CHAPTER 2

RELATED WORKS AND LITERATURE SURVEY

2.1 INTRODUCTION

An agent is “an entity whose work is to act for, or manage the needs of other people”. In the context of computers, a software agent is a program that performs some tasks on behalf of its owner. The software agent may be static or dynamic. A Mobile Agent can move or can be moved around the network. But, while considering a static agent, it works on one host computer on the network. The functionality of a static agent is to access resources which are on hosts other than the host on which the agent is executing. The Mobile Agent technology is a new and emerging technology for computers to communicate in the distributed environment. The advantages of using a Mobile Agent are more than those obtained from static agents and conventional web technology.

The Mobile Agent is one of the most well known technologies for various intranet and Internet applications, with its exclusive features. However, Mobile Agent technology has not attained higher level of perfection because of deployment concerns such as reliability and security. The reliability deals with the persistency of the Mobile Agent when subjected to a malfunction or the agent is supposed to encounter a Denial of Service attack. Works related to reliability can be found in Rothermel and Strasser (1998) 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 understanding of the Mobile Agent technology. Section 2.2 describes the existing works to protect the Mobile Agent platform against Malicious Agent attacks. It also describes the works related to policy-based techniques 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 itself plays the malicious role against the agent. Section 2.3 briefs the works related to the protection of Mobile Agent from malicious platform attacks.

In spite of the attacks in agent code and platform, the Multi-Hop Mobile Agent defends the data (information) from a single or a set of colluded host attackers. During the journey, an agent has the possibility to visit malicious hosts. The malicious host may change the data gathered from the preceding host by appending dummy irrelevant 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 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 Home platform. Mobile Agent protection identifies only the attack and attacker and is not capable of recovering the attacked Mobile Agent. To recover the Mobile Agent and the information collected by it from multiple remote servers, it is necessary to provide a proper recovery mechanism. The recovery mechanism helps the agent again (where the agent is killed) to continue its journey. From this perspective, Section 2.5 describes the related works to recover the Mobile Agent.
2.2 MOBILE AGENT PLATFORM PROTECTION

The main intention of the Mobile Agent generated from malicious platforms is to destroy remote platforms. To protect the platform from the malicious agent, software-based fault segregation is projected to implement fault isolation within a single address space. It is designed specifically 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 using 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 mechanism which provides execution environment separately for the agent. Every agent executes in a secure environment and access to any resource outside this environment is strictly controlled by a security manager. This is achieved by checking static 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 allows either 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 technique to protect the platform by agent code signing process. It is to authenticate the incoming agent by the platform. The agent code will be digitally signed by the agent owner, 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 merge 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. At any circumstance, an authenticated Mobile Agent from a registered user may have access to the entire database of the host, whereas an unknown Mobile Agent can only access a small publicly available part of the database.

The major drawback in this system is that the malicious host can also sign the agent code to expose itself as a trusted host. It means that the host which 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. The Central management of access permissions proposed by Peter Braun and Wilhelm Rossak (2005) has the same drawback.

2.2.3 Path History

Path History (Ordille et al 1995, Ordille 1996) is a system, in which the authenticable record of the platforms will be maintained by an agent visited previously. Based on those records, the newly visited platform can determine whether to process the agent or not. Path history includes the signed identity of each platform visited by the agent, and the identity of the next platform to be visited. The path history could be very lengthy for large distributed systems 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 whether it is safe to execute a program supplied by a distrusted agent (the code producer). To make this possible, the distrusted code supplier must provide a safety proof with the code that attests to the safety properties of the code.

The “Proof Carrying Code” is a prevention technique, while code signing is an authenticity and identification technique used to find, but not preventing 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 advanced 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.

2.2.5 Policy Technique to Protect the Platform

To protect the platform from the Malicious Mobile Agents and allow it to process, an access control policy is essential because the agents are originated from multiple hosts that may be the administrator or trusted third party servers or clients or neighbour servers. The agent that originates from multiple hosts can have different privileges. For this, the policy based protection technique is required to differentiate the agent and allow it to use the resources.

Paul (1997) proposed the policy technique, which built Secure European System for Applications in Multi-vendor Environment (SESAME) multi-domain distributed-system security architecture which is designed using
the authentication and privilege certificates. Both users and applications are controlled in the same way when accessing protected resources. First of all they must obtain proof of their privileges in the form of a Privilege Attribute Certificate (PAC) and then present it to application to which the resource access requisition is made.

The application, in turn may 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 technique for authorization. PAC revocation is avoided by relying on short delegation periods. While the focus of SESAME is mainly on static client-server type applications, it provides a good example of the core framework needed when using 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 behaviour of all agents that may attempt to visit an agent platform or a particular place of on 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 its control migration to a System Agent of the remote host. The SA in the remote host receives Mobile Agent and de-serializes it and 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.

The drawback of all the existing systems are summarized in Table 2.1

**Table 2.1 Drawback of Existing Agent Platform Protection**

<table>
<thead>
<tr>
<th>Existing Systems</th>
<th>Drawback</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wahbe et al (1993) - Sandboxing</td>
<td>It takes increased execution time for distrusted modules. Suffers from “all-or-nothing” problem.</td>
</tr>
<tr>
<td>Ordille et al (1995&amp;1996) - Path Histories</td>
<td>The path history could get very lengthy and thereby increase the cost of verification.</td>
</tr>
<tr>
<td>Joseph and Luis (1996) - Code Signing</td>
<td>Malicious host can also generate signed certificate for its agent.</td>
</tr>
<tr>
<td>Necula (1997) - 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>Kumari (2001) - Policy Technique to Protect the Platform</td>
<td>The main drawback is time taken for Certificate verification and communication before dispatching the agent.</td>
</tr>
</tbody>
</table>

**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 techniques (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 performed to make the agent program illegible and thus making it difficult to manipulate. The mobile code is shuffled before it is moved to a remote site. The technique used to modify the code makes it hard to retain the original code but preserves its unique behaviour. 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 particular 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 analyze. In this context, “hard” means that the analysis required understanding the agent’s functionality should take as much time as possible for an arbitrary attacker. The major drawback (Yao 2004) is that 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 which could be added to the agent’s code. Appraisal functions are used to determine the privileges that need to be granted 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 system is possible to protect the agent state from deceptive alterations.

2.3.3 Execution Tracing

Vigna (1997) developed a system 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 for 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 system lacks in the size and maintaining number of logs. An additional problem with this system is the lacks in accommodating multi-threaded agents and dynamic optimization techniques.
2.3.4 Replication and Voting

The idea of replication and voting (Schneider 1997) aims in using more than a single copy of an agent to perform a computation. That is, multiple copies of the agent are used. Even though a malicious platform may corrupt a few copies of the agent, the other replicates may perform the computation and complete the task. This technique seems to be 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 work. One noticeable 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 amidst agent platforms, it has to communicate 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 prevents 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 system is its higher cost for setting up the authenticated channel and the inability of the peer to determine which of the two platforms are 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) is to execute the agent (program) as an enciphered function without being able to discern the original function; i.e., instead of preparing 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 system for 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 unable to recover by the remote hosts because they do not have the relevant key. It is the best system to protect the Mobile Agent from attacks.

These two approaches have the following drawback: 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 eliminates 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 restricts 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 system 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: To form an upgraded version of the existing agent code, agent function modules must be dynamically added. 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: To form an upgraded version from the agent body, agent function modules can be dynamically deleted. The deletion of the function modules should also get authorization from the respective parties.

The deficiency in this approach is that, downloading the modules from the agent originator will increase the process time of the remote host when the modules are requested. 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 period 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 the time elapsed is more 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 consume more than the allotted time. This time lapse situation falsely terms the legitimate host as malicious.

2.3.10 Dersingh Technique

Dersingh et al (2009) presents an hierarchy based role assignment. The contextual terms are represented as ontology by means of Ontology Web Language (OWL). The OWL reasoning engine provides reasoning about class membership, classification and consistency. Based on all the above, the access management is done. In access management, a policy editor which is capable of accessing domain ontology to acquire domain vocabularies is used. Any change in the environment is updated in the semantic knowledge base. The introduction of Extensible Access Control Markup Language makes the task so easier to be performed.
2.3.11 Other Techniques

Apart from the above-mentioned protection mechanisms, some other systems are also developed to protect the Mobile Agent. Suen (2003) had proposed a technique 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 system is, the signed hash code of the agent code available with the entire 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 system 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 method 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 drawback of all the above existing systems are summarized in Table 2.2.
### Table 2.2 Drawback of Existing Agent Tampering Protection

<table>
<thead>
<tr>
<th>Existing Systems</th>
<th>Drawbacks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Libes (1992)-Code Obfuscation</td>
<td>Functionality of the newly generated agents is very difficult.</td>
</tr>
<tr>
<td>Vigna (1997)-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>Schneider (1997)-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>Sander and Christian (1998)-Computing with encrypted functions</td>
<td>This scheme doesn’t prevent denial of service, replay, experimental extraction etc.</td>
</tr>
<tr>
<td>Roth (1998)-Mutual Itinerary recording</td>
<td>The security issue here is the inability of the peer to determine which of the platforms are responsible if the agent is killed.</td>
</tr>
<tr>
<td>Riordan and Bruce (1998)-Environmental Key Generation</td>
<td>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. 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>
<tr>
<td>Grimley and Monroe (1999)-Factor of Time</td>
<td>Process time lapse falsely termed the legitimate host as malicious.</td>
</tr>
<tr>
<td>Wang et al (2000)-Code on Demand (CoD)</td>
<td>Increase traffic volume by getting modules to upgrade the agent.</td>
</tr>
<tr>
<td>Dersingh Technique-OWL</td>
<td>Provides a semantic knowledge based system. The implementation of this system in the real time is more complex.</td>
</tr>
</tbody>
</table>
2.4 MOBILE AGENT INFORMATION PROTECTION

In the distributed network the Multi-Hop Mobile Agent will roam to perform the computation and gather information on behalf of its initiator. 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.

2.4.1 Partial Result Authentication Codes

For this, Yee (1997) proposed the PRAC (Partial Result Authentication Codes) to protect the Mobile Agent information. Yee classifies the 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 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. Though even the PRAC scheme ensures date 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).
2.4.2 **KAG Protocols**

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.

2.4.3 **Backward chain relation**

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.

2.4.4 **Backward and forward chaining**

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 systems, this protocol also cannot defend the two-colluder truncation attacks (Xu et al 2006).

2.4.5 KAG Protocol with co-signing

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) pointed out, the security assurance relies on the assumption that the predecessor does not leak the secret key used by its successor. This requirement is potentially malicious 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).

2.4.6 Trusted Third Party Technique

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 at least one TTP, 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.
2.4.7 Two-Colluder Truncation Attack Scheme

Zhou et al (2004) analyzed and developed a protocol to overcome the weakness of the Cheng et al system to defend two-colluder truncation attack. The Zhou et al (2004) system 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.

2.4.8 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 hops $n$. Even though this has effective recovery, and is capable of avoiding the blocking attacks, it is not capable of protecting the agent against colluded 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.

2.4.9 Multi-Agent Based Information Protection

In addition to the chaining mechanism, the multi-agent systems were proposed to protect the information from the malicious host. In this multi-agent protection, Roth (1998) proposed a system to protect the information from malicious hosts. For this protection, they need two agents which move in different host fields to record itinerary information mutually. The 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. This will not be suitable for a Multi-Hop Mobile Agent with dynamic itinerary, because the path information must be carried 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) system 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 system 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 system 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 adds communication overhead to the network and computing overhead to the user (suppose the host is visited twice by the agent and its clone). The drawback of this system is the additional resources consumed by the clones.

The drawback of all the above existing systems are summarized in Table 2.3.
<table>
<thead>
<tr>
<th>Existing System</th>
<th>Drawback</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yee system (1997) – PRAC</td>
<td>Provides Weak forward integrity and is impossible for Multi-Hop Mobile Agents in a real network environment.</td>
</tr>
<tr>
<td>Karjoth (1998) - KAG</td>
<td>Do not provide solution for two-colluder attack.</td>
</tr>
<tr>
<td>Roth (1998) - Multi-Agent Based Information Protection</td>
<td>Two agents need authenticated communication.</td>
</tr>
<tr>
<td>Karnik (1999) - encrypted checksum</td>
<td>Does not support two-colluder attacks in the multi-hop Mobile Agent environment.</td>
</tr>
<tr>
<td>Corradi (1999) - protection protocol</td>
<td>Cannot defend the Two-colluder truncation attack.</td>
</tr>
<tr>
<td>Yao (2003) - TTP mechanism</td>
<td>Chances for single-point failure are high. It is not easy to find a TTP in open Internet.</td>
</tr>
<tr>
<td>Zhou (2004) - co-signing mechanism</td>
<td>It cannot defend against multiple-colluder (more than two) truncation attack.</td>
</tr>
<tr>
<td>McDonald (2004) - Multi-Agent Based Information Protection</td>
<td>Not fit for a Multi-Hop Mobile Agent with dynamic itinerary.</td>
</tr>
<tr>
<td>Xu (2006) - one hop backward and two-hop forward chaining method</td>
<td>Too complex to implement in real time.</td>
</tr>
<tr>
<td>Silei (2008a) - trusted computing technology</td>
<td>Additional resource consumption by clones.</td>
</tr>
</tbody>
</table>
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 also. 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.

2.5.1 Pair Processing

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.

2.5.2 Replica injection

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 system (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 system 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.

### 2.5.3 Mobile Shadow

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 and is responsible for executing a task T at a sequence of hosts described by its itinerary. Initially, the master spawns a shadow, shadow\textsubscript{home} at its home agent server before it migrates and executes at the first agent server in its itinerary, i.e. AG\textsubscript{i}. Before the master migrates to the next host in the itinerary, i.e. AG\textsubscript{i+1}, it spawns a clone or shadow\textsubscript{i} and sends a die message to shadow\textsubscript{home}. The shadow\textsubscript{i} repeatedly pings the agent server AG\textsubscript{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\textsubscript{i+1} and spawned a new clone shadow\textsubscript{i+1} to monitor the agent server AG\textsubscript{i+2}. However, assume that the master is lost due to an agent server crash at AG\textsubscript{i+1}. In this case, the shadow\textsubscript{i} at AG\textsubscript{i} detects the crash of its master, spawns a new clone shadow\textsubscript{i} and proceeds to visit the agent server AG\textsubscript{i+2}. Consequently shadow\textsubscript{i} is the new master.

- **Master:** A master pings its shadow at AG\textsubscript{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\textsubscript{i+1}. Before the master migrates, it will send a die message to terminate the shadow at AG\textsubscript{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 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.

2.5.4 Witness Agent and Co-operating Agents

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 systems 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.

2.5.5 K-Response

Venkatesan et al (2010) proposes K-response recovery model, in which the main intention of the attacker is to kill or modify the behavior of the agent during the middle of the journey to degrade the trustworthiness of the agent environment, after some threshold the current host will resend the clone of the agent to the next host. It is mandatory that each and every host should possess a clone to recover the agent before dispatching it and must send the response to the preceding host about the agent status (alive). During the recovery of the Mobile Agent all the components (agent code, itinerary, credential information, collected information and state) can be retrieved.
The drawback of all the above existing systems are summarized in Table 2.4

Table 2.4 Drawback of Existing Agent Recovery

<table>
<thead>
<tr>
<th>Existing Systems</th>
<th>Drawback</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gray and Reuter (1993)-Pair Processing</td>
<td>Primary and Shadow ping each other to determine the alive and raise the communication cost.</td>
</tr>
<tr>
<td>Vogler(1997)- Inject Replica</td>
<td>In the event of server crash, replica remains unavailable.</td>
</tr>
<tr>
<td>Pears (2003)- Mobile Shadow</td>
<td>Time out overhead and Mobile Shadow Overhead.</td>
</tr>
<tr>
<td>Wong (2004)- witness agent</td>
<td>Deals the server failure but not capable of dealing with the recovery of the Multi-Hop Mobile Agent.</td>
</tr>
<tr>
<td>Beheshti (2007)-co-operating agent</td>
<td>Address the single host attack but not capable of dealing with the recovery of the Agent.</td>
</tr>
<tr>
<td>Venkatesan (2010) - K-Response</td>
<td>Maintain the clone to recover the agent in every host.</td>
</tr>
</tbody>
</table>

2.6 OPEN ISSUES IN MOBILE AGENT SECURITY

Figure 2.1 shows the various systems that exist to protect the Mobile Agent environment. There are open issues in these systems. The issues are briefly described below:

During platform protection, 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.
Figure 2.1 Existing Systems for Mobile Agent Environment Protection
In agent code protection, 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 the data protection process, there is no possibility to protect against multiple colluded attackers in adjacent places in a Multi-Hop Mobile Agent. During the recovery of Mobile Agent, there is no possibility to recover the agent from colluded attacks in a Multi-Hop Mobile Agent.

2.7 MOTIVATION

In recent years, a Mobile Agent is used in many applications like Data mining (Klusch et al 2003), Grid computing (Kuang et al 2002), P2P networks (Lu and Fu 2006), Network routing (Manvi and Venkatram 2007), etc. The key reasons for incorporating the Mobile Agent concept in various applications are given below.

- Reduction of Network Load: Mobile Agents helps a lot to reduce the flow of data in the network by packing the conversation and dispatching it to the destination host for the agent to perform computation. The main advantage of the Mobile Agent is that the computation is portable and is performed where the data is available.

- Reduce Network Latency: Critical real-time systems, such as robots in manufacturing processes, need to respond to real-time changes in their environments. Controlling such systems through a factory network of a substantial size, involves significant latencies. For critical real-time systems, such latencies are not acceptable. Mobile Agents offer a solution,
because they can be dispatched from a central controller to act locally and execute the controller’s directions directly.

- **Dynamic Adaption:** Mobile Agents can sense their execution environment and react autonomously to changes.

- **Robust and Fault-tolerant:** If a host is being shut down, all agents executing on that machine are warned and given time to dispatch and continue their operation on another host in the network.

- **Client Customization:** In distributed computing like Remote Procedure Call (RPC) and distributed objects (RMI), the exposed functions are defined and established on the server and there is no opportunity for client customization. Clients are confined to the service provided by the server. In case the clients want to have a new service, the service must be installed on the server. But with the concept of Mobile Agent, the clients are virtually installing programs on to the server when the Mobile Agent migrates from one host to the others.

Despite its many practical benefits, users of the Mobile Agent technology suffer from security threats. To protect against these types of attacks, various solutions were developed, but the developed security systems for a Mobile Agent environment do not give a guarantee to protect it from new types of attacks. Also no recovery mechanism is reported for the Multi-Hop Mobile Agent. The findings reported in this research have been motivated by this fact to develop advanced protection and recovery systems for the Mobile Agent environment.
2.8 OBJECTIVES

The objective is to design advanced Mobile Agent security architecture to protect the Mobile Agent environment from various attacks with a focus on

i) Protection of Mobile Agent Code from Malicious Attacks

ii) Protection of Mobile Agent Platform from Malicious Agents of Malicious Hosts

iii) Protection of entire data carried by the Mobile Agent throughout the travel

iv) Mechanisms to recover the Mobile Agent even after a failure or loss of Agent

2.9 CONTRIBUTIONS

Despite most of the benefits of the Mobile Agent paradigm, the security issue is a paramount problem in its usage. Though many solutions are proposed, each of them has its own drawbacks in terms of a different environment. In this context, a set of new security systems are proposed to protect the Mobile Agent environment from various attacks from multiple malicious hosts.

Security Mechanism 1: The Dual Check Point Analysis (DCPA) has been proposed to safeguard the Mobile Agent platform from malicious attacks. The Authentication Table and the Authentication check performed at the Inner and Outer gate of the system helps in detecting the Malicious Agents which have performed Tailgating attack. The Fragmenter and Defragmenter helps for achieving this task.
Security Mechanism 2: The improved Malicious Identification Police (MIP) along with the Tripmarker proposed in this system helps to identify Malicious Attacks over Mobile Agents. This system in addition to the malicious attack identification provides mechanism to identify and control External Replay Attacks.

Security Mechanism 3: The Address Forward and Data Backward (AFDB) protocol is an effective tool against the Colluded Attacks that takes place in a Mobile Agent System. This sends the Data back to the system which requires it before making its next hop.

Security Mechanism 4: The Trusted Environment with Reference Clone (TERC) proposes two methods of keeping a Reference Clone of a Mobile Agent safe and recovers it when necessary. This mechanism is implemented in a Distributed Intrusion Detection System which helps the system to identify intrusions effectively.

Security Mechanism 5: The eXtended Volunteer Algorithm (XVA) has been formulated to attain a Fault-tolerant environment. A special category of Mobile Agent named Adaptive Mobile Agent is used in this system to attain roles from other Mobile Agents and thereby achieving a fault-tolerance.

The Proposed New security architecture are shown in Table 2.5.
### Table 2.5 Proposed New Security Architecture

<table>
<thead>
<tr>
<th>Security Mechanism</th>
<th>Issues Addressed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dual Check Point Analysis</td>
<td>Tailgating</td>
</tr>
<tr>
<td></td>
<td>Alteration</td>
</tr>
<tr>
<td>MIP with Tripmarker</td>
<td>Copy and Replay Attacks</td>
</tr>
<tr>
<td></td>
<td>Repudiation</td>
</tr>
<tr>
<td></td>
<td>Masquerading</td>
</tr>
<tr>
<td></td>
<td>Alteration</td>
</tr>
<tr>
<td>Address Forward and Data Backward</td>
<td>Alteration</td>
</tr>
<tr>
<td>Trusted Environment with Reference Clone</td>
<td>Unauthorized Access</td>
</tr>
<tr>
<td></td>
<td>Alteration</td>
</tr>
<tr>
<td></td>
<td>Masquerading</td>
</tr>
<tr>
<td></td>
<td>Denial of Service</td>
</tr>
<tr>
<td>Adaptive Mobile Agents</td>
<td>Denial of Service</td>
</tr>
<tr>
<td></td>
<td>Unauthorized Access</td>
</tr>
</tbody>
</table>

#### 2.10 ASSUMPTIONS

This research focuses on the protection of Mobile Agents from different types of attacks using the various protections Security Mechanisms. The protection mechanisms are based on certain assumptions which are as follows:

i) The public key of the hosts are already distributed to the remote hosts connected in the network.

ii) The proposed systems to protect the environment against the attack are the same in all hosts and also it is known by all the hosts of the network.
iii) All illegal activities are reported or intimated to the administrator to take further actions. The actions taken by the administrator is blocking the agent owner and host which perform the indirect platform attack from the network permanently or temporarily until the clearance of the complaint.

iv) The Trusted Environment is assumed to be secure from the access of malicious agents.

v) The Secure channel used is private and cannot be accessed by malicious agents.

2.11 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 technique to protect the Mobile Agent code and data, Mobile Agent server protection and agent fault tolerance systems 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 mechanisms.