Securing Mobile-Agent-Based Systems Against Malicious Hosts

Mohamed A. Madkour, Fathy E. Eassa, Abdullah M. Ali and Noor U. Qayyum

FCIT, King Abdulaziz University, Jeddah, KSA

Abstract: Mobile agent technology is an attractive alternative to traditional network computing models. It is suitable to many different types of distributed systems’ applications. The main obstacle for the widespread deployment of the mobile agent technology is the security issue, in particular the problem of protecting a mobile agent from malicious hosts that may completely block the agent or modify its accumulated data. The present work aims to introduce a solution for this problem based on identifying and skipping malicious blocking hosts within the mobile agent’s itinerary. The solution is extended without using expensive cryptographic techniques to ensure the integrity of the accumulated data when the agent eventually returns home. The proposed solution is simulated to examine feasibility, correctness and scalability of the developed protocol. A prototype is built and tested to investigate practical aspects of the solution.

Key words: Mobile agents · Malicious host problem · Blocking platforms · Distributed systems safety · Security.

INTRODUCTION

Mobile agent technology introduces the concept of code mobility that allows efficient use of valuable network bandwidth in distributed environments. In its simplest form, a relatively small piece of software migrates to one or more remote heterogeneous host systems, executes on local data at the host system and eventually returns back with the obtained results to the home host. This scenario is typical in many distributed applications where data from different host systems may be needed to perform certain tasks. In fact, moving a mobile code between hosts to perform a certain task consumes very little network bandwidth compared to the traditional paradigm which involves moving bulky chunks of data between hosts to perform the same task.

The main obstacle for the widespread deployment of mobile agent technologies is the security issue. A classical NIST report [6] about mobile agent security identifies four threat categories: threats stemming from an agent attacking an agent platform, an agent platform attacking an agent, an agent attacking another agent on the agent platform and other entities attacking the agent system.

For instance, in the first threat category the “Agent-to-Platform” problem focuses on the protection of host systems from malicious mobile agents. Sharma et al [18] provide a good survey and comparison between alternative techniques to counter this problem. These include the code signing technique, sandboxing and proof coding. Microsoft Corporation [11] provides a good introduction to code signing. Sandboxing is discussed by Gong [4]. A good review of code signing and sandboxing is given in Alfalayleh and Brankovic [2] and Ahmadi-Brooghani [1]. Loureiro et al [9] and Necula and Lee [12] are good references for the proof coding technique.

On the other hand, a more severe and challenging issue is the “Platform-to-Agent” problem identified in the second threat category. This problem considers the protection of mobile agents from malicious host systems. A malicious agent platform may ignore agent service requests, introduce unacceptable delays for critical tasks, simply not execute the agent's code, or even terminate the agent without notification [6].

In essence, the mobile agent comprises three components, namely: code, data and execution state. As a mobile agent moves from host to host accumulating partial results, it is vulnerable to malicious manipulation
by a remote host. Zachary [21] classifies the vulnerabilities in four broad areas: inspection, modification, replay and denial of service. These vulnerabilities exist because enforcement of security policy at remote hosts cannot be guaranteed. Recently, Marikkannu et al [10] provide a list of various types of problems and attacks in the mobile agent paradigm.

The present work focuses on suggesting a solution for the denial of service attack by a malicious host that blocks a visiting mobile agent and prevents it from continuing its itinerary. The rest of this paper is structured as follows: Section 2 contains a brief survey of related work. Section 3 describes a novel protocol for identifying and skipping malicious hosts in the mobile agent’s itinerary. Section 4 provides simulation results to examine the protocol feasibility, correctness and scalability. In Section 5, a prototype is developed to investigate practical aspects of the suggested solution and evaluate its performance. Conclusions are given in the last section.

Related Work: When a mobile agent executes at a hosting system, its code and data are under full control of the executing host. The host may reverse-engineer the mobile code to fully understand its operation, giving the host an unfair advantage. The malicious host may also alter the agent’s data in its favor and it may capture or destroy the visiting mobile agent. The “Platform to Agent” problem has not yet received an acceptable level of security guarantees and appears to remain a big challenge. The attacks identified above raise deep concerns about the widespread deployment of applications based on the mobile agent paradigm.

An excellent survey on the technical mechanisms that provide countermeasures for mobile agent security is given by Jansen [7]. Ssekibuule [19] gives a recent literature review of the most prominent security mechanisms for detection of security breaches against mobile agents and surveys various techniques for mobile agent protection against malicious hosts. Still Sharma [18] surveys many security techniques and mechanisms to deal with the same problem, pointing to their strength and limitations.

Based on cryptography, Gong [4] presented one of the earliest solutions to achieve integrity and privacy of the mobile agent. It protects the mobile agent against tampering by a malicious host and allows it to conceal its code when it is executed remotely in an untrusted environment. However, during mobile agent execution, parts of the code will be decrypted with a secret key and the code appears in the clear. A malicious host can directly intercept, debug and decompile the code. Another limitation is that mobile cryptography is too expensive to implement. A much less expensive solution to code confidentiality is to employ code obfuscation techniques [8]. Obfuscation is a protection technique for making code unintelligible to automated program comprehension and analysis tools. Armoogum and Caully [3] give an implementation of an agent obfuscator to provide code confidentiality. They present three different obfuscation techniques on java mobile agent code for security. Every time a code is obfuscated the resulting obfuscated code is different and appears scrambled. This is a reasonable alternative to encryption of mobile code that increases the code complexity rendering it unreadable and difficult to reverse-engineer. Another approach is based on computing with encrypted functions where a mobile agent can execute encrypted tasks while maintaining integrity and confidentiality. Sander [16] proposed a technique which allows for non-interactive addition or multiplication of two encrypted messages by manipulating cipher-text only. This method has a limitation as it requires the functions of the mobile code to be represented as polynomial. This technique cannot yet provide such schemes for general functions. Shao-Yin [17] extended this technique to provide a complete solution to mobile agents in order to keep the execution integrity, code privacy and data privacy in circuit level. Yet his technique is not applicable at a higher level and needs more research.

An interesting approach to the denial of service attack by a malicious host that blocks a visiting mobile agent is presented in the POM model [5]. In this model the mobile agent is divided into a security-critical master part and a security-free slave part. Besides, the entire distributed system space is divided into non-overlapping regions. The master part visits a safe host (called police office; PO) in a region consisting of a group of hosts and stays there. For every host in that region, the slave part will visit the host to do some processing such as data collection and then return back to the master. Upon finishing all hosts in a region, the mobile agent will move to the safe host of the next region and so on. In this way, should a malicious host capture the slave part; the master will launch another instance of that slave to continue visiting other hosts in the region. A malicious host will
not benefit from examining the security-free code of the slave part. The two main obstacles in this approach are region partitioning and mobile agent code division. Other drawbacks of this approach include the PO bottleneck and the add-on difficulties in agent programming.

A solution of a quite similar problem is given by Pears et al. [14] using a fault tolerance approach to maintain mobile agent availability in the presence of agent server crashes. They presented a dynamic mobile shadow scheme employing a pair of replica mobile agents, master and shadow, to survive remote agent server crashes. The shadow mobile agent operates at the agent server visited prior to its master’s current location. If a master crashes its replica is available as a replacement. A special mechanism is used to control the mobility of the master and shadow agents such that there is always a shadow mobile agent lagging in the same itinerary by one agent server behind the master. The shadow pings its master’s agent server. If a shadow detects a master crash it raises a local exception to signify the master failure. The exception handler skips the master’s current location and sends the shadow to the next agent server.

The Proposed Protocol: The present work considers the problem of capturing mobile agents inside a malicious host (platform). The developed mechanism aims to identify and skip every blocking malicious host in the itinerary of a mobile agent. This solution is comparable to the POM model [5] and the dynamic mobile shadow scheme [14]. It improves the POM approach by avoiding its two main obstacles mentioned in Section 2. Firstly, the developed mechanism eliminates the need to divide the mobile agent into a security-critical master part and a security-free slave part. Secondly, it avoids dividing the entire distributed system space into non-overlapping regions with a trusted Police Office server in every region. On the other hand, compared with the mobile shadow scheme, the present work employs the idea of a shadow agent but with a different viewpoint taking into account that one or more agent servers in the itinerary can be malicious. The mechanism of repeatedly sending ping messages to the next agent server to check its availability, as used in [14], is quite good when all agent servers are assumed benign. However this evidently does not guarantee that the master mobile agent is unblocked when an agent server is suspected to be malicious.

The problem of this work and its proposed solution are summarized as follows. Consider a home host \( H_0 \) that will launch a mobile agent to sequentially visit a set of hosts (agent servers) \( \mathcal{H} = \{ H_1, H_2, \ldots, H_n \} \) and collect partial data at each visited host. The mobile agent will accumulate the data and will eventually return back to the home host. There is a threat that one or more hosts in the specified itinerary are malicious and will block the mobile agent. The developed scheme relies on an acknowledgement and time-out mechanism to ensure that a mobile agent has visited a host in its itinerary and safely departed to the next one. Here there are two mobile agents a primary (PA) and a shadow (SA). Normally SA is lagging one step in the itinerary behind PA, that is when SA is waiting in an arbitrary host along the itinerary \( H_i \), \( i=1 \ldots N-1 \) PA will be executing in the next host \( H_{i+1} \). When PA completes its task in \( H_{i+1} \) it will move to the next one \( H_{i+2} \) and from the newly visited host it will send an acknowledgement to SA that is now lagging by two steps. Upon receiving the acknowledgement SA would move to the next host in its trip \( H_{i+1} \) and become again lagging by one step behind PA. The assumption here is that a host \( H_{i+1} \) is considered non-blocking should it allow the primary agent to continue its task and safely depart to the next host \( H_{i+2} \). Alternatively, SA will assume that one of the next two hosts \( H_{i+1} \) or \( H_{i+2} \) is malicious should it did not receive the acknowledgement, from PA at \( H_{i+2} \) within a specified time-out period. In this case SA will request help from the home host \( H_0 \) to take the necessary corrective actions for identifying and skipping the malicious host.

Subsection 3.1: Describes the above-mentioned solution and shows in details the proposed mechanism to identify and skip blocking hosts. This mechanism protects the data accumulated in SA from previous visits to other non-blocking hosts. Upon identifying a malicious host in the itinerary, the home host will launch a new instance from the primary agent \( PA_{new} \) to get the accumulated data from the shadow agent. \( PA_{new} \) will migrate directly to meet SA that already resides in the last visited non-blocking host. After loading all accumulated data from SA, \( PA_{new} \) would skip the identified malicious host and continue its trip in the remaining part of the itinerary.

An important threat must be considered here regarding the integrity of the accumulated data and the mobile code, regarding both PA and SA. Some non-blocking hosts can be malicious and unsafe in a different manner. Firstly it may deliberately change the accumulated data, in its favor, in both PA and SA, to gain undeserved benefit. Secondly it may reverse engineer the mobile code and insert changes in the code of PA.
Fig. 1: The itinerary of the mobile agent.

Fig. 2: Acknowledgment and timeout mechanism in safe conditions.

and/or SA. Such changes can be either to gain undeserved benefit or in the worst case it can turn both agents to be malicious. Subsection 3.2 provides a solution to detect such attacks.

**Normal Protocol Sequence:** The following describes the sequence of events when there are no malicious hosts in the itinerary. Consider an arbitrary host $H_{i-2}$ ($i=2, 3 \ldots N$) that is proven to be safe as will be shown next. SA will wait at $H_{i-2}$ and PA will continue to move to the next two hosts in sequence, namely $H_{i-1}$ and $H_i$. An acknowledgement and time out mechanism will be used to detect whether $H_{i-1}$ is malicious or not. Host $H_{i-1}$ will be considered safe if and only if PA arrives at the next host $H_i$. If $H_{i-1}$ is found to be safe, then the responsibility of the collected data will be transferred to it and the shadow agent SA will safely move to $H_{i-1}$.

For $i=2$, $H_2$ is the home host which is obviously safe. SA will move to $H_i$ only when PA sends an acknowledgement from $H_i$ indicating that $H_i$ is safe. As long as all visited hosts are safe this process will continue, where PA will be always ahead of SA by two hosts until eventually it will return back to the home host. Now there is a moving safety window consisting of three hosts as shown in Figure 1 in which host $H_{i-1}$ is proven safe. Whenever $H_i$ is found to be safe, this window
advances one step forward. At the beginning the safety window is \(\{H_0, H_1, H_2\}\) and SA will stay at \(H_0\). If \(H_1\) is found safe the window will be \(\{H_1, H_2, H_3\}\) and the shadow agent SA will move to \(H_2\). As long as PA visits safe hosts, the window at any intermediate point within the itinerary will be \(\{H_{i-1}, H_i, H_{i+1}\}\).

Figure 2 shows the sequence of events that implement the stated solution. These events are summarized below where \(H_{i-2}\) is proven to be a non-blocking safe host and the next three hosts in the itinerary \(\{H_{i-1}, H_i, H_{i+1}\}\) are members of a security checking window. The index “i” takes the values 1, 2, 3... N and initially for \(i=2\), \(H_2\) is the home host. Note that \(H_{i-2}\) is obviously the home host \(H_0\) and for the first move (\(i=1\)) steps 3 and 4 below are not applicable. When \(H_{i-2}\) is proven to be non-blocking, the security window moves forward in the itinerary to check the next host and so on until the primary agent returns back to its home host.

- Initially \((i=1, i=2)\), SA is empty and it will wait at home host \(H_0\) until \(H_1\) is proven safe. For \(i>2\), the loaded SA waits in the safe host \(H_{i-2}\) until it receives a proof that \(H_i\) is safe.
- PA completes its processing at \(H_i\) and moves to the next host \(H_{i+1}\).
- When PA arrives at host \(H_i\), it sends an authenticated acknowledgement \(\text{ack}_i\) back to host \(H_{i-2}\). The authentication is done by the private key of host \(H_i\) to establish a proof that the mobile agent PA was not blocked by host \(H_{i-2}\).
- SA moves to \(H_{i+1}\) and repeats the data collection process. It will wait there, carrying all data collected during its travel until it receives a proof that \(H_i\) is safe.
- When PA completes its work at host \(H_i\), it will depart to visit the next host \(H_{i+1}\).
- When PA arrives at host \(H_{i+1}\) it sends an authenticated acknowledgement \(\text{ack}_{i+1}\) back to host \(H_{i-2}\). The authentication is done by the private key of host \(H_{i-1}\) to establish a proof that the mobile agent PA was not blocked by \(H_{i-2}\).
- When SA at host \(H_{i+1}\) receives \(\text{ack}_{i+1}\), it will be sure that \(H_i\) is safe and it will move to host \(H_i\). SA should repeat the same processing at \(H_i\) so that it always carries an independent backup copy of the accumulated data already in the primary agent PA.
- Upon suspecting a malicious action, SA will request help from the home host \(H_0\).
- \(H_0\) will react by launching two agents as follows:
  - A final complementary step is needed to send SA also from \(H_0\) to \(H_0\). The data accumulated at both PA and SA should be identical and they can be compared to confirm the obtained results.

**Malicious Host Identification:** Referring to Figure 2, the SA would suspect a malicious action if \(\text{ack}_{i+1}\) was not received at \(H_{i+1}\) within a proper time out \(T\). This time out should be large enough to account for the longest estimated execution time at host \(H_i\) in addition to the transmission and queuing times. \(H_i\) and \(H_{i+1}\) are suspects and SA will request help from the home host \(H_0\) to identify the malicious host and take corrective actions. However, if a late acknowledgement (\(\text{ack}_{i+1}\)) arrives at \(H_{i+1}\) after the expiry of the time out, this should reset the corrective actions and allow PA to proceed.

Figure 3 illustrates the solution scenario when host \(H_i\) is malicious. A similar scenario holds when host \(H_{i+1}\) is malicious. The home host will react by sending two inspection agents (IA) to the suspected hosts. As shown in Figure 3 the home host will send IA\(_0\) and IA\(_1\) to hosts \(H_i\) and \(H_{i+1}\), respectively. When the home host identifies the malicious one, it will send an empty new instance of the primary agent PA to the safe host \(H_{i-2}\) to meet SA which carries a copy of the collected data. SA will reload the collected data into the empty PA. The newly loaded PA will continue its itinerary skipping the malicious host.

Eliminating the malicious host from the new itinerary simply means that the safety window is properly corrected. In particular, if \(H_i\) is found malicious, the window will be \(\{H_{i-1}, H_i, H_{i+1}\}\). Alternatively, the window will be \(\{H_{i-1}, H_i, H_{i+2}\}\) if the malicious host is \(H_{i+1}\). This mechanism will be still safe even if there are multiple successive malicious hosts. As the mobile agent PA moves along its itinerary, the window continues to move until PA arrives back eventually to its home, in this case the window will be \(\{H_{i-1}, H_i, H_{i+2}\}\). The following solution handles either case. This solution is also robust if there were multiple successive malicious hosts.

- Upon suspecting a malicious action, SA will request help from the home host \(H_0\).
- \(H_0\) will react by launching two agents as follows:
  - Launching an inspection agent IA\(_0\) to host \(H_i\) with an itinerary \(\{H_i, H_{i+2}\}\). IA\(_0\) would not actually do any processing at \(H_i\) it will just drop there and then decide to move to \(H_{i+2}\). If \(H_i\) is the malicious host, it will block IA\(_0\). Otherwise, it will allow it to move to \(H_{i+2}\). If IA\(_0\) arrives at
Fig. 3: Identifying the malicious host.

If $H_{i+2}$, it will send an authenticated acknowledgement $\text{ack}_{i+2}$ to the home host $H_i$ implying that $H_i$ is safe. Next, $\text{IA}_0$ will destroy itself.

Launching an inspection agent $\text{IA}_1$ to host $H_{i+1}$ with an itinerary $\{H_{i+1}, H_{i+2}\}$. A similar argument holds here with respect to host $H_{i+1}$. If $\text{IA}_1$ sends the acknowledgement $\text{ack}_{i+2}$ from host $H_{i+2}$ to host $H_i$, this will imply that host $H_{i+1}$ is safe.

- If the home host $H_i$ receives $\text{ack}_{i+2}$ from either $\text{IA}_0$ or $\text{IA}_1$, this will identify which of the two hosts is safe.
- The home host $H_i$ will launch a new instance of the primary agent $\text{PA}$ to host $H_{i+1}$ with a modified itinerary skipping the malicious host. This empty $\text{PA}$ will restore the accumulated data from $\text{SA}$ and repeat the processing at host $H_{i+1}$.
- Next, $\text{PA}$ will travel to the safe host $H_{i+1}$ ($H_i$ or $H_{i+1}$) to continue its modified itinerary. It will do processing and collect data at that host then move to $H_{i+2}$. The window is now $\{H_{i+1}, H_{i+2}, H_{i+3}\}$.
- If $\text{ack}_{i+2}$ were not received at all at the home host $H_0$ after a proper time out, this implies that $H_i$ and $H_{i+1}$ are malicious. In this case, the home host will launch another empty $\text{PA}$ that must skip both and move directly to host $H_{i+2}$. Now the window will be $\{H_{i+1}, H_{i+2}, H_{i+3}\}$.
- If $H_{i+2}$ or $H_{i+3}$ or both are also malicious, the same actions starting from step (2) above should be repeated regarding the newly suspected hosts.

**Code and Data Integrity:** As mentioned previously a non-blocking host may change either the mobile code or the accumulated data or both. Changing the PA and/or SA code to be malicious is a direct threat to the next host(s) in the itinerary. It is the responsibility of each host platform to protect itself and implement all necessary precautions to be immune against malicious hosts. Several solutions to the “Agent-to-Platform” problem were presented earlier in the introduction and this threat is not considered further here. Alternatively simple and effective methods are discussed next to allow detecting any changes in the mobile code and/or the accumulated data. This would take place when PA and SA eventually return back to the home host.
**Code Integrity:** The home host can verify the code integrity of PA and SA using cryptographic techniques. It is a simple task to produce a digital signature [20] for the mobile code of PA and SA prior to sending them along their itinerary. These signatures must be verified when they eventually return back to the home host. Any changes in the mobile code can be directly detected. Normally the consequences of such changes should be assessed and handled accordingly.

Moreover to provide an acceptable level of code confidentiality, obfuscation techniques [8] are recommended for implementing PA and SA. Although the shadow agent always moves to non-blocking hosts, the concern of being reverse-engineered still exists, so it is also important to obfuscate its code. Obfuscating both PA and SA will also help to check data integrity as explained next.

**Data Integrity:** The integrity of the collected data can be verified by allowing PA and SA to collect the same data independently from the visited host machines. Eventually when PA and SA return back, the home host will compare the collected data. Its integrity is proven only when both sets of data are exactly the same.

This scheme is vulnerable to a malicious host attack should that host be able to change the collected data in both PA and SA in the same way. In this case, data comparison in the home host will yield an exact match and it will not detect any changes. The following solution is proposed to detect such attack.

The main idea is to store each and every piece of data collected by PA and SA in two different forms. When a malicious host changes the stored data in both PA and SA there is no way to do the same changes in the original piece of data. In particular, let:

- \( D_i \) = piece of data collected from the \( i^{th} \) host.
- \( F_1 \) and \( F_2 \) = two reversible transformation functions used by PA and SA, respectively.
- \( S_{i1} \) = stored form of \( D_i \) at PA.
- \( S_{i2} \) = stored form of \( D_i \) at SA.

Where:

\[
\begin{align*}
S_{i1} &= F_1(D_i) \quad \text{and} \quad D_i = F_1^{-1}(S_{i1}) \\
S_{i2} &= F_2(D_i) \quad \text{and} \quad D_i = F_2^{-1}(S_{i2})
\end{align*}
\]

Eventually at the home machine the following equality must hold to prove data integrity.

\[
F_1^{-1}(S_{i1}) = F_2^{-1}(S_{i2}) \quad \text{(1)}
\]

Should a malicious host make changes to the stored forms \( S_{i1} \) at PA and \( S_{i2} \) at SA, there is no way to adjust these changes such that equality (1) will remain true. When the home host checks for data integrity the above test will fail indicating invalid data.

A simple reversible function is to perform XOR operations between the collected data \( D_i \) and some randomly selected large constant. A constant value consisting of 128-bits will be secure against brute-force attacks and this implies performing the XOR operations on a 16-byte boundary. To extract the original data at home, similar XOR operations using the same random constant will do. PA and SA must use two unequal random constants to produce different transformations of the same piece of data \( D_i \).

An extra precaution may be useful to reduce the chances that a malicious host can understand the operation of PA and SA and guess their random constants. This is based on designing and implementing PA and SA in two different ways with the condition that both should collect the same data. This objective can be achieved by using code and data structures obfuscating [8].

**Protocol Simulation:** A simulation program is developed to examine the correctness and scalability of the proposed protocol regarding its ability to identify malicious hosts that are scattered arbitrarily in the mobile agent’s itinerary. Many simulation experiments with different scenarios are conducted to cover the cases of a) No malicious hosts, b) A single malicious host and c) Multiple consecutive malicious hosts. In all scenarios the mobile agent managed to visit all non-blocking hosts and skipped the simulated malicious ones. This gives confidence in the validity of the proposed protocol.

To examine the protocol scalability some scenarios were tested using a very long itinerary length with a total of 200 hosts and about 20% of the hosts were flagged to be blocking. These blocking hosts were randomly scattered in each scenario, individually and in groups of up to 4 consecutive hosts, along the itinerary. In all experiments the home host was able to detect and skip all blocking hosts in the itinerary. This proves that the proposed protocol is scalable.
Following is a simple example to explain the protocol operation. In this example the mobile agent moves in an itinerary of ten simulated hosts $H_1$ to $H_{10}$. As shown in Figure 4 $H_3$ is a malicious host and also $H_6$, $H_7$ and $H_8$ are three consecutive malicious hosts. The other six hosts are non-blocking. In this experiment the simulated mobile agent succeeded to visit all non-blocking hosts and skip all malicious hosts. According to the simulated protocol PA and SA will start their journey from the home host $H_0$ and eventually return back to it. In this example the following scenario is simulated:

- The home host $H_0$, not shown in Figure 4, generates two mobile agents PA and SA. SA stays at $H_0$ and PA departs to visit $H_1$.
- SA waits for an acknowledgement from PA when it departs from $H_1$ and arrives at $H_2$. It starts a timer ($T_{sa}$) that will expire after some specified period that is enough to receive the acknowledgement if there is no malicious action.
- After completing its task at $H_1$ and collecting the required data, PA moves to $H_2$ and then sends an acknowledgement to SA.
- SA receives the acknowledgement before the timer $T_{sa}$ expires. This implies that $H_1$ is non-blocking. SA moves to $H_2$ and executes the same task of PA to collect the same data as a backup. Next it waits at $H_2$ for a second acknowledgement sent by PA when it leaves $H_2$ and arrives at $H_3$. SA sets the timer $T_{sa}$ once more appropriately.
- After completing its task at $H_2$, PA moves to $H_3$. Being malicious, $H_3$ blocks PA and prevents it from execution. Therefore PA is unable to send the acknowledgement.
- When the timer $T_{sa}$ expires, SA will suspect either $H_2$ or $H_3$ to be malicious.
- SA will request help from the home host $H_0$.
- $H_6$ will launch and send two inspection mobile agents: IA$_1$ with the itinerary {$H_5$, $H_4$} and IA$_2$ with the itinerary {$H_5$, $H_6$}. It will set a timer ($T_{io}$) appropriately.
- $H_6$ is non-blocking and it will allow IA$_2$ to drop there and leave to $H_7$.
- $H_7$ is malicious and it will block IA$_1$. Therefore IA$_1$ will be unable to depart and send acknowledgement from $H_6$.
- IA$_2$ will send an acknowledgement from $H_6$ to the home host $H_0$. This informs $H_0$ that $H_6$ is non-blocking.
- When the timer $T_{io}$ expires without receiving IA$_1$’s acknowledgement, $H_6$ will decide that $H_7$ is malicious.
- Now $H_6$ will launch and send a new instance of the primary agent PA with the itinerary {$H_5$, $H_4$, … $H_{10}$} skipping the malicious host $H_7$.
- Upon arrival at $H_2$, the new empty PA will get all the data accumulated in SA including the data collected from $H_2$. Next, PA will move to $H_3$ and SA will set its timer $T_{sa}$ to be ready for receiving the next acknowledgement when PA arrives at $H_3$.
- The same scenario will continue until PA eventually arrives at the next malicious host $H_6$ that will again block PA.
- SA and $H_6$ will act exactly as in the previous blocking case.
- In this case there are three consecutive malicious hosts $H_6$, $H_7$ and $H_8$. $H_6$ will keep sending inspection mobile agents until eventually it will get an acknowledgement from the inspection agent that has the itinerary {$H_5$, $H_4$}. This implies that $H_6$ is the first non-blocking host after $H_1$ in the itinerary.
- Similar to step (14) above, $H_6$ will launch and send another new instance of the primary agent PA with the itinerary {$H_5$, $H_4$, $H_{10}$} skipping the three malicious hosts.
- After loading the accumulated backup data from SA and visiting the remaining hosts $H_7$ and $H_{10}$, PA will return back to the home host $H_0$ followed by SA.

The given scenario shows in details how would the home host interfere, in case of suspecting that some malicious host has blocked the primary agent. It would identify that malicious host quickly and launch a new
Table 1: Algorithm for the static home agent (HA).
1. Create the user interface that controls the prototype program.
2. Launch the mobile agents PA and SA. Both agents have the same itinerary \( \{ H_1, H_2 \} \).

   // The primary agent PA would autonomously migrate to the first host \( H_1 \).
   // The shadow agent SA would initially wait at home host.
3. Wait for a ‘HELP’ message from SA (at host \( H_i \), \( i = 1, 2, ..., N-2 \)) or for two ‘FINISHED’ messages from PA and SA when they finish visiting all hosts in the itinerary.
4. if (the received message is ‘HELP’) then
   a. Send a ‘FINISHED’ message to the static home agent (HA). This message contains the collected data.
   b. The primary agent PA disposes itself.
   c. The shadow agent SA disposes itself.
   d. Identify the malicious host(s) based on the received ‘ACKNOWLEDGEMENT’.

Table 2: Algorithm for the primary mobile agent (PA).
1. Initially PA is launched at home host \( H_0 \) to move along a specified itinerary.
   // from host \( H_1 \) to host \( H_2 \) and then back to the home host \( H_0 \).
   // PA keeps moving along its itinerary until it reaches the home host, or blocked
   // by a malicious host.
2. if (the visited host is \( H_i \)) then // \( i = 1, 2, ..., N \)
   a. if (\( i = 2 \)) then send ‘GO-TO-NEXT’ message to SA in host \( H_2 \).
   b. Execute the required task at host \( H_i \) and collect data.
3. Move to the next host \( H_{i+1} \). // \( H_{i+1} = H_1 \) is the home host.
4. if (\( H_{i+1} = H_0 \)) then
   a. Send a ‘FINISHED’ message to the static home agent (HA). This message contains the collected data.
   b. PA disposes itself.

Table 3: Algorithm for the shadow mobile agent (SA).
// Initially launched at home host \( H_0 \) and given the same itinerary as that of PA.
1. Set a timer for a suitable time-out period \( T_o \).
2. Wait for either receiving the ‘GO-TO-NEXT’ message from PA or for the timer expiry.
3. if (the ‘GO-TO-NEXT’ message is received) then
   a. Move to the next host \( H_{i+1} \) in the itinerary.
   b. if (\( H_{i+1} = H_0 \)) then // \( H_{i+1} = H_0 \)
   i. Execute the required task at host \( H_{i+1} \). This task collects the same data as PA did.
   ii. Go to step 2.
else
   i. Send a ‘FINISHED’ message to the static home agent (HA). This message contains the collected data.
   ii. SA disposes itself.
   else if (the timer expired) then
   a. Send ‘HELP’ message to HA.
   b. Wait for ‘NEW ITINERARY’ message from the new PA.
c. Pass all accumulated data to PA.
d. Go to step 2.

Table 4: Algorithm for the inspection mobile agents (IA_1 and IA_2).
// The same algorithm applies for both agents.
// For \( IA_1 \), the suspected host is \( H_{sus} = H_1 \) and the itinerary is: \( \{ H_1, H_2 \} \).
// For \( IA_2 \), the suspected host is \( H_{sus} = H_2 \) and the itinerary is: \( \{ H_2, H_1 \} \).
1. Move to the suspected host \( H_{sus} \).
2. Move to the next host \( H_{sus} \).
3. Send ‘ACKNOWLEDGEMENT’ message to the static home agent (HA).
4. The inspection agent disposes itself.

The “platform-to-Agent” problem is a challenging threat that hinders the widespread deployment of mobile...
agent technology in distributed applications. A malicious host has the full capacity to fully understand the operation of a visiting mobile agent and hence to update its accumulated data in its own favor. More severely, it can block the mobile agent and prevent it from continuing its itinerary. The present work investigates the possibility of securing mobile agent-based systems against blocking hosts and guaranteeing the integrity of the mobile agent’s data and code. A protocol is proposed to facilitate identifying and skipping any blocking hosts in the mobile agent’s itinerary. Simple and effective precautions are also suggested to detect any malicious updates in the mobile agent’s data that is accumulated from all visited hosts and/or the agent’s code. This would take place after completing the itinerary and returning back to the home host.

The suggested solution for identifying and skipping blocking hosts fits for sequential and hybrid (sequential/parallel) itinerary agent-based distributed system architectures. It is based on launching two mobile agents, primary and shadow, in the same itinerary to do the same functionality and collect the same data. However, the shadow agent will be always lagging behind the primary and always visiting hosts that did not block the primary. Whenever a malicious host blocks the primary agent, the shadow agent will detect that event and request help from its home host. The later will manage to identify the blocking host by sending a couple of inspection mobile agents to a pair of suspected hosts within the itinerary. An acknowledgement and time-out mechanism allows the home host to decide which of the two suspects is blocking. This mechanism is also effective to identify a series of cascaded blocking hosts. Once the home host identifies the blocking host(s), it will launch a new instance of the primary agent and send it to the shadow agent. The new primary agent will get all collected data from the shadow agent and hence continues its itinerary skipping the malicious host(s).

When both mobile agents return back, the home host will verify the integrity of their code and data. By comparing the digital signature of the mobile agent’s code before and after its trip, any changes in the agent’s code would be obvious. To check the data integrity the home host would compare the data collected by the primary and shadow agents. Data integrity is proven if and only if the same data is collected by both agents.

A fault-tolerance approach is suggested to ensure code and data integrity. Primary and shadow agents should be two variants having two different implementations. Code obfuscation is recommended to reduce the chances that a malicious host can understand their operation. An effective method is employed to detect any malicious changes in the data collected by the primary and shadow mobile agents. They do not actually store the collected data directly; each agent stores a transformed form of the same data using a different reversible transformation function. Before comparing the collected data to examine its integrity, the home host should first restore the original data collected by each agent using the inverse functions. In this way any malicious changes of the stored data forms will be detected because the two functions are different.

The proposed protocol is simulated and a practical prototype is developed and implemented using four host computers connected by a LAN. Simulation results showed that the suggested solution is effective and scalable to fit very long itinerary lengths. Different scenarios were simulated using 200 hosts in the itinerary and about 20% of these hosts were flagged to be blocking. The blocking hosts in these scenarios were arbitrarily scattered along the itinerary in a random mix of single and multiple successive malicious hosts. All simulation experiments proved that the suggested mechanism is robust against blocking attacks. Practical testing is conducted using a prototype comprising four host computers interconnected by a LAN. The successful results obtained practically give a positive indication concerning the robustness of the proposed solution in real life environment. In spite of the obtained encouraging results from the simulation and practical experiments, yet more testing in a large scale real distributed system is still needed. This real-life experimentation and also the verification of the data and code integrity would be the subject of future work.

REFERENCES