CHAPTER 4

MALICIOUS IDENTIFICATION POLICE (MIP)

4.1 INTRODUCTION

From the security perspective, a mobile agent environment consists of three basic components: the originator, remote host and the agent. Here, the agent worries about the tampering by a remote host, and the host worries about the potential damage an agent could do. This chapter concentrates on how to protect the host from the malicious agents. The malicious host can send its malicious agent to gain unauthorized access to a host’s resources or secrets, by exploiting the security holes in the remote host system or masquerading as an authorized agent. Also, the agent can launch denial-of-service attacks by consuming an excessive amount of resources.

To prevent these malicious agent attacks, a number of solutions are given by researchers, like path history (Ordille et al 1995, Ordille1996), proof carrying code (Necula et al 1996), Sandbox (Wahbe et al 1993), etc. Even though the protection mechanism is available, the opportunity for the attacker still exists.

4.1.1 Existing Security Issues in Mobile Agent Platform

The mobile agent from the malicious host can perform multiple types of attacks on the legitimate hosts. Table 4.1 gives the significant types of attacks on the legitimate host (Axel et al 2006), and its related issues.
Table 4.1 Types of Attacks on Mobile Agent Platform

<table>
<thead>
<tr>
<th>Type of Attacks</th>
<th>Issues</th>
</tr>
</thead>
</table>
| Denial of Service (DoS) | • Overloading the agent platform with too many agents  
                          • Overloading the remote agent hosts with too many service requests  
                          • Consuming the computing resources of the remote agent hosts by non-terminating agents  
                          • Consuming the remote agent host’s computing resources by too many messages |
| Unauthorized Access   | • Shutdown the platform  
                          • Modifying policy file  
                          • Killing an agent in the platform  
                          • Replacing the java security manager |
| Agent based attack    | • Spamming the agent with dummy requests  
                          • Suspending the agent  
                          • Sending a signed message with a fake sender ID  
                          • Manipulating the agent’s resources  
                          • Spamming the agent with meaningless information requests. |

In the above list of attacks, the existing mobile agent platforms prevent only some of the attacks. For example, the Secure Mobile Agents (SeMoA 2003) platform prevents the cloning of the mobile agent and killing the agent (it has no option to clone and kill the agent) (Axel et al 2006). But the clone is a must in all the platforms to recover the mobile agent, and the option to kill the agent is a must to discard the agent when it is executing a number of dummy requests. Likewise, the prevention mechanism available in
the platform requires some more additional capabilities to make the environment smart.

Axel et al (2006) pointed out that (after the experimental test) to minimize the risk of DoS attacks by the mobile agent, the platform should have a well-designed security policy. An efficient method of granting permissions in a restrictive manner can help against some DoS attacks. Also, Axel et al (2006) suggest to researchers to develop a model with the police on the platform side to prevent the malicious activities of the agent.

On this basis, the Malicious Identification Police (MIP) like anti-virus is proposed to protect the mobile agent platform with the agent owner privileges. This proposed MIP protects both the direct attack (originator may send the malicious agent) and the indirect attack (the intermediate host may change the behavior of the agent to attack the forthcoming host).

4.2 SIMPLE MALICIOUS IDENTIFICATION POLICE (MIP) MODEL

An agent traversing from one host to another may become malicious through its intelligence or from a malicious originator or from the malicious host on its route. The aim of the malicious agent may be to put the remote host in trouble. To protect against this type of attack, the MIP based Attack Identification Scanner (AIS) model is proposed. It works like the security person having the scanner or detector to scan the people before allowing them into a sensitive place. The scanner used to detect the malicious code, is named as the Attack Identification Scanner (AIS) and the security person is the MIP.

In this model, the byte code of the agent is scanned by the AIS (like an anti-virus) to detect the presence of any malicious code in the given agent
byte code. If the AIS find any malicious codes within the agent code then the agent is either prohibited or killed by the agent platform. If the agent is genuine then it will be permitted to perform its computation. After the completion of the required computation, the agent will be dispatched to the next remote host or to its home as per its itinerary. Generally, the agent from the owner or from the intermediate hosts is the byte code stream (compiled java program – in the case of java). The agent byte code with an additional malicious code is considered as the malicious agent. Here, the AIS will scan the byte code and compare it with the available malicious codes stored in the server on a similar principle of signature based intrusion detection system. A malicious agent migrated from one machine to another machine may have illegal codes like accessing a prohibited database, killing another agent, shutting down the platform, cloning more number of agents, etc. The MIP with the AIS scanner will scan the agent byte code to detect malicious instructions.

Figure 4.1 shows the mobile agent roaming in the network and it is scanned by the platform to prevent any attacks. A mobile agent which starts from the home consists of the state, code, credential information (Id and encrypted Id), dummy data and itinerary (it is not shown in Figure 4.1) and it will visit the remote hosts to perform some computation. After the agent reaches the remote host, the MIP should follow the following MIP algorithm to prevent the platform attack.

Step 1: [Decryption] The MIP decrypts the encrypted identity of the agent owner with the public key of the agent owner.

Step 2: [Verification] The MIP verified the similarity of the identity to avoid masquerading.

Step 3: [AIS Initialization] The MIP initiates the AIS to scan the agent byte code.
Step 4: [Scan] The AIS will scan the agent byte code and report the presence of malicious code if any, to the MIP.

Step 5: [Decision] The MIP will take the decision at to whether to discard the agent in the case of the availability of a malicious code or allow the agent to execute in the case of a legitimate host.

Figure 4.1 Mobile Agent Roaming with the Simple MIP based AIS Scanning
Every succeeding host in the mobile agent environment with the MIP model will process the above algorithm to protect the platform from the malicious agents.

### 4.2.1 Malicious Code Scanning Model

Agent platforms have a database that contains a list of malicious codes to detect the agent with malicious activity. The AIS will scan the agent code by searching for the string (malicious code) in the set of byte codes. To search for the malicious code in the agent byte code, the no-best algorithm is used even though the best and quick search Boyer-Moore algorithm (Boyer and Moore 1977) is available, because it has the process of preprocessing the string in the set of strings (file). The preprocessing is the arrangement of the list in order to speed up the searching in the same set of data. But in the mobile agent environment, the search is done only once for the byte code. So, the ordinary linear search algorithm is used to scan the agent to detect the malicious codes.

The searching model of AIS is shown in Figure 4.2 to search the two strings from the set of strings. It shows the searching for two strings (exit, clone) from the set of strings (exi exit ass clone). The first character of the search string is compared with the first character of the available string (e=e, then x=x and i=i) and consecutively. After that, if the fourth character of the two is not same (≠ e) then start from the first character ‘e’ and the second character ‘x’ of the available string. Next, the second substring is matched with the search string (e=e, x=x, i=i, t=t). Then it continues to identify the second search string clone from the beginning letter ‘e’ but Figure 4.2 shows the search starting from the letter ‘a’ for illustration purposes. The scanner (searching) continues to scan the agent until the searching malicious code is found or the agent code reaches its end. Today’s Java has the code
verification mechanism to identify the object initialization, stack overflow and underflow (Xavier 2003) but it will not prevent malicious access, exiting the platform, denial of service, etc. The MIP model will prevent all these attacks. It works like an anti-virus for system protection. Some of the malicious codes available with the agent are clone (cloning more number of agents to perform DoS attack), exit (shutdown the platform), accessing the prohibited database, killing the other agent which runs in the current platform, etc.

\[
\text{Stored Signature} = \{\text{exit, clone}\}, \ \text{sample part of agent code}= \{\text{exi exit ass clone}\}
\]

**Figure 4.2 AIS Scanning Model**

### 4.2.2 Complexity Measures of AIS

The scanning time of the AIS is based on the size of the agent code and the availability of malicious codes, if any. The initial mobile agent does not perform any task; it does not generate the result for that task. The \(A_{code}\) or \(AC\) is the agent byte code and the \(A_{state}\) is the state of the agent. The \(T(A_{code}, M_{code})\) is the time required to find the malicious code \(MC\) or \(M_{code}\) in the agent code \(A_{code}\).

\[
T(A_{code}, M_{code}) = R + Z
\]  
(4.1)
Let variable \( p \) be the time taken to match the character of the malicious code in the stored database and agent code. Let \( R \) and \( Z \) be initially zero to calculate the total time to scan for the availability of the malicious code in the agent code. The value of \( R \) and \( Z \) is calculated based on the below algorithm.

**Procedure AIS()**

```plaintext
var tv=0; //to exit the algorithm when attack identified
For i=0 to len (AC)-1 do
  if MC_0 = AC_i then
    Z=Z+p;
    For j=1 to len (MC)-1 do
      if MC_j = AC_i+j then
        tv=tv+1;
        R=R+p;
      next j;
    else
      if tv=len(MC) then
        exit AIS()
      else
        tv=0;
        QuitLoop j;
    end if
  end if
next i;
end if
end procedure
```

The complexity of the scanning model to identify the single malicious code from the agent codes is based on the size of the agent code \( (v) \) and the size of the malicious code \( (u) \). The three cases of the complexities are:

- **Best Case** – \( O(u) \)
- **Average Case** – \( O\left(\frac{(u-1) \cdot v}{2}\right)\)
Worst Case – $O((u-1)^v)$  \hspace{1cm} (4.4)

4.2.3 Experimental Results

4.2.3.1 Experimental Setup

The MIP based Attack Identification Scanner model is experimented with three types of illegal codes using the IBM Aglet agent server. It is implemented in the machine with the configuration of 2.2 GHZ dual core processor and 1 GB RAM. The three types of illegal codes are to:

(i) Shutdown the platform

(ii) Update the administration database

(iii) Clone more number of agents to overload the server.

The remote host which received the agent will initiate the Attack Identification Scanner (AIS) to scan the byte code of the mobile agent to detect and prevent the illegal code.

4.2.3.2 Obtained Results of Simple MIP Model

Table 4.2 shows the execution result of the MIP model to scan different sized agent codes with and without the above malicious codes.

Table 4.2 Simple MIP Scanning Time for Different Size of Agent Codes with and without Malicious Code (in milli seconds)

<table>
<thead>
<tr>
<th>Size of Agent Code</th>
<th>Scanning Time without Malicious Code</th>
<th>Scanning Time with Malicious Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>3KB</td>
<td>93</td>
<td>109</td>
</tr>
<tr>
<td>6KB</td>
<td>109</td>
<td>125</td>
</tr>
<tr>
<td>12KB</td>
<td>156</td>
<td>172</td>
</tr>
<tr>
<td>15KB</td>
<td>172</td>
<td>203</td>
</tr>
</tbody>
</table>
Figure 4.3 shows the time complexity performance of the MIP scanner to scan the incoming agent code with (exit, clone and update malicious codes) and without malicious codes for different sizes of the agent. The result shows that the scanning time will increase in the case of the malicious agent; otherwise, it will be less. In some cases, the agent code may consist of a partial matching malicious code; it will increase the MIP scanning time but no attack is detected. In this scenario, the partial matching code is stored by the MIP to identify the new type of malicious codes. Here, the scanning time is not proportional to the file size, because apart from the scanning time (searching time), the agent code retrieval time and agent identity verification time is also included with the agent byte code scanning time shown in Tables 4.2 and 4.4.

**Figure 4.3** Computational Performance of Simple MIP to Scan the Agent Codes with and without the Attack Code

**4.2.4 Rearranging the Frequently Detected Malicious Codes**

Every platform contains a list of malicious codes to check with the byte code of the received agent. Platforms will not search the agent code for
the availability of all the malicious codes in the database. If any one of the malicious code is available in the agent, it will suddenly report to the MIP to take necessary action. When searching the malicious codes of the agents, some of the codes may be frequently available in most of the agent codes. Based on the frequently available malicious codes in the agent, the malicious code in the platform should be rearranged to increase the scanning efficiency. Instead of taking the stored attack code from the database, the frequently detected attack code is stored with MIP and accessed. The MIP stored attack code is rearranged for efficient scanning.

For example, a platform contains the malicious codes in the order of update, clone, and exit. While searching the agent, it will look for the update; then it will look for the clone; and then it will look for the exit. If any one of the malicious code is found in the agent, it will be suddenly blocked by the MIP.

Consider the MIP scans of 100 malicious agents, with 60 malicious agents having the malicious code exit, three having the malicious code update and one having the malicious code clone. From these, the MIP will automatically rearrange the malicious codes based on the frequently available order. The rearrangement of the malicious code in the platform is represented as in below.

\{update, clone, exit\} – Before Rearrangement

\{exit, update, clone\} – After Rearrangement

4.3 POLICY BASED MIP MODEL

4.3.1 Need for Policy based MIP Model

In the above AIS model, the agent from the remote host is scanned by the AIS scanner and the entire remote host agent in the network treated equally (remote host may be the administrator, requisition client, spectator,
attacker, etc.). To avoid this, the policy based scanning model is required for scanning the agent. If an agent migrates from a host to the remote host, it should have identification (I_{A}) and the encrypted (or) signed form of that Identification (E_{I_{A}}). The manager agent (MIP) in the remote machine will verify the identity of the agent with the plain and encrypted form of identity and initiate the AIS to scan, based on the privileges given to the agent owner.

### 4.3.2 Functions of Policy Based MIP Model

Figure 4.4 shows the migration of the mobile agent among the hosts with the policy based MIP model. The host h_{0} dispatches an agent that contains the code, state, information, and credential information (I_{A} and E_{I_{A}}) to the remote host h_{1}. While receiving the agent, the MIP of the host h_{1} will verify the identity of the agent as the simple MIP model shown in section 4.2. After authentication, the MIP will find the equivalent policy in the policy file and provide it to the AIS to scan the agent based on the policy. For this, every host should have the policy file for access control, which is static or dynamic. For the dynamic policy, the agent should provide a valuable reason (for example, the spectator agent can come on behalf of the administrator when the administrator agent is busy or in trouble) to the platform.
If the AIS detect any violation by the agent apart from the access list, then it will prevent the agent from executing, or agent is killed or the violation is reported to the administrator through MIP for a further process. If the AIS find the agent to be genuine after scanning, it will allow the agent to execute to get the required data and dispatch it to the next host $h_2$. This process will continue until the agent reaches its home.

In a distributed environment, the administrator has all the rights in the machines linked with the network. The agent from the spectator has the particular rights like updating the database, cloning the agent or deleting the agent. The client has no rights to make any change in the remote machines. If the MIP identifies whether the agent is from the administrator, the AIS is not initialized to scan the agent. The reason is that, the administrator has all the rights to the hosts connected to the network. Table 4.3 shows the privileges for the agents from the different originators. The three risk codes in Table 4.3 are the sensitive malicious codes considered in this thesis; apart from these malicious codes, a number of malicious codes are available as given by Axel et al (2006) and are to be incorporated in the access list for efficient scanning.

Table 4.3 Privileges to Agent from Different Originators
4.3.3 Experimental Results

4.3.3.1 Experimental Setup

The policy-based MIP is experimented with the same three types of malicious codes as in the simple MIP scanning model, using the IBM Aglet agent server (2004). It is implemented in the same machine with the configuration of 2.2 GHZ dual core processor and 1 GB RAM.

The time taken ($T_t$) to scan the agent code as per the policy-based MIP is given in equation (4.5). It includes the verification ($T_v$) and the scanning time ($T_s$) of the agent code (it is based on the agent code size). The identity verification time ($T_v$) includes the decryption time of the identity ($T_D$), mapping time and the policy verification time ($T_{PV}$) as in equation (4.6).

\[
T_t = T_v + T_s \quad (4.5)
\]

\[
T_v = T_D + T_{PV} \quad (4.6)
\]

4.3.3.2 Obtained Results of Policy based MIP Model

<table>
<thead>
<tr>
<th>Risk Codes</th>
<th>Administrator</th>
<th>Spectator</th>
<th>Client</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Grant</td>
<td>Block</td>
<td>Grant</td>
</tr>
<tr>
<td>Clone</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>Shutdown</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>Update</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
</tr>
</tbody>
</table>
Table 4.4 shows the execution result of the MIP model to scan different size of agent codes with and without malicious codes. Obviously, the scanning time of the policy-based MIP model is more than the simple MIP scanning model because of its identity verification time and additional policy verification time given in equation (4.5) and (4.6).

Figure 4.5 shows the time complexity performance of the MIP scanner to scan the incoming agent code with (exit, clone and update malicious codes) and without malicious codes for different sizes of agents.

Table 4.4  Policy-based MIP Scanning Time for Different Size of Agent Codes with and without Malicious Code (in milli seconds)

<table>
<thead>
<tr>
<th>Size of Agent Code</th>
<th>$T_i$ without malicious code</th>
<th>$T_i$ with malicious code</th>
</tr>
</thead>
<tbody>
<tr>
<td>3KB</td>
<td>203</td>
<td>235</td>
</tr>
<tr>
<td>6KB</td>
<td>265</td>
<td>297</td>
</tr>
<tr>
<td>12KB</td>
<td>344</td>
<td>359</td>
</tr>
<tr>
<td>15KB</td>
<td>375</td>
<td>390</td>
</tr>
</tbody>
</table>
Figure 4.5  Computational Performance of the Policy-based MIP to Scan the Agent Codes with and without the Attack Code

4.4  COMPARISON WITH THE EXISTING PLATFORMS

Table 4.5 shows the comparison of the MIP model with the existing platforms. The limitations of the existing models SeMoA (2003) and JADE (2006) are pointed out by Axel et al (2006) and the limitations of the IBM Aglet (2004) are tested for multiple number of executions with different attack codes in different agent code sizes in the experimental setup of Section 4.2.3.1. In the JADE platform, denial of service (cloning the multiple number of agents), unauthorized access (modifying the yellow pages) and shutting down of the platform (exit) or killing the agent (dispose) are possible.

Table 4.5  Comparison of the Existing Agent Platforms and Proposed MIP Model

<table>
<thead>
<tr>
<th>Attacks</th>
<th>JADE</th>
<th>SeMoA</th>
<th>Aglet</th>
<th>Proposed MIP model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Denial of Service</td>
<td>Possible</td>
<td>Preventable</td>
<td>Possible</td>
<td>Preventable</td>
</tr>
<tr>
<td>Unauthorized Access</td>
<td>Possible</td>
<td>Possible</td>
<td>Possible</td>
<td>Preventable</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>----------</td>
<td>----------</td>
<td>----------</td>
<td>-------------</td>
</tr>
<tr>
<td>Shutdown the platform or killing the agent</td>
<td>Possible</td>
<td>Preventable</td>
<td>Possible</td>
<td>Preventable</td>
</tr>
</tbody>
</table>

In SeMoA, denial of service is not possible because no cloning is allowed. Even though it avoids the denial of service, the clone function is important to recover the mobile agent after failure in the remote hosts. Then unauthorized access (replacing the policy filter) is possible but shutting down the platform and killing the agent is not possible. In Aglet, it is possible to have the denial of service attack by cloning the agent, unauthorized access and shutting down the platform (System.exit). But with the proposed MIP model, it is prevented by scanning the incoming agent byte code.

4.5 IDENTIFYING THE NEW ATTACK CODES

In some cases, the agent may arrive at the remote hosts with a new type of attack codes which are not already available with the remote hosts. The new attack codes may be partial matching codes or it is completely unknown attack codes.

4.5.1 Partial Matching Attack Code

In this scenario, the AIS may find the partial matching with the stored attack code, and report it to the MIP. The MIP will store and keep the partial matching code (50% and above matching code only) in the separate partial matching database until the respective agent completes its execution and migrates from the host. If any unknown attack is generated on the host at the time of the respective agent computation, then the partial matching code in the partial matching database is moved to the attack code database.
Sometimes, it is possible to kill the MIP but the partial matching attack code always remains in the database (backup is available in the case of database attack). The MIP initialized after death will first move the code from the partial matching database to the attack code database and then it will continue its monitoring function. To identify the partial matching code, no additional time is required other than the scanning time, because at the time of scanning itself it can separate the partial matching code.

### 4.5.2 Completely Unknown Attack Code

In this scenario, the AIS could not find any matching or partial matching codes but the attack will happen and also it is not identifiable. To protect the host from this type of attacks, additional functions and intelligences are required by the AIS such as (i) Scanning, (ii) Lexical analyze and (iii) Detecting unknown functional codes of the incoming agent.

i) Scanning: The Intelligent AIS (IAIS) will perform the scanning function and report the presence of the malicious code to the MIP. If no malicious code is found then it will go for a lexical analysis by its own decision. This is achieved through its intelligence.

ii) Lexical Analyze: The IAIS will perform a process to convert the agent byte codes into tokens. The tokens, started with alphabetical characters only, are taken for attack code detection because the attack codes of the agent platform start with alphabets (it is identified from the experiment of the simple and policy-based MIP model).

iii) Detecting Unknown codes: The mobile agent platform already consists of the agent’s known functional codes. The IAIS will match the identified token with the functional codes of the
host to identify the non-match token of the agent byte code.

Finally, the IAIS reports the unknown codes if any to the MIP.

After the report is received by the MIP, it will store the unknown codes in the database until the agent completes its process and is dispatched from the host. If any malicious operation was there in the environment within the time period, than the MIP can decide that the unknown code is the attack code and it will update the attack code database with the new identified attack code. This proposed IAIS based MIP model has two good advantages than the existing Anti-Virus. They are:

i) The host itself detects the new attack code

ii) The host itself can update the attack code database with the newly identified attack codes.

4.5.2.1 Analysis of Unknown Attack Code Identification

In the earlier stage, the host should have all the functional codes of the agent in the tree structure. Figure 4.6 shows the sample tree structure storage of the agent’s functional codes such as clone, dispatch, create, etc.
The agent from the remote hosts is the byte code. The sample byte code of the incoming agent ($A_1$) to the host is shown in Figure 4.7 and the sample tokens are shown in the figure itself by darkening the strings other than the general functions such as java/io, venkat/de1- specifies agent name, println, etc.

Now the IAIS maps the tokens of the agent code with the agent’s functional code stored in the tree structure storage to detect the unknown codes. The unknown code after the mapping function is “dispose”. The IAIS will report the availability of the unknown codes to the MIP. The MIP will store the code into the unknown code database until the agent completes its computation. At the time of the agent ($A_1$) running in the host, the other agent ($A_2$) was killed. Hence, the MIP will decide that the unknown code “dispose” is the attack code and it will move the “dispose” from the unknown code
database to the attack code database. The killed agent was recovered using the
K-response recovery model of the security architecture in chapter 7.

The additional computation time of the IAIS includes the lexical
analyze time \( (\lambda) \) and the identification of the unknown codes \( (\rho) \). The total
time taken by the IAIS to identify the new attack code \( (T_{IAIS}) \) including the
scanning time \( (T_s) \) is

\[
T_{IAIS} = T_s + \lambda + \rho
\]  
(4.7)

4.5.2.2 Experimental Results of Unknown Code Identification

The MIP with the IAIS model to identify the unknown codes is
experimented in the same set up of the simple and policy-based MIP model
with a simulated agent. Table 4.6 shows the detection of the unknown codes
from the different agent code size. It includes the total number of tokens
extracted from the agent byte code, number of tokens matching with the
available functional codes in the host, the number of basic functional codes
(such as java, aglet, connect, string, langstring, printstream, etc.) and the final
number of unknown codes in the incoming agent. The unknown codes of the
agent are stored in the temporary database to identify whether the code is
malicious or legitimate.

Table 4.6 Identification of the Unknown Codes using the IAIS based
MIP Model

<table>
<thead>
<tr>
<th>Size of Agent Code</th>
<th>Number of Tokens</th>
<th>Number of Matching Token</th>
<th>Number of Basic Functional Codes</th>
<th>Number of Unknown Codes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1KB</td>
<td>72</td>
<td>0</td>
<td>71</td>
<td>1</td>
</tr>
<tr>
<td>3KB</td>
<td>140</td>
<td>3</td>
<td>137</td>
<td>0</td>
</tr>
</tbody>
</table>
Table 4.7 shows the execution time of the unknown code identification model based on the equation (4.7). The difference in time among the agent codes is based on the number of tokens and the number of matching functional codes. For example, the time to identify the unknown codes of the 3KB agent size is 141 and the 6KB agent size is 78. The reason for this is that, the number of matching tokens in Table 4.6 for the 3KB agent code is higher than those of the 6KB agent code.

Table 4.7  Processing Time of the IAIS-based MIP Model to Identify the Completely Unknown Codes (in milli seconds)

<table>
<thead>
<tr>
<th>Size of Agent Code</th>
<th>Scanning Time ($T_s$)</th>
<th>Time to Generate Tokens ($\rho$)</th>
<th>Time to Identify Unknown Codes ($\lambda$)</th>
<th>Total Time ($T_{IAIS}$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1KB</td>
<td>150</td>
<td>94</td>
<td>94</td>
<td>338</td>
</tr>
<tr>
<td>3KB</td>
<td>203</td>
<td>172</td>
<td>141</td>
<td>516</td>
</tr>
<tr>
<td>6KB</td>
<td>265</td>
<td>297</td>
<td>78</td>
<td>640</td>
</tr>
<tr>
<td>12KB</td>
<td>344</td>
<td>281</td>
<td>109</td>
<td>734</td>
</tr>
<tr>
<td>15KB</td>
<td>375</td>
<td>297</td>
<td>109</td>
<td>781</td>
</tr>
</tbody>
</table>

4.5.2.3  Attack Code Detection Strategy

Agent platform will always have multiple agents at a time for different executions. According to IAIS scanning, all the unknown codes of multiple agents are stored in the database until the agents are dispatched from the host. If any attack happens, there is no opportunity to identify
which unknown code is the attack code. To identify the unknown attack code from the set of codes, a testing is required. Testing is done by creating a new agent with the unknown codes available with the platforms. The efficient strategy is required for testing the unknown codes for malicious. It can be done through the binary group testing. For example, if we have 100 unknown codes then we have to group it into two on 50-50 basic and make a test on both groups of codes. If we find attack code on any one of the groups, then fragment that particular group into 25-25 and then to 12-13 until we identify the attack codes. Sometimes, it is possible that both the groups may have the malicious codes. It refers that the multiple attack codes are available in the set of unknown codes. In that case, both the group should be fragmented until the attack codes are to be identified.

4.6 SUMMARY

This chapter proposes a model for preventing mobile agent malicious attacks against legitimate platforms. The proposed simple MIP with the AIS scanning is replaced with the policy-based AIS scanning, to improve the efficiency of the scanning method and to differentiate the agent owners. The AIS scans the mobile agent code using the linear search algorithm without any preprocessing and submits the report to the MIP. In addition to this, the attack codes are rearranged, based on frequently identified attack codes to minimize the scanning time. Hence, no malicious agents are allowed to perform the computation in the remote machines. Also, the comparison of the proposed model with the existing models illustrates that it can prevent attacks which are not prevented by the existing models. The most important feature of this model is that it is able to identify and protect the host from partial matching and completely unknown attack codes.