CHAPTER 2
RELATED WORK AND LITERATURE SURVEY

2.1 INTRODUCTION

An agent is “a person whose job is to act for, or manage the affairs of other people”. In the context of computers, a software agent is a program that performs certain tasks on behalf of its owner. The software agent may be mobile or static. A mobile agent can move or be moved around the network, but a static agent works on one host computer on the network, including accessing resources which are on hosts other than the host on which the agent is executing. A new and emerging technology for computers to communicate in the distributed environment is via the mobile agent. The advantages of the mobile agent are too big that should not get from the static agents and conventional web technology.

With its features, the mobile agent is one of the most prominent technologies for various intranet and internet applications. However, mobile agent technology has not been perfected because of deployment concerns such as reliability and security. The reliability deals with the persistency of the mobile agent when the malfunction or the agent itself is under a denial of service attack. Works on reliability can be found in Rothermel and Strasser (1998a, 1998b), Zhong et al (2004). The security problems, however, need much more effort, since there are still many unsolved issues. The aim of this chapter is to provide a deep insight into mobile agent technology and the state of the art. Section 2.2 describes the existing works to protect the platform
against the malicious agent attacks, and also it describes the works related to policy-based models to protect the platform. The attacks performed by malicious agents are denial of service, shutting down the platform, unauthorized access, etc.

While protecting the mobile agent platform against a malicious agent, sometimes the mobile agent platform also plays the malicious role against the agent. Section 2.3 briefs the works related to protect the mobile agent from malicious platform attacks.

Despite the agent code attack and platform attack, the multi-hop mobile agent defends the data (information) from a single or a set of colluded host attackers. During the journey, an agent may also have the possibility to visit the malicious host. The malicious host may change the data gathered from the preceding host by appending the dummy data or replacing the existing data with fake data or deleting/altering some part of the data. To protect against these types of attacks, various available solutions in the literature survey are discussed in Section 2.4.

Apart from the protection of the mobile agent, the recovery of the agent is a major issue to the originator. Mobile agent protection only identifies the attack and attacker, but is not able to recover the attacked mobile agent. To recover the mobile agent and its information collected from multiple remote servers, there is a need for a recovery mechanism. The recovery mechanism will trigger the agent again (where the agent is killed) to continue its journey. From this perception, Section 2.5 describes the related works to recover the mobile agent.
2.2 MOBILE AGENT PLATFORM PROTECTION

The mobile agent from a malicious platform will have the intention to disrupt remote platforms. To protect the platform from the malicious agent, software-based fault isolation is proposed to implement fault isolation within a single address space. It is meant to separate the untrusted code in separate software-enforced fault domains, so that the distrusted code cannot modify other data or execute another code except through an explicit cross-fault domain RPC interface. Access to system resources can also be controlled through a unique identifier associated with each domain referred to as sandboxing (Wahbe et al 1993).

2.2.1 Sandbox

Sandbox (Wahbe et al 1993) is the protection model which provides a separate location for the agent to execute in the environment. Every agent executes in a secure environment and any access to anything outside this environment is strictly controlled by a security manager. This is achieved by the static checks on the code before it is executed. Static check is the byte code verifier (type correctness, no stack overflow or underflow, code containment, registration initialization and object initialization) (Xavier 2003). Sandboxing suffers from an “all-or-nothing” problem that either allows complete access if the signer of the mobile agent is trusted or very limited access for all mobile agents (Yao 2004).

2.2.2 Code Signing

Joseph and Luis (1996) proposed a model to protect the platform by signing the agent code. It is to authenticate the incoming agent by the platform. The agent owner should digitally sign the agent code for the authenticity of an agent, its origin and its integrity. If the host trusts the signer
of the mobile agent, it will allow it to carry out its execution with full access
to all the resources available in the execution environment.

However, code signing does not have the “all-or-nothing” problem. Currently, the common
technique used is to marry the code signing and sandbox techniques. The idea is to use code signing to identify partially trusted mobile agents that can be executed under a less restrictive security policy. For instance, an authenticated mobile agent from a registered user may have access to the entire database of the host, while an unknown mobile agent can only access a small publicly available part of the database. (This approach has actually been implemented in Java 2 (Gong 2002) by certain Web browsers (for example Netscape Communicator) in the context of signed Java applets).

The major drawback in this model is that the malicious host can also sign the agent code to pose as the trusted host. It means that the host (may be malicious) can send the malicious agent with the genuine digital signature and certificates. It is not able to protect it by the code signing method. With the same drawback, Braun and Rossak (2005) also proposed the Central management of access permissions, in which each agency has its own security management responsible for authentication and authorization of incoming agents.

2.2.3 Path History

Path History (Ordille et al 1995, Ordille 1996) is a model, in which an agent has to maintain the authenticable record of the prior platforms visited by it. Based on that record the newly visited platform can determine whether to process the agent or not. Path history includes the signed identity of each agent platform visited, and the identity of the next platform to be visited. The
path history could get very lengthy and thereby increase the cost of verification.

### 2.2.4 Proof Carrying Code

Necula (1997) described a software mechanism, called the “Proof Carrying Code”, based on well-known principles from logic, type theory and formal verification, that allows a host system (the code consumer) to determine with certainty that it is safe to execute a program supplied by a distrusted agent (the code producer). For this to be possible, the distrusted code supplier must provide with the code a safety proof that attests to the code’s safety properties.

The “Proof Carrying Code” is a prevention technique, while code signing is an authenticity and identification technique used to deter, but not prevent the execution of an unsafe code (Jansen 2000). In this technique, safety policies can be defined to stipulate not only standard requirements such as memory safety, but also more abstract and fine-grained guarantees about the integrity of data-abstraction boundaries. In this aspect, the “Proof Carrying Code” goes beyond the safety guarantees provided by other mechanisms such as software fault isolation as described above. The approach is problematic since it requires advance knowledge of the security policy under which the code will be executed and it is currently not possible to produce proofs for the arbitrary code automatically.

The summarized drawbacks (Ramya 2000) of all the existing models are given in Table 2.1.
Table 2.1 Drawbacks of the Existing Agent Platform Protection Models

<table>
<thead>
<tr>
<th>Existing Models</th>
<th>Drawback</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sandboxing</td>
<td>It takes slightly increased execution time for distrusted modules.</td>
</tr>
<tr>
<td>Path histories</td>
<td>The path history could get very lengthy and thereby increase the cost of verification. This technique does not prevent the agent platform from behaving maliciously, but it acts as a deterrent.</td>
</tr>
<tr>
<td>Proof Carrying Code</td>
<td>The main practical difficulty lies in generating the safety proofs and also there is a need for a technique to limit the potentially large size of proofs.</td>
</tr>
<tr>
<td>Code Signing</td>
<td>Malicious host can also generate signed certificate for its agent</td>
</tr>
</tbody>
</table>

2.2.5 Policy Model to Protect the Platform

To protect the platform from the mobile agent and allow it to process, an access control policy is essential because the agent will be from multiple hosts that may be the administrator or trusted third party servers or clients or neighbor servers. The agent that originates from multiple hosts can have different privileges. For this, the policy based protection model is required to differentiate the agent and allow it to use the resources.

For the policy model, Paul (1997) built Secure European System for Applications in a Multi-vendor Environment (SESAME) multi-domain distributed-system security architecture with the use of authentication and privilege certificates. Both users and applications are controlled in the same way when accessing protected resources - they must first obtain proof of their privileges in the form of a Privilege Attribute Certificate (PAC) and then present it to a target application when requesting resource access.
The target application may, in turn, access another target using the delegated privileges. Access control information is represented generically to facilitate mapping to the different types of access controls on targeted resources. SESAME follows a delegation-only model for authorization. PAC revocation is avoided by relying on short delegation periods. While the focus of SESAME is solely on static client-server type applications, it provides a good example of the underlying framework needed when applying certificate-based solutions for distributed system security.

Later, Jansen (2001) introduced an attribute certificate with the policy certificate. Attribute certificates convey policy rules associated with an agent. A policy certificate conveys policy rules governing the behavior of all agents that may attempt to visit an agent platform or a specific place of an agent platform. Remote hosts verify the certificate which consists of the identification (ID), version, attribute, etc., and the agents are allowed to execute in their environment based on their levels. An agent with different certificates will increase the verification time during the journey.

Kumari et al (2007) introduced a System Agent (SA) for each platform. The mobile agent has to request the SA for every migration to a destination system. The System Agent of the local host contacts the System Agent of the remote host and verifies or correlates the policies. The SA of the local host serializes the mobile agent to make it capable of transferring to a System Agent of the remote host. The SA in the remote host receives mobile agent and de-serializes it. It also initiates the execution of the mobile agent from its current state. It has the drawback of having the system agent in each platform and communicates with the destination system before dispatching the agent.
2.3 MOBILE AGENT PROTECTION

Protecting the mobile agent from the malicious remote host is a critical task, because the remote host is responsible for agent execution. In order to protect the mobile agent, the trusted node method (Farmer et al 1996), trusted hardware (Wilhelm 1999) and co-operating agent’s models (Roth 1998, Merwe and Solms 1996) are developed. However, a malicious platform may cause an agent to operate incorrectly; the existence of enough replicates ensures the correct end result. The drawbacks are the high cost to develop the model, the limitations of the agent mobility and difficulty in deciding the best offer.

2.3.1 Code Obfuscation

Code obfuscation (Libes 1992) is to make the agent’s program illegible and thus difficult to manipulate. The mobile code is rearranged before it is moved to a remote site. The technique used to modify the code makes it difficult to reengineer the code but preserves its original behavior. Based on this technique, black box (Hohl 1998) security is proposed to protect the mobile code against malicious hosts. No eavesdrop or modification attacks are possible. However, a serious problem with this technique is that there is no known algorithm or approach for providing such security. To make it more practical, Hohl redefined the “blackbox” to exist only for a certain known time interval. Hohl has proposed several conversion algorithms. In short, the task of a conversion algorithm is to generate a new agent (out of an original agent), which differs in code and representation but yields the same results. In addition, the newly generated agent is assumed to be hard to analyse. In this context, “hard” means that the analysis required to understand the agent’s functionality should take as much time as possible for an arbitrary attacker. Nevertheless, it has major drawback (Yao 2004):
- It does not protect against every possible attack. For example, it is still possible for the host to deny execution or to return wrong system call results. Also, it is still possible for an attacker to read or to manipulate data and code, but since he cannot determine the role of these elements for the application, the attack results are random.

2.3.2 State Appraisal

A State Appraisal (Farmer et al 1996) approach is proposed to identify the alterations of the agent’s state information by malicious attacks. The author or owner of the agent has to create the appraisal functions and that could be added to the agent’s code. Appraisal functions are used to determine what privileges to grant to an agent, based both on conditional factors and whether the identified state invariants hold. An agent whose state violates an invariant can be granted no privileges, while an agent whose state fails to meet some conditional factors may be granted a restricted set of privileges. When both the author and owner digitally sign the agent, their respective appraisal functions are protected from undetectable modification. This model is possible to protect the agent state from deceptive alterations.

2.3.3 Execution Tracing

Vigna (1997) developed a model to identify the malicious modifications on the agent code. The platform where the agents execute is required to create and retain a non-repudiation log of the operations performed by the agent and to submit a cryptographic trace. A trace consists of a sequence of statement identifiers and platform signature information. If any malicious results occur, the appropriate traces and trace summaries can be obtained and verified; then the malicious host can be identified. This model lacks in the size and number of logs to be retained. Another problem with this
model is the lack of accommodating multi-threaded agents and dynamic optimization techniques.

2.3.4 Replication and Voting

The idea of replication and voting (Schneider 1997) is more than a single copy of an agent performing a computation; multiple copies of the agent are used. Although a malicious platform may corrupt a few copies of the agent, enough replicates avoid the encounter to successfully complete the computation. This technique seems appropriate only for specialized applications where the agent can be duplicated without problems, the task can be formulated as a multi-staged computation, and survivability is a major concern. One obvious drawback is the additional resources consumed by the replicate agents.

2.3.5 Mutual Itinerary Recording

Allowing an agent’s itinerary to be recorded and tracked by another co-operating agent and vice versa (Roth 1998) is a mutually supportive arrangement. While an agent moves between agent platforms, it has to convey the preceding, current and succeeding platform information to the co-operating peer through an authenticated channel. The peer maintains a record of the itinerary and takes appropriate action when inconsistencies are noted. Also, it will prevent the revisiting attack. For some applications it is also possible for one of the agents to remain static in the home platform. The major drawback of this model is its higher cost for setting up the authenticated channel and the inability of the peer to determine which of the two platforms is responsible if the agent is killed. However, the mobile agent static itinerary (Borrell et al 1999, Mir and Borrell 2003) and dynamic itinerary (Carles et al 2008) are protected but not efficiently for the code and data.
2.3.6 Encrypted Functions

Computing with the encrypted function (Sander and Christian 1998) model is to execute the agent (program) as an enciphered function without being able to discern the original function; i.e., instead of equipping an agent with function $f$, the agent owner can give the agent program $P(E(f))$ which implements $E(f)$, an encrypted version of $f$. An agent’s execution could be kept secret from the executing host as would any information carried by the agent. Next to the encrypted function, Salima et al (2006) develops the model to give the mobile agent code security by dividing the agent code into modules. Each host can only use its respective module with the help of the symmetric and asymmetric keys transmitted earlier between the local and remote host. The modules other than the allotted module are not able to recover by the remote hosts because they do not have the relevant key. It is the best model to protect the mobile agent from attacks.

These two approaches have the following drawbacks: nothing prevents a malicious host from running the encrypted agent code again and again with some other input, and continuing in this way until the agent has leaked the secret completely. Also, it will eliminate the active mobile code that performs some immediate action on the host (Algesheimer et al 2001).

2.3.7 Environment Key Generation

Environmental Key Generation (Riordan and Bruce 1998) allows constructing an agent to take predefined action when some environment condition is true. The agent’s private information can be encrypted and only revealed to the environment once the predefined condition is met. The major drawback of this approach is that the condition is not met on a particular host, and the private information is not revealed to the platform. Also, the agent platform typically limits the capability of an agent to execute the code created dynamically, since it is considered as an unsafe operation.
2.3.8 Code on Demand (CoD)

Code on Demand (Wang et al 2000) is a model to preserve the integrity of the mobile agent by the gradual construction of the agent’s code in which new modules can be added and those redundant can be entrenched at the runtime. The CoD consists of two functions:

(i) Addition: Agent function modules can be dynamically added to the existing agent code to form an upgraded version. Each function module should include the function code, and also, the proper digital certificate regarding the place from which this code is fabricated, namely, the source agent factory, as a proof of its validity. For example, a digital signature of the agent factory over the collision resistant hash value of the function module can be helpful to prove both the authenticity and integrity of the code. The addition of any particular code modules should get authorization from the proper parties.

(ii) Deletion: Agent function modules can be dynamically deleted from the agent body to form an upgraded version. The deletion of the function modules should also get authorization from the respective parties.

For example, after the original agent provided by the factory is sent out by the user, it travels to a series of hosts to complete some tasks. When it finds out that it needs some additional modules, it will seek to download these modules from the code provider or the agent originator.

The deficiency in this approach is that, downloading the modules from the agent originator when the modules are requested, will increase the process time of the remote host. Also, the set of malicious remote hosts will
raise the network traffic volume by continuously seeking for modules to the 
agent originator without using them.

2.3.9  **Factor of Time**

The Factor of Time (Grimley and Monroe 1999) is to identify the 
malicious host based on the duration the agent is occupied by the hosts. If we 
provide a limited time to execute the agent, then the chance to tamper with the 
code is limited. If that time elapsed in the untrusted host, the agent must shut 
down or move to the next host specified on its itinerary. This method is not 
effective because the legitimate host with multiple processes will take more 
than the allotted time. This time lapse situation falsely termed the legitimate 
host as malicious.

2.3.10  **Other Models**

Apart from the above protection models, some other models are 
also developed to protect the mobile agent. Suen (2003) had proposed a 
model to protect the agent. Here, in addition to the agent data, the signed hash 
code of the mobile agent is added to protect the agent code. The mobile agent 
from the current host will dispatch to the next remote host with the crypto 
information. The host receiving the mobile agent has to verify all the signed 
hash codes appended with the information from all the preceding hosts and 
allow the agent to execute. The drawback with this model is, the signed hash 
code of the agent code available with all the preceding host’s encapsulated 
offers will increase the size of the mobile agent, and also it will increase the 
data integrity verification time at all the remote hosts.

Benachenhou et al (2006) developed a model to protect the mobile 
agent with the help of the clone available in the trusted server. The mobile 
agent that visited the host has to be compared with the clone and authenticate
its integrity. If any modification is done by the malicious host on the agent code, the comparison with the clone will identify that, and also, it will recover the agent. It is not an appropriate model to protect the mobile agent by having the clone and execute it in the trusted server with the information from the mobile agent’s execution environment. The summarized drawbacks (Ramya 2000) of all the above existing models are given in Table 2.2.

**Table 2.2 Drawbacks of Existing Agent Tampering Protection Models**

<table>
<thead>
<tr>
<th>Existing Model</th>
<th>Drawbacks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Digital Signatures</td>
<td>Although, a malicious server cannot brainwash the agent, it can make the agent to forget the data. This could be a source of nuisance.</td>
</tr>
<tr>
<td>Time Limited Black Box Security</td>
<td>The Black Box test attack: The agent platform can try to determine the characteristics of the inside of the black box, by executing it with different input parameters and by watching the effects. The recorded reactions can be formal results like output values or characteristic “activity patterns”.</td>
</tr>
<tr>
<td>Mutual Itinerary recording</td>
<td>The security issue here is the inability of the peer to determine which of the platforms is responsible if the agent is killed.</td>
</tr>
<tr>
<td>Execution Tracing</td>
<td>The detection process is only triggered occasionally, based on suspicious results or other factors. Also, the size of the logs could get unmanageable.</td>
</tr>
<tr>
<td>Environmental Key Generation</td>
<td>i) A platform that completely controls the agent could simply modify the agent to print out the executable code upon receipt of the trigger, instead of executing it.</td>
</tr>
<tr>
<td></td>
<td>ii) An agent platform typically limits the capability of an agent to execute the code created dynamically, since it is considered as an unsafe operation.</td>
</tr>
</tbody>
</table>
Table 2.2 (Continued)

<table>
<thead>
<tr>
<th>Existing Model</th>
<th>Drawbacks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Computing with encrypted functions</td>
<td>This scheme doesn’t prevent denial of service, replay, experimental extraction etc.</td>
</tr>
<tr>
<td>Partial Result Authentication Codes</td>
<td>A malicious platform could retain copies of the original keys or the key generating functions of an agent.</td>
</tr>
<tr>
<td>Replication and Voting</td>
<td>This technique seems appropriate only for specialized applications where the agent can be duplicated without problems, the task can be formulated as a multi-staged computation, and survivability is a major concern. One obvious drawback is the additional resources consumed by the replicate agents.</td>
</tr>
<tr>
<td>Factor of Time</td>
<td>Process time lapse falsely termed the legitimate host as malicious.</td>
</tr>
<tr>
<td>Code on Demand (CoD)</td>
<td>Increase traffic volume by getting modules to upgrade the agent.</td>
</tr>
</tbody>
</table>

2.4 MOBILE AGENT INFORMATION PROTECTION

The multi-hop mobile agent will roam in the distributed network to perform the computation and gather information on behalf of its owner. A malicious server in the distributed network may expose, modify, insert or truncate the data of the agent collected from the preceding host to benefit itself. For this, security mechanisms are mainly required in terms of the integrity of the data.

For this, Yee (1997) proposed the PRAC (Partial Result Authentication Codes) to protect the mobile agent information. Yee classifies his algorithm into three types:
i) Simple MAC-based PRACs

ii) MAC-based PRACs with one-way functions

iii) Publicly Verifiable PRACs

These three types of PRACs are the key associated. The key of the current host will be erased by the mobile agent prior to migrating to the next server or host. The agent has the list of encryption keys for each server to be visited. Even though the PRAC scheme ensures data integrity, agents must determine how many keys they need to carry before leaving the owner. Also, the agent has to carefully protect the keys and erase the used key once they complete their actions on each server. PRAC only provides weak forward integrity (Wong et al 1997). Also, this is impossible for multi-hop mobile agents in a real network environment (Xu et al 2006).

Karjoth et al (1998) extended the ‘yee’ schemes with a set of protocols called KAG (Karjoth Asokan Gulcu). It consists of a digital signature and hash functions to protect a chain relation with the help of different combinations of cryptographic mechanisms. Each host generates a signing key for its successor and certifies the corresponding verification key. Using the received signature/verification key pair, a host signs its partial result and certifies the new verification key of the next host. This technique will avoid the modification attack in the above scheme, but not a two-colluder attack. In this attack two visited hosts can collude together to discard the partial results collected between their respective visits.

Karnik and Tripathi (1999) uses an encrypted checksum to build a backward chain relation to link an agent’s previous result with that of the agent that generated the data at the currently visited host. It guarantees that only new data can be added to the results of the agent collected and no data can be deleted from them. Karnik et al agent contains three kinds of protected
objects: Read-only objects whose tampering can be deleted, encrypted objects for specific servers and secure append-only list of objects. But it does not support two-colluder attacks in the multi-hop mobile agent environment (Xu et al 2006). It is the compact method than the above Karjoth et al (1998) scheme.

Corradi et al (1999) developed a protection protocol which uses backward and forward chaining. The chain consists of the cryptographic proof of the previous hosts, the results of current host and identity of the next host. Like the above models, this protocol also cannot defend the two-colluder truncation attacks (Xu et al 2006).

Cheng and Wei (2002) enhanced the KAG protocols with a co-signing mechanism to defend the two-colluder truncation attacks. Here, a preceding host co-signs a result generated at the current host. Attackers need their preceding non-attacker to co-sign fake offers when they launch two-colluder truncation attacks, and then their actions can be detected. Here also, the publicly verifiable forward integrity property generates a pair of one time secret private and public keys at each host for its successor. As Yao et al (2003) and Songsiri (2005) pointer out, the security assurance relies on the assumption that the predecessor does not leak the secret key used by its successor. This requirement to potentially malicious is not realistic. To defend stemming attack, a special case of two-colluded truncation attacks, the protocols needs to be modified and requires two-way authentication (Xu et al 2006).

Yao et al (2003) and Songsiri (2005) used the Trusted Third Party (TTP) mechanism to protect the mobile agent’s information. TTP is required to protect the itinerary information directly or indirectly. The major drawback with this is, that they need one TTP at least, and the mobile agent needs to communicate with it ceaselessly, so the TTP will become a bottleneck and
even cause single-point failure. Also, it is not easy to find a TTP in the open Internet.

Zhou et al (2004a, 2004b) analyzed and developed a protocol to overcome the weakness of the Cheng et al model to defend two-colluder truncation attack. The Zhou et al (2004b) model is the same as the Cheng et al (2002) scheme and uses a co-signing mechanism in which a host needs the preceding host’s signature on its encapsulated offer before sending it to the next host. Even though it is able to protect the data from the two-colluded attack, it cannot defend against multiple-colluder (more than two) truncation attacks.

Xu et al (2006) proposed a protocol to defend against the two-colluder truncation attack with the help of the one hop backward and two-hop forward chaining method. It also defends against the multiple colluder truncation attacks, fake stem attack and then the interleaving attack. It will defend the multiple colluder truncation attacks as long as any two of the colluders are not adjacent. This protocol can be extended to overcome the adjacent attacker limitation, but the protocol process will be too complex to implement.

2.4.1 Multi-Agent based Information Protection

In addition to the chaining mechanism, the multi-agent models were proposed to protect the information from the malicious host. In this multi-agent protection model, Roth (1998) also proposed a model to protect the information from malicious hosts. For this protection, they need two agents who move in different host fields to record itinerary information mutually. Their main flaw is that the two agents need authenticated communication (which is hard to achieve). Next, the information protection protocol developed by McDonald et al (2004) divides the agents into three:
task agent, data computation agent and data collection agent. It is not fit for a multi-hop mobile agent with dynamic itinerary, because the path information must go along with the data computation agent supposing the agent is a multi-hop mobile agent with a static itinerary, and the protocol provides nothing to protect the path information carried by the mobile agent. Later, Jiang et al (2004) proposed a protection protocol with the use of oblivious transfer and encrypted circuits but it has the same problem as that of the McDonald et al (2004) model which is pointed out by Silei et al (2008a).

Silei et al (2008a) proposed the idea with two kinds of agents - Task Agent (TA) and Secondary Agent (SA). It also uses the functionalities and mechanism provided by the trusted computing technology. The task agent (TA) moves freely in the network to perform certain computation. The secondary agent (SA) moves to an anonymous third party who has trusted platform module on it, then uses the data computed by TA to extend some platform configuration register in the trusted platform module irrevocably. The difficulty with this model is identifying the anonymous third party and the other one is transferring the task agent or its offers to the secondary agent for integrity verification.

Apart from having two agents, again Silei et al (2008b) proposed a model of having two dimensional chain relations among multi-agents to protect the data collected by the mobile agent. Silei et al (2008b) uses the mobile agent clone for the two dimensional chain like the simple agent. Having the clones, the communication overheads added to the net and computing overheads added to the user (suppose the host is visited twice by the agent and its clone). One obvious drawback is the additional resources consumed by the clones.
2.5 MOBILE AGENT RECOVERY

Apart from the protection schemes, the recovery of the mobile agent is most important in the mobile agent environment, because an agent destroyed in the $n^{th}$ remote host will lose all the preceding $(n-1)$ remote hosts information, and also, the agent originator should once again send the agent to collect the information from all the $N$ remote hosts, but there is no guarantee that the agent will return to the originator in the second round. Hence, it is required to recover the mobile agent when it is either in an unsafe mode or it is destroyed (agent is killed by the host). An unsafe mode is the modification of the agent code or the modification of the information collected from the preceding hosts.

Pair processing (Gray and Reuter 1993) is a famous technique for improving process reliability. It is a collection of two processes which provide a service. One is considered as the primary and the other one is considered as the shadow. If the primary gets any changes, then the shadow would also get the changes. If the primary fails, then the shadow will take over. The two primary and shadow processes ping each other to determine that each is still alive. Unrh et al (2005) also applied this pair process model to his Semantic-Compensation-Based Recovery model. This pair process is not applicable to the colluded host’s attack in a multi-hop mobile agent environment.

There is a significant attention within the mobile agent fault tolerance community concerning the loss of mobile agents at remote agent servers, that fail by crashing; hence researchers concentrated on the shadow model (Silva et al 2000, Strasser et al 1998, Schneider 1997, Pleisch and Schiper 2000, Silva and Popeschu-Seletin 2000, Mohindra et al 2000). Vogler et al (1997) developed a model that allows a mobile agent to inject a replica into a stable storage upon arriving at an agent server. However, in the event of an agent server crash, the replica remains unavailable for an unknown period.
Pears et al (2003) proposed the mobile shadow scheme which includes a pair of replica mobile agents, master and shadow, to survive remote-agent-server crashes. The master is created by its home agent server $h_0$ and is responsible for executing a task $T$ at a sequence of hosts described by its itinerary. Initially, the master spawns a shadow $\text{shadow}_{\text{home}}$ at its home agent server before it migrates and executes at the first agent server in its itinerary, i.e. $AG_i$. Before the master migrates to the next host in the itinerary, i.e. $AG_{i+1}$, it spawns a clone or $\text{shadow}_i$ and sends a die message to $\text{shadow}_{\text{home}}$. The $\text{shadow}_i$ repeatedly pings the agent server $AG_{i+1}$ until it receives a die message from its master.

- **Shadow**: A shadow or clone in the preceding server will terminate when it receives a die message from its master. This signifies that the master has completed the execution at $AG_{i+1}$ and spawned a new clone $\text{shadow}_{i+1}$ to monitor the agent server $AG_{i+2}$. However, assume that the master is lost due to an agent server crash at $AG_{i+1}$. In this case, the $\text{shadow}_i$ at $AG_i$ detects the crash of its master, spawns a new clone $\text{shadow}_i$ and proceeds to visit the agent server $AG_{i+2}$. Consequently $\text{shadow}_i$ is the new master.

- **Master**: A master pings its shadow at $AG_{i-1}$ concurrently with the execution of task $t$. In the normal case the master completes its execution and spawns a new clone shadow’ to monitor the next host, $AG_{i+1}$. Before the master migrates, it will send a die message to terminate the shadow at $AG_{i-1}$. If the master detects a shadow crash it spawns and dispatches a “replacement shadow” to the preceding active agent server. Before the master migrates to the next host in its itinerary it sends a die message to terminate the replacement shadow.
The major drawback of this scheme is the timeout overhead and mobile shadow overhead. The timeout overhead represents the resending of the agent and the mobile shadow overhead represents the time for pinging the shadow with the master running in the remote agent server. Despite this recovery, it does not concentrates on the agent server crash (i.e., if an agent server crashes, then the agent will automatically crash). In addition, this scheme is also not applicable to recover the agent from the colluded attacks.

Other than these, Wong et al (2004) used the witness agent and Beheshti et al (2007) used the two co-operating agents in the name of the witness agent to identify and recover the dead agent. Both these models are capable of dealing with server failure and single host attack, but not capable of dealing with the recovery of the multi-hop mobile agent from colluded attacks.

2.5.1 Blocking Attack

The host with malicious intentions who refuses to transmit the agent to the next host, either on a predetermined path or determined by the agent based on dynamically gathered information, is the blocking attack. To overcome the blocking attack, Shao and Zhou (2006) introduced the \(<t_f,n>\) fault-tolerant scheme where \(t_f\) is the fault-tolerant execution time and \(n\) is the fault tolerant roaming hop \(n\). The fault-tolerant execution time \(t_f\) is used by the agent owner to periodically track the agent’s location. Here, if the preceding host identified the failure of the agent in the succeeding host, it will send the clone to the agent home without continuing to the remaining host in the itinerary.

Apart from this, the agents have to send partial offers or acknowledgements to the home about their being alive for a fault-tolerant execution time \(t_f\) or when the total number of hops increased in or multiples of
$n$ fault tolerant roaming hop $n$. Even though this has effective recovery, and is capable of avoiding the blocking attacks, it is not capable of protecting agent against colluded attacks.

### 2.6 OPEN ISSUES IN A MOBILE AGENT SECURITY ENVIRONMENT

Figure 2.1 shows the various models that exist to protect the mobile agent environment. There are open issues in these models. The issues are briefly described below.

In platform protection models, the agent is overloaded by carrying the history of the previous host visit, and the attack codes (Denial of Service, Unauthorized Access, killing the agent platform, etc.) are not detected in the early stage of protection.

In agent code protection models, more encipher computations are required to verify the integrity of the agent and it is not possible to prevent a false malicious claim by the malicious host against the legitimate preceding hosts.

In data protection models, there is no possibility to protect against multiple colluded attackers in adjacent places in a multi-hop mobile agent.

In the recovery of mobile agent models, there is no possibility to recover the agent from colluded attacks in a multi-hop mobile agent.
Figure 2.1 Existing Models to Protect Mobile Agent Environment
2.7 SUMMARY

Security is the bottleneck for the incorporation of the mobile agent in a wide network. In this chapter, the existing state-of-the-art to protect the mobile agent code and data, mobile agent server protection with policy model and agent fault tolerance model are discussed with their drawbacks.

The open issues in the multi-hop mobile agent environment for data protection and recovery of agent are the colluded attacks, no earlier prevention for platform attack and no protection for a malicious claim. To overcome all these security issues in both the agent and platform, the forthcoming chapters propose advanced security models.