Security Enhanced Mobile Agents

Vijay Varadharajan

Distributed System and Network Security Research
School of Computing and IT, University of W.Sydney, Australia
(Visiting Researcher, Microsoft Research Cambridge, UK)
Email: vijay@cit.nepean.uws.edu.au

ABSTRACT

This paper describes a security model for mobile agent based systems. The model defines the notion of a security-enhanced agent and outlines security management components in agent platform bases and considers secure migration of agents from one base to another. The security enhanced agent carries a passport that contains its security credentials and some related security code. Then we describe how authentication, integrity and confidentiality, and access control are achieved using the agent's passport and the security infrastructure in the agent bases. We also consider the types of access control policies that can be specified using the security enhanced agents and the policy base in the agent platforms. We discuss the application of the security model in roaming mobile agents and consider a simple scenario involving security auditing in networks.

Keywords: Mobile Agents, Security Model, Secure Agent based Application

1. INTRODUCTION

Mobile code technologies are beginning to receive a great deal of interest from both industry and academia as they have a lot to offer towards achieving the vision of usable distributed systems in a heterogeneous network environment. The ability to move computations across the nodes of a wide area network helps to achieve the deployment of services and applications in a more flexible, dynamic and customizable way than the traditional client-server paradigm. Different types of mobile code paradigms have been proposed over the recent years [e.g. 1,2] such as code on demand, remote evaluation and mobile agents. In general, these paradigms are concerned with the movement of the executable content and the associated execution state between different computational environments (computer nodes (hosts) on the network). For instance, if one needs to perform a specified search of a large database through a computer network, it could be more efficient to move the program (mobile software agent) to the database server rather than have the client program communicate with the server via the classical client server computing especially if there is a wide area network separating the client and the server. Mobile code technologies provide several advantages over remote procedural call and message passing such as reduced network usage, increased asynchrony between clients and servers, increased concurrency and addition of client–specified functionality into servers. However, there lie some fundamental issues that need to be solved, the major ones being in the areas of security and robustness. For instance, proper security measures are required to control the ability of the agents downloaded from the network as well as to protect the agents from the nodes and while they are on transit. Key to this is the systematic understanding of security requirements and a comprehensive security model for mobile agents that is lacking at present.

In any distributed system, when a request for a certain service is received by one principal from another, the receiving principal needs to address at least two questions. Is the requesting principal the one it claims to be and does the requesting principal have appropriate privileges for the requested service? These two questions relate to the issues of authentication and authorisation. There are also other security concerns such as auditing, secure communication, availability and accountability. When it comes to mobile agents, the security issues become further complicated. First, there is a greater opportunity for abuse and misuse and second, mobile agents introduce specific issues that are unique, which challenge some of the common assumptions that are often made in secure systems design. For instance, it is not always easy to identify a particular mobile process with a particular known principal and to depend on the reference monitor approach to enforce the security policy. Recently, there have been several pieces of work related to mobile agent security [e.g. 3,4,5,6,7,8]. Work on Java and other script languages for remote programming such as safe Tcl [5] consider safe execution of untrusted code using sandboxing technique to address the problem of rogue agents. Some agent systems propose basic privacy mechanisms such as secure channel between hosts via encryption of agents and messages on transmission. Some offer authentication and integrity via the signing of agents and messages sent between hosts. Even fewer agent systems consider mechanisms to control resource consumption. However important challenges still remain. For instance, there is not a clear solution at present to tackle the problem of agent being attacked by the host where the agent resides. Furthermore, there has not been much work done on the development of a comprehensive and overall security model and architecture for mobile agent systems. These are necessary for widespread commercial adoption of mobile agent systems technology. The main aim of our project is to develop an overall security model and design and implement secure agent systems. The security model should be helpful for the designers to better understand the design choices involved in the development of secure agent based applications.

The paper is organized as follows. After discussing some security issues for mobile agents in section 2, in section 3 we describe a basic security model. The model defines the notion of a security-enhanced agent, outlines security management components in agent platform bases and considers how authentication, integrity and confidentiality, and access control are achieved. Section 4 first discusses the application of the model in roaming mobile agents and then considers the access control policies that can be specified using the security enhanced agents and the policy base. We also describe a simple scenario involving security auditing in networks. Finally section 5 concludes the paper by making some observations on trust in the outlined security model.

2. MOBILE AGENTS AND SECURITY

A number of models are being developed for describing agent systems. However for discussing security issues, to begin with, it is perhaps sufficient to use a simple model comprising two main components, namely an agent and an agent base. The agent consists of the code and state information needed to perform some computation. The agent is mobile in the sense that the agent can migrate from one agent base to another and the computation is mobile. We will refer to the agent base in which an agent is created and from where it originates as the home agent base or just home base. In general, an agent base can consist of one or more hosts and may support multiple computational environments. However we will assume in this paper that an agent base is supported on a single host and will use the terms agent base and hosts interchangeably. When an agent moves to another agent base, it may be referred to as the foreign base or the visiting base. Agent interpreters execute the agents at the agent base. The agent migration involves both the transfer of program code as well as data. The agent is considered to be autonomous as it has its own thread of execution after arriving at an agent base. The migration of an agent from one base to another is treated as a move operation and not as a copy operation.

There are a variety of ways of classifying security threats in such a mobile computation environment. We consider these in terms of agents attacking the agent base, agent base attacking the agents, agents attacking each other in an agent base and attacks against the agents when they are transferred over the network. In each of these categories, a variety of common security threats such as masquerading, unauthorized access, unauthorized disclosure, unauthorized modification and non-repudiation can arise. Let us briefly highlight some of these attack scenarios that are relevant for the development of the security model.

In terms of masquerading, the threat is one of "an agent claiming to be of a different identity". Often in security, when a program attempts a certain action, its identity is related to the principal (e.g. a user) who is requesting that action to be performed. In the case of mobile agents, this may be the principal who is sending the agent. Knowing that an agent comes from a particular sender is not often adequate when it comes to determining the level of trust that can be placed by the agent base on the agent. What may be required is that the principal that one trusts is the one who has created that particular agent. So it may be required to authenticate both the sender and the creator principals of the agent. Furthermore, often in mobile agent systems, many programs are obtained from unknown or untrusted sources. For instance, when a user clicks on a hypertext link and a program is downloaded, it may not be clear as to what level of trust that can be placed on the source from where the program comes from. Looking at the other side of the coin, the sender of the mobile agent may also wish to authenticate the visiting agent base host where the agent is to be executed prior to sending the agent. Hence mutual authentication is required in peer-to-peer transactions.

Having authenticated the mobile agent, the agent base needs to determine what actions the mobile agent is allowed to perform and does it have the necessary privileges to carry them out? In general, the authorization decision for a mobile agent to perform certain action can be based on a combination of privileges such as the privileges of the creator of the mobile agent, the sender of the mobile agent as well as the function of the program code and the state of the agent. The authorization mechanisms control the behaviour of the agent within the agent base thereby allowing protection of local resources. However as we mentioned above given that it may be difficult to identify the principal to whom the action can be attributed, it poses difficulties in determining whether or not the action should be permitted. Even more difficult issue is that of protecting the agent from a malicious agent base. This is because the agent base provides and controls the computational environment in which the agent operates. A malicious agent base can modify the agent's code and state and hence can affect the running of the agent. It can introduce unacceptable delays or simply not execute the code or even terminate the agent.

The migration of a mobile agent from one base to another over the network needs to be protected against unauthorized disclosure and unauthorized modification. Similarly any message that is sent by the mobile agent over an untrusted network needs to be protected. In principle, this can be achieved using cryptographic techniques such as encryption and cryptographic checksums. In practice, the main issues here are concerned with key management and key storage and the use of different cryptographic mechanisms. It is important to note that often it is not possible to hide anything within the agent without the use of cryptography. This in turn implies that an agent cannot transport its key in a form that can be used on untrusted hosts.

In a distributed environment, often the need for an entity to act on behalf of another arises. This is particularly true in the case of mobile agents, which by definition perform their actions on behalf of its sender and/or creator. A delegation is a temporary permit issued by a delegator to a delegate that authorizes the delegate to act on its behalf in performing certain actions. In this case, the target needs to verify whether the delegator has actually transferred the privileges to the delegate agent and whether it is the delegated agent, which is making the request. Revocation involves the removal of privileges and there are not easy solutions when it comes to distribution of revocation state to agent bases in a large network environment.

The ability to prove that an agent did a particular action can be important in many situations. For instance, when an action such as the purchase of an item is done by an agent at the remote host's agent server in an electronic commerce situation, it may be necessary for the agent base to prove that "this agent did this action at this time" in case of any disputes at a later point in time. The agent may also wish to record that "such and such an action was done at this agent base" for later use by the sender of the agent. For instance, an agent may wish to record that it purchased a video *abc* and paid *$x* for it on such and such a day.

Mobile agents can launch denial of service attacks to consume a large amount of agent base's resources. An agent can also launch denial of service attacks against other agents in the agent base. In general, this is a difficult problem to solve. However some of the security and availability techniques can be used to detect and curtail these attacks to a certain degree. In this paper, we will not be considering these attacks in our model. The main aim of this paper is to consider the design of a security model that contributes to the development of an overall security framework for mobile agents; in particular, we consider a security model that allows authentication of agents by agent bases, privilege based authorization of agents, privacy and integrity of agents in communications. We have also extended the model to address delegation and cloning issues. However, due to lack of space, these are described separately in another paper.

3. A Security Model for Agents

We assume that each agent base has a trusted security management component, which we refer to as SMC. We assume that each agent has an identifier that is unique over its lifetime and is independent of the agent base it is executing in. The identity of the execution environment in the agent base is same as the identity of the agent base (e.g. the URL)[1]. Agents communicate via messages. The group of agent bases that obey the same security policies are grouped together in a domain. Each domain has a security authority referred to as the Security Management Authority (SMA). The SMA interacts with the SMCs in the domain in the establishment and maintenance of security policies within the domain; it interacts with SMAs of other domains in inter-domain situations. In practice, we envisage that a SMA's role to include some of the functions of a Certification Authority.

Consider the situation where an agent arrives at an agent base and runs within the execution environment of the agent base. The SMC acts as a reference monitor and determines whether a certain requested action is allowed or not. The security policy within the SMC specifies the conditions under which a request is to be granted. The security policy can have varying degrees of granularity. It can be based on the identity of the principal that created the agent in the first place, the principal that sent the agent, on the characteristics of the agent such as the level of trust associated with it as well as the state of the target application. Let us now consider the situation whereby the agent carries with it certain security credentials. We refer to these agents as "security-enhanced agents" (SeA), which encapsulate not only the actions but also the privileges and other security attributes to perform these actions. That is, the agent carries part of the security information needed to make the authorization decision. In some sense, such a security enhanced agent based authorization model has characteristics of both access control lists and capabilities. The agent carries security information such as its privileges and the agent base's SMC contains security policy information such as what actions that an agent with certain privileges can perform and under what conditions; both these information are needed to determine whether an access request is granted or not. Furthermore, these security-enhanced agents are active structures in that they contain code that can be executed to make dynamic decisions. For instance, consider Cagent sent by Customer-C to agent base of Bank-A to withdraw some money from account of Customer-C. Cagent carries with it the identity and privileges of its sender Customer-C. The SMC has a policy base which has rules that specify under what conditions a customer is able to withdraw. If these conditions are satisfied then the money is transferred to Cagent (which is acting on behalf of Customer-C).

Hence so far, conceptually, our model has the following elements: Security Management Authority (SMA), Security Management Component (SMC) and Security enhanced Agents (SeA). Each agent-enabled resource has a SMC and each domain has an SMA. SMCs and SMA are trusted entities. The SMC provides mechanisms needed for the generation and validation of SeAs. The SMC maintains security policy information as well as other security information such as public and private keys and a list of some of the commonly used certificates and name servers. In an inter-domain situation, the SMA in one domain will be involved in negotiating with the SMA in another domain concerning the validation of SeAs that have originated from the other domain. We assume that each SMC has a public key and private key pair and so does each SMA. We also assume that the principals in this system have public key – private key pairs.

It is essential that a SeA should be unforgeable. Furthermore, the SeA should only have the capability to make those decisions for which it has been allowed to do so and it should not make any unauthorized decisions and requests. The SeA may need to prove to the target the identity of the sender or creator principal and that it has not been compromised in any way. The trust on a SeA is based on the guarantees provided by the creator principal's SMC and the sender principal's SMC. The guarantees provided by the SMCs should relate to both the code and data part of SeA. We will see how this is achieved below. In general, different SMCs can be trusted at different levels. This in turn implies that different SeAs signed by different SMCs will be trusted at different levels. Similar arguments apply to SMA as well. In fact, chain of trust on certificates starts with the SMA to SMC to SeA. A related issue is where should the agent be when it is making its decisions. In general, the agent should move to a host that is trusted (by its creator or sender) so that the execution environment does not affect the decisions illegally. This is necessary in many practical situations, as participating hosts often compete with each other and hence may be mutually suspicious of each other. For instance, one host may try to change the state of the agent in an unauthorized manner thereby causing different agent behaviour. This relates to the more general problem of how to protect the agent from its execution environment for which there is no easy answer at this stage.

[1] An appropriate qualifier if there are more than one execution environments within the agent base.

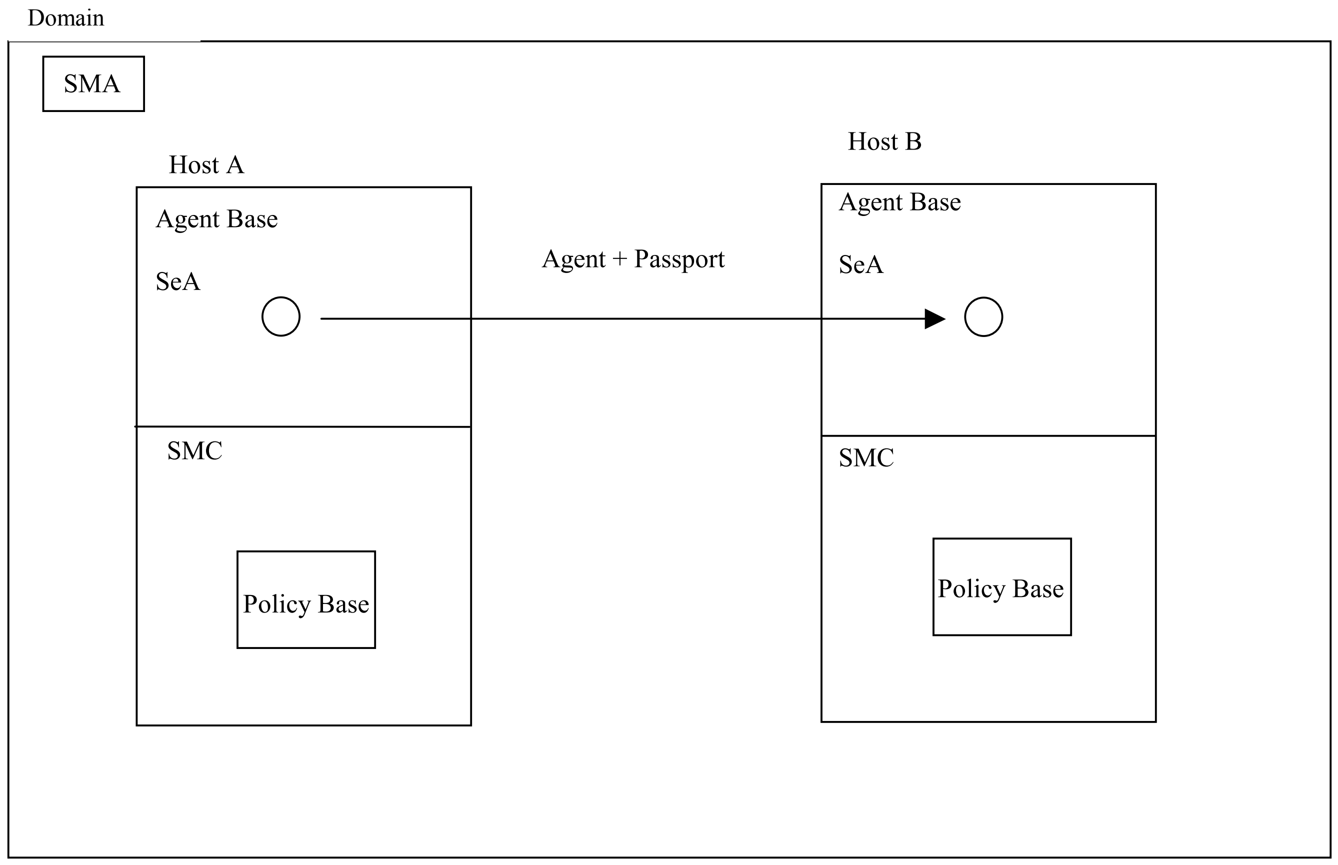

Figure 1. Security Enhanced Agent Environment.

3.1 Security Enhanced Agent Structure

The structure of SeA has several elements containing information on the identities and privileges of the principals such as the creator and the sender, validity of the privilege information and other information gathered during its propagation. The security enhanced agent can be thought of as an original mobile agent plus a "passport" where the passport specifies the security characteristics of the agent. The passport is used by the agent base to determine the types of actions that an agent can perform in its environment as well as what actions it can perform on the agent.

The basic structure of the security-enhanced agent is as follows:

- Identifier: (SeA Identifier, Creator-Principal Certificate, Creator-SMC Certificate, Timestamp, Lifetime)
- Privilege_Token: {<Identifier No, Privilege, Timestamp, Lifetime>}
- Agent_Code : (Security Code, Application Code)
- Data_Store : (Data, Propagation Path)
- Security_Tags : (Security-Tag-C, Security-Tag-S)

Identifier: Each agent has a unique SeA identifier is (e.g. a number) assigned at the time of creation. The Creator- Principal Certificate refers to the identity of the principal who created the agent. The principal certificate contains the usual elements such as the identity of the principal and its public key and the validity periods, all signed by the SMC using its private key. As there is a single SMC per host, the SMC also identifies the host (agent base) where the agent is created. The Creator-SMC certificate is signed by the SMA. The timestamp identifies the time at which the agent is created and the lifetime indicates the intended lifetime of the agent.

Privilege_Token : This token specifies the set of privileges of the agent. These are described in terms of operations or methods that are applicable to a set of objects in a particular class. These privileges are given by the creator principal to the agent. So if an agent has a privilege Pr given to it by the creator principal, then the interpretation is that, during execution, the agent is able to request an operation that requires the use of the privilege Pr. These privileges will be used in conjunction with the policy at the target agent base (host) in determining whether a request by an agent is to be allowed or not. This implies that even though the agent has the privilege Pr, it may not be able to perform an operation that requires this privilege due to other conditions that need to be satisfied in the policy base. Each privilege in the token has a number that acts as an identifier, a timestamp, which indicates the time at which this privilege was created along with a lifetime. The token is protected via appropriate cryptographic sealing, which is described below.

Data_Store : The SeA has a data store that contains the state of the agent in terms of the data variables and the initial values Subsequently, as the agent is executed, it also stores any results of operations carried out by the SeA. These results may need to be protected for confidentiality, integrity and origin authentication. The data store also contains a propagation list that consists of a list of identity of hosts visited by the SeA. The propagation list is created by the default method Propagation Path method described below. Encapsulation of results using cryptographic techniques is described below in section 4.

Agent_Code : The agent code is the core part of the SeA. It consists of a set of executable methods or code. There are two types of code: the Application Code and the Security Code. The Application Code is the normal application code of the agent and is specified by the creator principal. This is the code that does the useful work. The Security Code is a default set of methods automatically added when a SeA is created. The security code is concerned with the generation and maintenance of security parameters.The Security Code contains the following methods:

Propagation Path method allows the agent to create a list of identities of agent bases, as the agent moves from one agent base to another. Before the agent leaves an agent base, the identity of the target agent base is included in the propagation path. So when an agent leaves an agent base X for Y, its propagation path will have (X,Y).

Checksum method is used to calculate the cryptographic checksums needed in the generation of the various security tags (see below).

The Application Code contains the code to perform the required tasks. For instance, consider an agent whose task is to determine the travel itinerary, which involves booking of flights and hotels. In this case, the agent contains application code to search for suitable flights and hotels that can include the preferences of the travelling principal. Consider another example where a customer creates an agent to perform financial transactions. In this case, the application code will contain methods to perform operations such as withdraw from and deposit to accounts.

Security-Tag-C : This security tag is created by the creator principal and it contains the hash value *hash-c* of the original SeA signed using the private key of the creator principal. *hash-c* is calculated on the following: Identifier, Privilege_Token, Agent_Code (Application and Security Codes), and the initial fixed values in the Data Store. The Security-Tag-C provides an integrity check for the initial code and data contents of the agent along with the authentication of the creator agent. (Security-Tag-C|X refers to the security tag signed by X).

Security-Tag-S: Each sending client principal generates this security tag and hence there is one such security tag per sending principal. It contains the hash value *hash-s* signed using the private key of the sending principal. *hash-s* is calculated on *hash-c* and Data_Store as well as the contents of the request. The request contains sender principal identity, the operation along with a timestamp (see below). The Security-Tag-S provides an integrity check on the request and authentication of the sender principal, that is, where the SeA is coming from at present. (Security-Tag-S|X refers to security tag signed by X).

3.2 System Operation

At the time of creation of an agent, the creator principal defines the application code of the agent and the initial data values. A unique SeA identifier is generated and the certificate of the principal (signed by the SMC) that created the agent along with the certificate of the SMC (signed by the SMA) are added. The privileges that the agent is to have are defined, the default security code is added and the security tags generated. Hence in addition to the security methods mentioned earlier, the secure agent infrastructure should have methods to generate the identifier, collect the certificates, specify the privileges and generate timestamps and lifetimes. We will assume that such functions are available in security enabled agent host. E.g. Identifier method is used to generate the identifiers required for the SeA. It will generate a unique identifier number for the SeA, gather the certificates of the principal which is creating the agent and the SMC certificate and generate the timestamp and the lifetime needed for the SeA.

The migration of an agent to another host is achieved using an agent transfer protocol. The protocol request specifies the identity of the sending principal and its SMC certificate, the target agent base (host) to which it is being sent, the operation that is being requested, the time at which this request is made and the time period for which this request is valid. Some common examples of operations include transfer, execute, delete and retrieve. For transfer and execute operations, the SeA is included in the body of the request; for retrieve and delete operations, only the identifiers of the SeA are needed. Once again we will assume that the secure agent infrastructure has methods to generate the parameters needed in the request.

Let us consider an agent A created by principal X in an agent base XB which migrates to agent base YB to perform certain task. The agent has two privileges Pr1 and Pr2 granted by the creator principal. Its application code contains Program and has some Initial Data. It has security tags C and S signed by X.

```
{SeA
 Identifier
     <Name : A; Identifier : Id-A; Creator-Principal-Certificate
     : Cert-X; Creator-SMC-Certificate : Cert-XB;
      Timestamp : T-a; Lifetime : Period-a>
Privilege Token:
     { <Identifier : N1; Privilege : Pr1; Timestamp : T-1;
        Lifetime : L-1>
      <Identifier : N2; Privilege : Pr2; Timestamp : T-2;
       Lifetime : L-2> }
 Agent Code :
     Security Code
       {Propagation Path, Checksum}
      Application Code
        {Program}
  Data Store
      (Initial Data)
      (Propagation Path : {(XB,YB)}
   Security Tags
      (Security-Tag-C|X), Security-Tag-S|X)
```

Let us now consider the security characteristics. The agent consists of two parts: a "static" part, which has been produced at the creation time and is intended to remain the same and a "dynamic" part that changes as the agent moves from one host to another. The static part includes the application code and the creator granted privileges and these are integrity protected and signed using the private key of the creator principal. The dynamic part is integrity protected and signed using the private key of the sender principal. The latter also includes the contents of the request, which includes the operation and a timestamp. Hence the receiving agent base YB can verify the authenticity of the sender of the message and the timeliness of the request to perform the operation. YB can also check the integrity of the application code and satisfy itself that the code of the SeA is as sent by the creator originally. In order to do these verifications, the SMC of YB needs the public key of the principal that originally signed it. The SMC uses the Identifier in the SeA to get the identity and the public key certificate of the principal and uses the source SMC certificate to validate it. At this stage, we assume that the principal that originally created the SeA (and hence the source SMC) reside in the same domain as the target. Hence the target SMC can easily obtain the public key of the source SMC where the SeA was created. If this is not the case, then SMA is used to get the public key certificate of the required SMC in another domain. In any case, an implementation of this model will require one or more certification authorities, but their role is perfectly standard. If confidentiality of the agent is required, then this can be done via encryption either using the public key of the receiving principal or using a combined symmetric key and public key method. For instance, the agent can be encrypted using a symmetric key and the symmetric key protected using the public key of the receiving principal. In terms of access control, whether an agent is able to perform a certain operation when it is executing its program is determined using the privileges the agent carries along with the policy specified in the Policy Base of the SMC. YB is also able to read the privileges of the agent; in this case, granted by the creator. In general, the access decision is made using both the privileges granted by the creator principal and the privileges acquired by the agent as it moves from one host to another (proxy privileges). We will discuss access control and the policy base in section 4.2.

If all checks are successful, then the SeA code is run and the request is granted and the results returned to the SeA. A copy of the results is stored in the Data_Store in the SeA. The SMC of YB produces a signed hash digest of the results along with a timestamp using its private key and this is also stored. This signed hashed digest provides integrity and origin authentication of the results. In this case, the SeA, the creator principal as well as other principals will be able to read the contents of the store containing the results. If confidentiality protection of the results is also required, then the SMC of YB generates a secret data key and uses it to encrypt the results. The secret data key is encrypted using the public key of the client (the principal that requested the operation). A pure public key based approach can also be used where the data is encrypted using the public of the requesting client principal. In either case, with such confidentiality protection, once the results are stored in the Data_Store in the SeA, the SeA and other principals are unable to read the results. The SeA returns to the client principal, which can verify the signature of the YB's SMC on the results. This will provide the necessary guarantee to the client that it was YB that has generated these results. If confidentiality of results has been provided, then the results can be retrieved by first obtaining the secret data key and then using it to decrypt the results. Alternatively, the SeA might decide to reside within the remote host but pass back the results to the sending client. If this were to occur, once again protection of the results for confidentiality, integrity and origin authentication can be achieved in a manner similar to the above.

4. DISCUSSION : SECURITY MODEL

4.1 Roaming Mobile Agents

There are different scenarios when it comes to how mobile agents can be used to produce useful work. In one scenario, the agent moves from one host to another and performs some tasks in the foreign host. After performing the computations, the agent sends back the results to the originator in the form of messages or returns to the originator. An alternative scenario is that an agent moves from host to host performing certain tasks and either returning results time to time or storing all the results until it returns to the originator in the end. The latter roaming agents model introduces additional security issues. In this section, we consider how the security model described above deals with roaming mobile agents.

We will assume that when an agent moves from one host to another, it performs certain computations and generates results and these results are stored in the data store in the agent. We will also assume that the hosts visited by the agent are in a competitive environment and are mutually suspicious of each other. That is, the intermediary results produced by one host should be protected in some form against other hosts. We will consider later whether it involves integrity, confidentiality or both. The main objective is that the intermediary results gathered by the agent are not modified without detection and that the originator believes this to be the case. We will also assume that the path taken by an agent is not predetermined in advance. For instance, it may be that the originator can specify the set of hosts that the agent can visit but the agent might decide to visit only some of them and also the order in which these hosts are visited is not known in advance.

Consider a simple travel scenario where a customer (originator) wishes to book the air tickets for her trip. The customer creates and dispatches an agent which moves from one airline host to another collecting information on the prices offered for the client specified itinerary. Let us assume that the agents visits hosts A1 to An in some order and in each host, the agent uses its program to query the price for the tickets as specified by the customer. Hence the application code of the SeA contains the code for the itinerary and the query and the results of the query are stored in the data store.

Let us consider some of the security properties that are required in this scenario.

1. When the agent arrives at an airline host:

   (a) The agent base of the host should be able to verify that the code of the agent has not been tampered with (Code Integrity and Authentication).
   (b) The agent base should be able to check the privileges of the agent in determining whether the operation requested by the agent is to be allowed or not (Access Control)

2. When the agent returns to the originator:

(a) The originator should be able to identify which price belongs to which airline and that the prices have not been tampered with. That is, both integrity of data and data origin authentication are required (Data Integrity and Origin Authentication).

(b) Only the originator should be able to read the quoted prices. That is, prices quoted by an airline need to be kept confidential (Data Confidentiality)

(c) No illegal data can be inserted into the data store unless the agent has visited the corresponding host and that the appending of the data to the store has been explicitly done by that host (Insertion Protection).

(d) No legally appended data can be deleted from the store without detection (Deletion Protection).

(e) No host can repudiate that the data that has been appended to the agent (Non-repudiation)

The first set of requirements is similar to those considered earlier. There is no addition of privileges (proxy privileges) as the agent moves from one airline to another in terms of proxy tokens. We will also assume that there is no cloning of the agent and the same agent actually moves from one host to another. When the SeA arrives at a host, the SMC in the agent base is able verify the Security Tags C and S using the public key certificates in the usual manner. The successful verification of Security-Tag-C leads to checking the integrity of the agent's code and the identity of the creator of the code. The verification of Security-Tag-S leads to checking of the identity of the sender of the agent and the integrity and timeliness of the request. The checking of the Security-Tag-C also ensures that the Privilege_Token has not been illegally modified and that the agent has the given privileges. These can in turn be used in making the access decisions when running the agent program.

Let us now consider the second set of requirements in detail. Assume Pi is the price offered by the airline Ai. Let the initial data in the data store D(o) consists of some random number Ro chosen by the customer and the next host to be visited A1. In fact, let D(o) be Offer(o) = ([Ro, A1]PK-C, {hash(Ro,A1)}SK-C). That is, Ro and A1 are encrypted using the public key of the customer and a hashed version of Ro and A1 are signed using the private key of the customer. When the agent arrives at the airline A1, the airline's agent base adds its price to the data store. It adds Offer(1) = ([A1, P1, R1, A2]PK-C, {hash(A1, P1,R1,A2,Offer(o)}SK-A1) to the data store. Now D(1) has (Offer(o), Offer(1)). Hence D(i) has (Offer(o), …., Offer(i-1), Offer(i)) where Offer(i)=([Ai, Pi, Ri, Ai+1]PK-C, {hash(Ai, Pi, Ri, Ai+1, Offer(i-1)}SK-Ai). At the end, the agent returns to the originator C. This idea of chaining the results is similar to that used which have been used in nested and chained delegations in [9] and in [11] using secret hash chains. Now let us see whether the security properties in (2) above are satisfied.

(a) Data Integrity: First let us suppose that an attacker wishes to change the price offered by Ai-1. Recall that Offer(i-1) = ([Ai-1, Pi-1,Ri-1,Ai]PK-C, {hash(Ai-1,Pi-1,Ri-1,Ai,Offer(i-2)}SK-Ai-1). Let the modified offer be Offer'(i-1). But D(i) will still have Offer(i) within its hash function. Hence it is not possible to modify Offer(i-1) without modifying Offer(i). Hence by induction, we can see that none of the offers can be modified without detection. Let us now suppose that the airline Ai wishes to change the price Pi-1 offered by Ai-1. If it changes Offer(i-1) then the because Offer(i-1) has been signed by Ai-1, changes cannot be done without detection. Note also that the price and the identity of the airline offering that price are included within the signature and protected using the public key of the customer. Hence the customer can identify which price belongs to which airline and verify that the prices have not been tampered with.

(b) Data Confidentiality: This is achieved if the public key based encryption method used to secure the price in the offer is secure. The price Pi is encrypted along with a random number using the public key of the customer and hence only the customer will be able to read the price.

(c) Insertion Protection: Let us assume that a new price is inserted in between Offer(i) and Offer(i+1). Given that the offers are chained by including them in the calculation of the next hash function, this is not possible. The argument is similar to that used in the data integrity case. However, successful insertion can be achieved at the end of the chain. The attacker can append at the end of a chain n his own offer by calculating

Offer(n+1) = ([An+1,Pn+1,Rn+1,C]PK-C,
{hash(An+1,Pn+1,Rn+1,C,Offer(n)}SK-n+1.

So it is possible to add a fake offer before sending the agent back to the originator.

(d) Deletion Protection: Given a chain of prices from a list of airlines, it is not possible to delete a price offer in between without detection. The argument is similar to that given for data integrity. However it is possible to delete all the offers after i and then add a final one at the end (as in insertion)

(e) Non-Repudiation: If an airline has provide a price offer then it cannot repudiate that offer at a later time because that offer has been signed by the airline as part of the hash function calculation.

Remark

This example illustrates the situation where it is necessary to chain the data added to the data store as the agent moves from one agent base to another. In the model described in the previous section, each agent base that the agent visited (where certain computation was performed and results produced) protected the results for integrity and confidentiality depending on the requirements. In this example, the results of the previous agent base are included in the checksum calculation at the current agent base. This chaining of results gave additional security properties involving insertion and deletion protection, though insertion at the end of the path can still occur.

4.2 Access Control: Privileges and Policy Base

The ability of an agent to perform a certain action in the visiting agent base is determined by the privileges given to the agent as well as the policy statements stored in the policy base in the visiting base. Let us now consider some of the access polices that can be specified using this approach.

An agent's request to perform an operation is of the form access(s,o,m,p) where s identifies the agent session, o is the object which is being accessed, m identifies the operation or method being invoked and p is the set of parameters for the invocation. The identifier and the privilege token are available within the agent session. The policy base contains rules as to which operations are allowed for which principals. These rules are of the form : If <condition expressions>, then <action>. The condition expression tests the identifiers and privileges of the agent, its creator and sender principals and their agent bases along with the conditions associated with the parameters and values of the object being invoked. We adopt a language-based approach to specify the rules in the policy base. For a detailed description of such a language, refer to [10]. Here we only briefly describe some of features of the language using some common policy examples.

The principals in our system include the agent, the applications that create and send the agents, the users, the agent bases (with the SMCs) where the agent is created and executed as well as the SMA. As agent bases (and their SMCs) can belong to certain domains, we use the notion of domain to group together some principals. For instance, we can have DOM1 : Agent Base Domain = {SMC1, SMC2}. The language also provides a number of operations such as set intersection (and), set difference (or), equality (=) and test for inclusion (€). We will use the notation such as agent.creator to denote the creator principal of the agent. Using these constructs, we can have variety of policy specifications. Here are some examples of the policy rules based on agents and their passport attributes:

- Agent Identity based Rule
  - If <agent.name = Name> and <agent.id = Id> , then grant access to operation m
- Agent and Agent Base Identities based Rule
  - If <agent.name = Name> and <agent.id = Id> and <agent.base = SMC-X> then grant access to operation m
- Creator and Agent Base Rule
  - If <agent.creator = XYZ> and <agent.base = SMC-X> then grant access to operation m. For any agent created by XYZ in agent base SMC-X, grant access to operation m.
- Creator and Sender based Rule
  - If <agent.creator = XYZ> and <agent.sender = ABC> then grant access to operation m. For any agent created by XYZ and sent by ABC, grant access to operation m.
- Domain based Rule
  - If <agent.base € DOM1> where DOM1 = {SMC-X, SMC-Y}, then grant access to operation m.
- Role based Rule
  - If <agent.creator.role = {Role}> then grant access to operation Op.
  - If <agent.creator.role = {Role1}> and <agent.sender.role = {Role2}> then grant access to operation m.
  - If <agent.creator.role € ROLES> and <agent.sender.role € ROLES> then grant access to operation m.
- Privilege based Rule
  - If <agent.creator = XYZ> and <agent.privileges = {Priv}> and <is-current(Priv) = true> then grant access to m. That is, if the agent has a privilege Priv given by the creator agent XYZ and if the privilege is not expired, then grant access to operation m.

Policy base might also contain rules that are dependent on certain conditions on attributes of the request as well as of the object being invoked. The attributes that are part of the request are referred to as the transaction attributes. The attributes that are part of the rules in the policy base are referred to as rights attributes. The conditions are expressions that compare the transaction attributes with the rights attributes. Cond(op, trans-attr,rights-attr) is a logical expression that specifies how transaction attributes are compared with the rights attributes for the operation op. In general, a Cond expression is composed of Cond Elements and Cond Elements can be combined using *and/or* logical operators and nested parenthesised expressions. The *and* operator has higher precedence than the *or* operator when no parentheses are used. Consider a simple condition expression for a request operation done by an agent in a financial application. Assume that the request operation is a Withdraw operation where an agent wishes to withdraw $1000 from a bank account object Bank-Account-A. The transaction attributes associated with this operation are say Account-Type, Amount and Balance. The rights attributes specified in the policy base are Cheque-Limit and Credit-Limit. Cond expression is (Account-Type: Savings) and ((Amount <= Cheque-Limit) or (Balance <= Credit-Limit)). If this evaluates to true, then the condition is satisfied. Here are some examples of policy rules based on such conditions on transaction and rights attributes and combined with the agent attributes.

Privilege and Constraint based Rules

- If <agent.creator = xyz> and <agent.privileges = {Priv}> and <is-current(Priv) = true> and <Cond(trans-attr, rights-attr) = true>, then grant access to operation m.
- If <agent.creator = xyz> and <agent.base = SMC-X> and <agent.privileges = {Priv}> and <is-current(Priv) = true> and <Cond(m,trans-attr, rights-attr) = true>, then grant access to operation m.
- If <agent.creator = xyz> and <agent.sender = abc> and <agent.privileges = {Priv}> and <is-current(Priv) = true> and <Cond(m, trans-attr, rights-attr) = true>, then grant access to operation m.
- If <agent.base € DOM1> where DOM1 = {SMC-X, SMC-Y} and and <agent.privileges = {Priv}> and <is-current(Priv) = true> and <Cond(m, trans-attr, rights-attr) = true>, then grant access to operation m.

Hence it can be seen that a variety of access control policies such as identity based, role based and privilege based access policies can be specified using the passport capabilities of the agent and the rules in the policy base.

4.3 Security Auditing in Networks Scenario

Consider a network environment with a number of hosts and a central management station (CM) responsible for administration of these hosts. As part of the administration, the central station performs security auditing of these hosts. The auditing checks are specified in an agent *auditagent* and the agent is dispatched by the central station to the remote hosts. The *auditagent* arrives at a host and executes certain operations such as reading the status of certain files and objects and their permissions, and checks whether they conform to the rules specified in its audit checks. Each of the hosts has its own policy base that determines what agents can do in its environment. Let us assume that an *auditagent* wishes to perform certain tasks such as (read-file, filename), (read-permissions, filename) and (read-permissions, directory). These are part of the application code. A secure *auditagent* is generated by the system administration principal sys-adm. The principal sys-adm grants the agent a privilege called Audit which is specified as part of the Privilege Token of the secure *auditagent*.

{SeA *auditagent*

Identifier

<Name : Sherlock; Identifier : Id; Creator-Principal-Certificate

: Cert-sys-adm; Creator-SMC-Certificate : Cert-CM;

Timestamp : T; Lifetime : Period>

Privilege Token

{ <Identifier : N1; Privilege : Audit ; Timestamp : T1;

Lifetime : L1>}

Agent Code

Security Code

{Propagation Path, Checksum}

Application Code

{read-file, read-permissions}

Data Store

(Initial Data)

(Propagation Path : {(CM, H)}

Security Tags

(Security-Tag-C(sys-adm), Security-Tag-S(sys-adm)) }

When the agent arrives at the host H, the host checks the certificates and the security tags to authenticate the identity of the agent, the integrity of the application and security codes, the integrity and the validity of the privileges as well as the propagation path of the agent. When the agent is executed in the host and when it performs operations such as read-file or read-permissions, the privilege specified in the token is used in conjunction with the policy rules in the host's policy base to determine whether access is to be granted.

In this case, the policy base may have the following simple rule:

If <agent.creator = sys-adm> and <agent.sender = sys-adm> and <agent.privileges = {Audit}> and <is-current(Audit) = true> then grant access to op where op = {read-file, read-permissions}

Alternatively, there could be additional conditions based on the transaction and rights attributes of the operation. For instance, there could be a condition Cond(read-file, filename, allowed-directories) which constrains which files can be read by the agent. E.g. Cond(read-file, filename, /users) = true if and only if filename € /users directory.

In general, different hosts can have different sets of policy rules. Even within a single organization, it is conceivable that different rules might arise depending upon the function of the organization unit being audited and the trust to be placed on the agent generated by the sys-adm principal.

6. CONCLUDING REMARKS

We conclude this paper by making some observations on trust in the outlined security model. A fundamental assumption in this model has been the existence of a trusted component SMC in each agent-enabled host. This component has been trusted by system principals to perform security related operations such as signing and encryption, and to generate integrity checksums and timestamps. There is no reason why each of the SMCs should be trusted to the same level. In fact, it is not difficult to have different levels of trust associated with different SMCs. For instance, we had earlier examples of policy rules based on groups of trusted hosts. This trust concept was further refined in the case of proxy tokens which led to different levels of trust associated with the privileges granted by different principals in different hosts. Finally, the trust on the code and the data of the agent are based on the cryptographic checksums generated using the private keys of different principals (the creator and the senders). All this, in some sense, helps to extend the trusted environment of an agent's home base to other agent bases. For instance, when an agent arrives at a foreign agent base, trust on its code is determined from the security checksum generated and signed by the creator principal. The creator principal's signature is verified using its certificate, whose trust is in turn dependent on the SMC certificate. The level of trust on the SMC itself is based on the trusted group that it belongs to. Similarly, trust on the privileges carried by an agent is dependent on the signature of the principal that signed it, which is in turn dependent on the SMC signature and the trusted group it belongs to. The agent's passport, which contains the privileges and other security credentials, is used in conjunction with the policy base in the SMC of the host to determine whether an agent should be accepted by an agent base and what actions it should be allowed to perform. As the agent moves from one host to another accumulating results, the partial results are protected for confidentiality and integrity in such a way that an intermediary host cannot read or modify or insert data without detection. Signing has been used to ensure that a host cannot later dispute the information that it has previously entered into the data store of the agent. These techniques provide some measures for detecting tampering of agent code and data as it moves from one host to another. However they do not offer protection against untrusted malicious hosts where there is no trusted component. This is because an agent is completely susceptible to the agent base (and host), which controls the computational environment where the agent operates. The usual approach is to try and reduce the size and functionality of the trusted component to a minimum. The problem of protection of agents against untrusted malicious hosts still remains a challenging issue.

Acknowledgements

I would like to thank Microsoft Research Cambridge, UK for hosting me at Cambridge when this paper was written. In particular, I would like to thank Roger Needham and Luca Cardelli for their valuable comments on the paper as well as the anonymous referees.

REFERENCES

(1) D. Chess et al.,``Itinerant Agents for Mobile Computing'',IBM Research Report RC20010 1995.

(2) D. Lange, M. Oshima, Programming and Deploying Java Mobile Agents with Aglets, Addison-Wesley, 1998

(3) J. Tardo and L. Valente, "Mobile Agent Security and Telescript", Proc. IEEE CompCon, 1996.

(4) D. Dean, E. Felten and D. Wallach, ``Java Security : From HotJava to Netscape and Beyond'', Proceedings of the 1996 IEEE Symposium on Security and Privacy, , USA.

(5) R. S. Gray, "A Flexible and Secure Mobile Agent System ", 4th Annual Tcl/Tk Workshop Proc, 1996.

(6) S. Berkovits, J. Guttman, V. Swarup, "Authentication for Mobile Agents", in Mobile Agents and Security, Editor Vigna, LNCS 1419, 1998

(7) G. Karjoth, D. Lange, M. Oshima, "A Security Model for Aglets", Internet Computing, July 1997.

(8) T. Sander and C. F. Tschudin, "Towards Mobile Cryptography", Proc. of the 1998 IEEE Symposium on Research in Security and Privacy, USA.

(9) V. Varadharajan, P. Allen, S. Black, ``An Analysis of the Proxy Problem in Distributed Systems'', Proc. of the 1991 IEEE Symposium on Research in Security and Privacy.

(10) M. Hitchens, V. Varadharajan, "Tower: A Language for Role based Access Control", Accepted for Publication, IFIP SEC2000.

(11) G. Karjoth, N. Asokan, C. Gulcu, "Protecting the Computation Results of Free Roaming Agents", Second International Workshop on Mobile Agents, MA'98, LNCS-1477, 1998.