By introducing the concept of principles and by formulating a language for defining policies, the host and the aglet owner can now specify their policies. These policies control who can be allowed access and to what degree. Thus this is a very systematic approach and allows the enforcing entities to have a fine-grained control over the access of the resources.

D'Agents provide security to a group of machines using a combination of authentication, authorization and enforcement. Bu the authentication scheme has a major drawback. As soon as an agent leaves its group of trusted machines, it becomes anonymous as it migrates. Thus when we deal with machines that do not trust each other, an application that needs full access rights to accomplish its task cannot send a single agent that migrates through the machines since the agents becomes anonymous on the second jump. Hence we have to dispatch the agent to one machine and wait for results and then dispatch it to another machine. This increases the network traffic by a small percentage and hence adds to the overhead. Here other problem is that the PGP protocol used for authentication is slow. The various techniques mentioned for protecting agents do not provide guaranteed protection. Also some of these techniques involve a substantial overhead due to the additional encryption involved and hence there is a performance penalty.

7. Conclusion

The area of mobile agent security is in a state of immaturity. While numerous techniques exist to provide security for mobile agents, there is not at present an overall framework that integrates techniques into an effective security model. The traditional host orientation toward security persists, and the focus of protection mechanisms within the mobile agent paradigm remains on protecting the agent platform.

D'Agents is a simple but powerful mobile agent system that has been used in numerous distributed applications, particularly information retrieval applications. Several security related future remain however. Even taken together, the techniques cannot provide complete protection. In addition many of the techniques involve substantial cryptographic overhead forcing an agent to trade performance for protection. But we could realize adequate protection through a combination of these techniques. None of these solutions are currently implemented in the D'Agents System.

The Aglets Security System Model aims for alleviating the threats to the mobile agent systems. The model defines the principals within an aglet system and it also explains how agents migrate

and also points out their access to resources. With reference to the above developed model, security architecture is defined, introducing two dimensions of security architecture, the policy database and the owner specified preferences and also demonstrates how these elements control security unaware aglets. However the protection of an agent internal state against snooping and tampering and a generic solution to this problem is still a very challenging research topic.

References

[1] Michael Greenberg, Jennifer Byington, Holding, Harper, **"Mobile Agents & Security**" IEEE Communication Magazine, July 1998.

[2] Robert S. Gray and David Kotz and George Cybenko and Daniela Rus. **D'Agents: Security in a multiple-language, mobile-agent system**, volume 1419. In Giovanni Vigna, editor, Mobile Agents and Security, Lecture Notes in Computer Science, Springer-Verlag, 1998.

[3] Vijay Varadharajan," **Security Enhanced Mobile Agents**", CCS's Athens Greece,1-581131-204-4/00, ACM 2000.

[4] David Chess," **Security Issues in Mobile Code Systems**", *In* Mobile Agents and Security, volume 1419 of Lecture Notes in Computer Science. Springer-Verlag, 1998.

[5] Karnik, Tripathi, "**Design Issues in Mobile Agent Programming Systems**", IEEE Concurrency, July-Sep 1998.

[6] W.F. Farmer, J.D Guttman, V.Swarup, "**Security for Mobile gents: Issues and Requirements**" NIST, Baltimore1996, pp591-597.

[8] Danny Lange, Oshima,"**Seven Good Reasons for Mobile Agents**", Communications of the ACM 1999.

[9] C.G Harrison, D.M. Chess, A. Kershenbaum, "**Mobile Agents: Are They A Good Idea?**"Tech report, IBM T. J. Watson Research Center Yorktown Heights N.Y.

[10] G. Karjoth, D.B. Lange and M. Oshima,"**The Aglet Security Model**", IEEE Internet Computing, July-Aug1997

[11] F. Hohl,"**Protecting mobile agents with Black box security**", Proc1997Wksp.Mobile Agents and Security, Univ of Maryland, Oct 1997.

[12] W.M.Farmer, J.D.Guttman, and V.Swarup "**Security for Mobile Agents: Authentication And State Appraisal**", Proc, Computer Security 1996

[13] Robert Gray, "**Aglet Tcl : A flexible and secure mobile agent system"**. In proceedings of the 1996 Tcl/Tk Workshop pages 9-23 July 1996.

[14] Thomas Sander ."**On Cryptographic protection of mobile agents**". In proceedings of the 1997 Workshop on Mobile agents

[15] Duchess , B. Grosof, C. Harrison, D.Levine, C.Parris and G. Tsudik " **Itinerant Agents for Mobile Computing**" IEEE Personal Communication Vol 2 No.5, Oct19995