Preventing Attacks on Mobile Agents by Malicious Hosts

Nathan Balon
University of Michigan Dearborn

Abstract. The security concerns that come with using mobile agents is the main reason why the mobile agent paradigm has not been widely adopted. A number of new security problems are introduced with the use of mobile agents. Both, the host computer that executes an agent and the agent, need security mechanisms to protect against the threats posed by malicious agents and hosts. The issue of protecting a mobile agent from a malicious host is a more difficult problem than protect a host from a malicious agent. This paper will look at three solutions that have been proposed to protect agents from malicious hosts: creating time-limited black boxes, computing with encrypted functions, and environmental key generation.

1 Introduction

Computer networks have evolved with time. Originally, computer network applications were based upon a client/server model. Network applications have progressed to support distributed computing through the use of remote procedure calls and distributed objects. A further extension to the distributed model comes from the use of mobile agents. A mobile agent is an autonomous software entity that can execute on a host in a network and then suspend itself and transfer its execution to another host in the network. A mobile agent can be described as follows:

> Mobile agents refer to self-contained and identifiable computer programs, bundled with their code, data, and execution state, which can move within a heterogeneous network of computer systems. They can suspend their execution on an arbitrary point and transport themselves to another system. During this migration the agent is transmitted completely, that is, as a set of code, data, and execution state. At the destination computer system, an agent's execution state is resumed at exactly the point where it was suspended before [1].

Mobile agents offer many advantages when developing network applications, as compared to previously mentioned models. First, using mobile agents can conserve the amount of bandwidth needed by an application. For instance, an agent may be sent out to find the best price for an item. Instead of having to return each price found for the item to the client, an agent can go from host to host gathering prices and then return best price. One of the criticisms of mobile agent systems is that almost anything that can be done by an agent can be accomplished using traditional methods. Although this is often the case, mobile agents make accomplishing many tasks easier. One area where mobile agents surpass the other models is in the case of real-time systems. In this case, an agent can be moved to the node that it needs to operate on for time crucial tasks, which removes the delays caused by sending messages between a client and a server. One commonly proposed use for mobile agents is for e-commerce applications. However, the use of mobile agents is not limited to e-commerce, a number of other useful applications have been proposed ranging from network management to intrusion detection.

Although mobile agents offer a number of benefits over traditional network technologies, a number of new problems are introduced by the mobile agent paradigm. The main concern that comes with using mobile agents is the new security threats they introduce. In a client/server model, the client program sends messages to the server, which then executes the request on behalf of the client and returns the results to the client. Mobile agents, on the other hand, are able to migrate to the host machine and then execute there. The security threats in this model are: attacks to a host from an agent, attacks from a malicious host on an agent, attacks from other agents on agents, and attacks from the outside on the agent system [4].

When a mobile agent executes on a host, security mechanisms are needed to protect the host from malicious agents. Protecting a host from an agent uses many of the familiar security features found in traditional client/server computing. Some of the methods used to enforce security on a host are: digital signatures, encryption, access control lists and sandboxing.

Mobile agents also need to be protected from malicious hosts. This problem is more difficult to solve than protecting the host. Traditionally, when a program is developed it is assumed that the program will run in a safe environment. The problem of the execution environment attacking a program typically does not need to be addressed. For instance, if someone were to create a new text editor, the creator of the editor doesn't have to address the issue of the execution environment attacking the editor. Usually the owner of the program and the execution environment are the same person, in the case of a mobile agent system this is not the case. For this reason, little research has taken place in the area of protecting mobile code from a hostile execution environment.

One solution that has been proposed is that agents could be restricted to trusted hosts. Limiting an agent to a trusted host severely limits the usefulness of the mobile agent system [3]. In the case of electronic commerce, an agent would be limited in the number of hosts it could visit. Also, how does an agent determine which hosts are trusted and how will a new trusted host be added to the system? Because an agent must be executed on a remote host, some experts believe that there is no way to ensure the safety of an agent without using tamper-resistant hardware.

A number of possible software solutions have been proposed to protect mobile agents from malicious hosts. The proposed solutions to protect agents have aimed at either detecting attacks on agents or preventing them. This paper will explore some of the possible solutions, which can be used to prevent attacks on mobile agents. Section 2 will introduce a possible mobile agent application and some of the security problems associated with the application. Section 3 will discuss the threats in a mobile agent system on agents. Section 4 will look at some of the security mechanisms that have been proposed to protect agents. Next, section 5 will introduce three techniques: time-limited black boxes, computing with encrypted functions, and environmental key generation. Section 6 will then explore the methods used to conceal the mobile code of an agent. Section 7 will give a comparison of the benefits and drawbacks of using these methods.

Section 8 discusses the obstacles to implementing these security solutions. Finally, section 9 will present the conclusion.

2 A Sample Mobile Agent Application and the Need for Security

The classic example used to describe mobile agent systems is one that uses mobile agents to determine the best price for an airline ticket. In this example, the agent is sent out to find the best possible price for an airline ticket to a specific destination. An agent will be sent to visit each host that offers airline tickets. The agent will return to the user when all of the possible ticket prices have been evaluated, and it will return with a list of best prices to the agent's owner.

A number of attacks are possible on the agent searching for airline tickets. If no means are used to protect the agent's code, data, and flow control the agent is open to a number of attacks from malicious hosts. A malicious host could change the data that is carried by an agent. For example, a malicious host could delete airline ticket prices that are cheaper than it can offer, in an attempt to win the agent's business. If the agent contains some form of e-cash, the host may try to steal it. Furthermore, a malicious host could modify the flow control of an agent so that the agent will bypass other hosts with cheaper airline tickets. Next, if the agent's code is in plain text, the host could also learn any of the proprietary algorithms used by the agent. Last, the host could simply deny the execution of the agent, so the agent is never able to complete its task.

These are just a few of the possible attacks that could take place in a mobile agent system. All of these possible threats show the importance of having security mechanisms in place to protect a mobile agent from malicious hosts. Without security mechanism to protect the agent, this possible ecommerce application for mobile agents would not be possible.

3 Threats to Mobile Agents

In a mobile agent system a malicious host is defined as a host that executes an agent and tries to attack the agent in someway. When an agent is executed on a host it must use the resources available on that host. The host can monitor an agent's memory usage and each instruction given by the agent to the host. A malicious host may then attempt to attack an agent in a number of ways.

The four main forms of attacks by hosts on mobile agents are [4]:

- A host masquerading as another host.
- Denial of service by the host to the agent.
- Eavesdropping on an agent's activity.
- Alteration of the agent by the host.

3.1 Masquerading

A masquerading host may try to trick the agent into believing it is another host and cause the agent to give the host sensitive information. Once the masquerading host is able to gain the trust of the agent, it may then be able to read or modify any of agent's code, data and state if mechanisms are not put in place to protect this type of attack. The main solution to prevent this type of attack is to use a strong authentication protocol to authenticate a host to an agent.

3.2 Denial of Service

A host may deny an agent a specific service provided by the host. It is possible for a host to both intentionally and unintentionally deny an agent a service. A host may deny an agent service so that the agent is not able to complete its task. Another possible attack is the host could terminate the agent altogether. Furthermore, a host may deny a request from an agent on a time-sensitive task so that the agent is unable to complete its task in its allotted time [3].

3.3 Eavesdropping

The next attack that can be performed by the host on an agent is eavesdropping. In a client/server environment, the typical eavesdropping attack comes from the monitoring of a communication channel. In the case of a mobile agent system, malicious hosts may try to determine the code, data, or flow control held by the agent. This form of attack is difficult to prevent and detect. The goal of all three of the methods to be discussed in this paper is to limit the information that can be read by a host.

Even when all of the information is hidden from the host, the host may still be able to infer some information from the agent. The main problem is that the agent must execute on the host so the host is able to record each instruction given to it by the agent.

3.4 Alteration

The final form of attack by a host on an agent is the alteration of the agent. The host can alter an agent by changing the data, code and control flow. A malicious host may try to change the code of an agent so that the agent performs other tasks than were intended by its creator. A host may also try to change the data contained in the agent.

4 Security Mechanisms to Protect Agents

A number of approaches have been developed to protect mobile code. The approaches can be classified into four types of protection [5]:

1. Mobile agents are only able to migrate to trusted hosts in the system.
2. Organizational methods are employed to protect agents (i.e. creating a closed system where only trust worth parties can operate a host).

3. Tamper-resistant hardware is used to ensure the integrity of an agent.
4. Restricted environments are setup and cryptographic protocols are employed to make tampering with mobile code difficult and time consuming.

The security mechanisms focused on in this paper all uses the 4th form of protection.

A number of security solutions can be used to protect an agent from a malicious host. The security solutions used by a mobile agent can be classified into those used for prevention or detection [2]. Mechanisms aimed at prevention use security techniques to prevent the unauthorized access of code and data. On the other hand, mechanisms aimed at detection attempt to detect any unauthorized modification of an agent. The preventive techniques are proactive, while the detection techniques are reactive. Table 1 lists some of the countermeasures that have been created to protect mobile agents and states whether the mechanism is aimed at detection or prevention [3].

Countermeasure	***Category***
Mutual Itinerary Recording	Detection
Partial Result Encapsulation	Detection
Itinerary Recording with Replication and Voting	Detection
Execution Tracing	Detection
Time Limited Black box	Prevention
Computing with Encrypted Functions	Prevention
Environmental Key Generation	Prevention

Table 1. Countermeasures [3]

Some of techniques used for prevention employ methods such as: replication of mobile agents, the use of digital signatures to detect tampering and various cryptographic schemes. The goal of each of these mechanisms is to determine when an agent is attacked.

In contrast with mechanisms aimed at detection, the mechanism used for prevention aim at not allowing an attack to occur in the first place. The preventive techniques aim at hiding the code, data, and flow control from the hosts that execute them. The three main preventive techniques used to protect an agent are creating a time-limited black box of an agent, computing with encrypted functions, and the use of environmental key generation.

5 Preventive Security Mechanisms

Three methods have been proposed to prevent the analysis of a mobile agent. Each of the methods attempts to hide the code and data of the mobile agent. First, the time-limited black box approach conceals the code of a mobile agent by obfuscating the agent's code.

Second, cryptographic methods are used in computing with encrypted functions. Third, environmental key generation is used to generate a key to unlock the encrypted code contained in the agent.

5.1 Time-Limited Black Box (Obfuscated Code)

Fritz Holh proposed the use of a time-limited black box as a way of protecting mobile agents [2]. Each agent is a time-limited black box for which it is possible only to observe the input and output of the black box. In this situation, the host environment is given obfuscated code by the agent. The aim of using obfuscated code is that a host will execute the code and have no idea what the code is actually doing. One of the problems with this approach is there is no known way to enforce the black box property for an infinite period of time, for this reason a time limit was introduced. While there is no way to prove that host will not be able to determine the meaning of the code, Holtz suggests using a lower time bound for how long the agent is to remain valid. The lower bound is used establish how long it will take a host to discover the meaning of the code given to it by the agent for execution. After the time limit expires, the agent is invalidated. The major problem with creating an agent that has the time-limited black box property is determining what the time limit should be.

5.2 Computing with Encrypted Functions

Sander and Tschidun proposed computing with encrypted functions (CEF) [7]. The goal of computing with encrypted functions is to hide the actual functions used by the agent from the host. One of the main drawbacks seen with time-limited black box method is that it fails to provide a verifiable form protection. For this reason, CEF was created so a verifiable level of protection can be given to an agent. In this scheme each agent has a program that is made up of encrypted functions. The authors propose the use of homomorphic encryption both to keep the agent's functions secret from the host and host data secret from the agent.

5.3 Environmental Key Generation (Clueless Agents)

Another method that employs encryption is environmental key generation, which was introduced by Riordan and Schneier [6]. As opposed to introducing a new form of encryption, environmental key generation aims at hiding the key to the encryption algorithm. Environmental key generation is a mechanism that causes an agent to take a specific action when an environmental condition becomes true. Each mobile agent will contain a portion of its code in plain text and another portion that is cipher text. The agents are labeled clueless because the full capability of the agent cannot be discovered until the cipher text is decrypted. When the environmental condition becomes true, the agent is then able to execute the code that was hidden cryptographically. The environmental condition that triggers decryption is hidden by a one-way hash function.

6 Concealing an Agents Mobile Code

One assumption usually made is that a mobile agent must have plain text code and data, which makes an agent vulnerable to a range of attacks from a host. As mentioned in the previous section a number of methods have been developed to hide the code and data of a mobile agent. The following techniques: time-limited black boxes, computing with encrypted functions, and clueless agents, all make attempts to conceal their code from a host but do so using different methods.

The main goal of each of these proposed solutions is to conceal the execution of a mobile agent from a host. Before these three approaches were introduced it was believed that the only way that mobile agents could be protected was through the use of hardware solutions. The advantage to each of these approaches is they offer a software solution.

One of the main considerations when developing a security solution to protect mobile agent is that the solution does not require an interactive protocol (i.e. where the agent must send messages to its owner). If an interactive protocol is used in which an agent must always contact its home agency many of the benefits of using mobile agents are lost. For the most part, each of these techniques can be used without interacting with other hosts in the systems. Except for a few variations in some of the solutions, such as allowing a clueless agent to contact a trusted timeserver, each of the solutions can be used non-interactively.

The approaches developed to hide the identity of mobile agent are based on either obfuscation or encryption.

6.1 Concealing an Agent Using Obfuscation

The goal of a time-limited black box is to hide all of the information contained in an agent from others. The basis of hiding the internals of an agent comes from the use of obfuscation. The code and data contained in the agent is obfuscated so that it will take an attacker a long period of time to determine the internals of the agent. Because it is not possible to guarantee the secrecy of the agent forever with this method, Holh proposes putting a time-limit on how long the agent is valid. To be considered a time-limited black box the agent must posses the following properties.

Definition: Time-Limited Black Box Property [2]

- an agent is a black box if:
  1. for a certain known time interval
  2. code and data of the agent specification cannot be read
  3. code and date of the agent specification cannot be modified
- attacks after the protection interval are possible
  4. but these attacks do not have effects

6.1.1 Limiting the Time a Black Box is Valid

Since the level of protection offered by the black box is limited in time length, some method must be used to invalidate the objects contained by the agent. In this case, an agent will contain two types of data: token data and non-token data.

The token data are self-contained documents that depend on the identity of the issuer. Some examples of token data are electronic money and encryption keys. Since these items will be able to be determined after a period of time, an expiration date must be put on the token data so that they are not used after the time limit expires. The main drawback to this approach is it limits the items that can be carried in an agent. If a token needs a longer period of protection then can be offered by the time-limited black box then it must not be carried by an agent. For instance, it may be possible for an agent to carry an encryption key that the agent uses for a limited period of time and to have the key expire after the time period. On the other hand, agent wouldn't want to carry private key which it uses for public key encryption since these keys are typically valid for a long period time.

The non-token data is all the other data contained by the agent. In the case of the airline ticket example non-token data would be data such as the cheapest price for an airline ticket. After the black box expires, the variables cannot be used for an attack since they are not tied to the identity of the owner.

6.1.2 Creating a Time-Limited Black Box

The first question is how to create a time-limited black box. To create a time-limited black box, an obfuscating ("mess-up") algorithm is used. An agent and a random input parameter are given to a conversion algorithm to create a time-limited black box (see Figure 1). The input parameter is used to prevent dictionary attacks on the black box. By using an input parameter along with the mobile agent, it is possible to create a number of different black boxes from a single mobile agent. It is also possible to extend the life of an agent by having a trusted host re-obfuscate the agent. The one problem with re-obfuscation is the expiration date set on the token data needs to be adjusted.

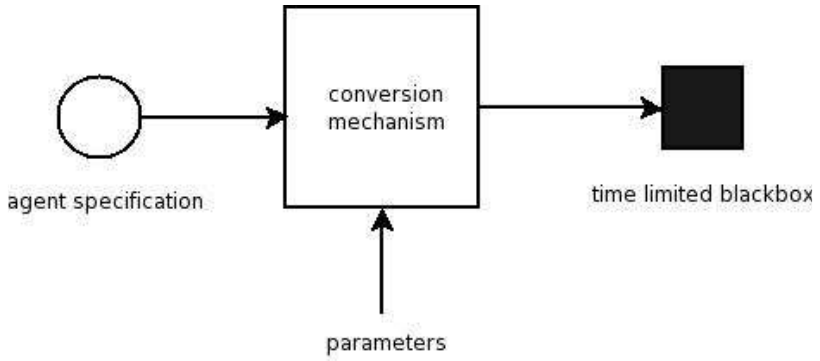

Figure 1. Creating a time-limited black box

6.1.3 Obfuscating Algorithm

An obfuscating algorithm is used to convert a mobile agent into a time-limited black box. While a host will still be able to read each line of code and know what it is doing, the goal is to obfuscate the code in a way that makes the overall meaning of the code unknown to the host, in turn making the agent hard to analyze. The obfuscating algorithm does not make it impossible to determine the mean of the code it simply makes the analysis take a long period of time. The level of protection given to the agent is based on how well the obfuscating algorithm "messes up" the code of the agent. Holh proposes three methods that can be used in combination when creating a time-limited black box.

1. Variable recomposition
2. Conversion of control flow elements into value-dependent jumps
3. Deposited keys

First, variable recomposition takes each variable in the code and cuts the variable into segments. The segments are then rejoined into a new segment. In this case, bits can be chosen from a number of different variables and then be recombined into a new variable. Figure 2 below show how three variables can be decomposed and then combined to create two new variables.

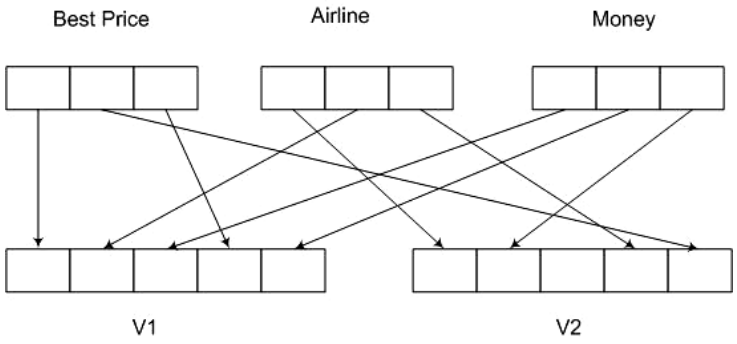

Figure 2. Variable recomposition

Second, the control flow of the program must be converted. It easy to gain an understanding of a program by examining its control statements (i.e. inspecting the "if" statements and "while" statements of a program). Hohl suggests that control statements should be converted to a form that depends on variable contents by creating switch statements and using complex variables.

Third, deposited keys are based on the idea that eventually the protection provided by the black box will be broke and the malicious host will discover the contents of an agent. In this case, the sensitive information that is need by the agent is held on a trusted server. The agent will indicate its state to a trusted server, and when the agent's state is correct the trusted server will return the information requested to the agent. The drawback to using this method is the agent has to continual contact the trusted server.

These are not the only methods that can be used to obfuscate an agent. Holh envisions that better algorithms will be developed to "mess-up" the code and data of an agent.

6.2 Concealing an Agent Using Encryption

The problem with using an obfuscating algorithm to hide the code and data of an agent is it generally is not possible to prove, with mathematical certainty, the amount of time it would take an attacker to determine the content of an agent. Also, a number of tools have been developed to reverse the obfuscation of the mobile code. As better obfuscating algorithms are developed better tools are being created to reverse the obfuscation. It is a never ending battle to come up with new methods of obfuscation to stay ahead of attackers. For this reason, a number of schemes were developed that use encryption to protect an agent.

6.2.1 Concealing Functions and Data with CEF

Computing with encrypted functions builds upon the work done by Holh. Instead of creating a black box that is limited by time, Sander and Tschidun use the notion of a black box but attempt to remove the time limit.

The problem that computing with encrypted functions tries to solve:

> Alice has an algorithm to compute a function f. Bob has an input x and is willing to compute f(x) for her, but Alice wants Bob to learn nothing substantial about f. Moreover, Bob should not need to interact with Alice during the computation of f(x) [7].

The solution proposed is to encrypt the functions used by the mobile agent so that the host is unable to learn anything about the function. In this scheme the functions used are encrypted and are represented by E(f). Then P(f) is the program that contains the encrypted function E(f).

Computing with encrypted functions includes the following steps [7]:

1. Alice encrypts f.
2. Alice creates a program P(E(f)) which implements E(f).
3. Alice sends P(E(f)) to Bob.
4. Bob executes P(E(f))(x) for Alice.
5. Bob sends P(E(f))(x) to Alice.
6. Alice decrypts P(E(f))(x) and obtains f(x).

In the following example Alice would be the owner of the mobile agent and Bob would be the host that executes the agent.

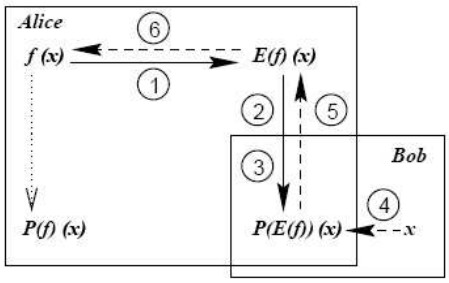

Figure 3. Computing with encrypted functions [7]

Sander and Tschudin implemented the encryption using an algebraic homomorphic encryption scheme. The protocol works for polynomials over rings *Z/MZ* with smooth integers *M*, where smooth integers are those that consist of small prime factors [1].

The main problem with computing with encrypted functions is that there is no known solution to implementing them on a large scale. Random programs cannot be encrypted using this scheme. The authors found that the encryption they propose only works for a restricted set polynomial and rational functions. The paper shows that there is a possibility of using encrypted functions in the future to protect mobile agents. Some critics on the other hand feel that it is unlikely that a solution to the encrypted function problem will be found. The authors envision that their paper will form the basis of a new field of study, which they label "mobile cryptography".

6.2.2 Creating Clueless Agents

Environmental key generation uses the notion of clueless agents. This method differs from computing with encrypted function in that only a portion of the agent is encrypted. The agent is considered clueless because it is unable to have its encrypted code executed until a certain environmental condition is met. When the environmental condition is met, a key is generated and the cipher text is decrypted. Then, the previously encrypted code or data can be acted upon.

A one-way hash function is used to test if the environmental condition is true and then creates the key needed by the agent. A number of possible hash functions can be used to generate the encryption key [6]:

1. if H(N) = M then let K := N
2. if H(H(N)) = M then let K := H(N)
3. if $H_i(N_i)$ = M then let K := $H(N_1,\ldots N_i)$
4. if H(N) = M then let K := $H(R_1,N)$ xor R_2

The following variables represent: N the integer value for the environmental condition, H the hash function, M a value carried by the agent, R a nonce, K a key.

A number of variations are possible for how the encrypted keys can be generated. First, the agent can look for the key on a fixed channel. Possibilities where the key could be

located include: within an html document, hidden using stenography in an image, on a newsgroup, and in a file system [6]. A second approach is to use a trusted timeserver to generate a key. It is possible to create a number of hashing schemes that generate the key based on either a forward or backward time. In the case of forward time, the key can only be generated after a certain amount of time has passed. The backward time restriction only allows the key to be generated before a certain time. A number of possible variations of these are possible.

One example where environmental key generation could be used is the case of an agent going out to search for something that the agent's owner doesn't want the host to know it is searching. Riordan and Schneier give the example of Alice searching a patent database using a mobile agent without allowing the host to discover the patent for which she is trying to locate. In this case, Alice would create a key using a hash function from a phrase she trying to find in the database. Alice would also create a random nonce. Next, Alice would then encrypt the code that she wished to have executed by her agent when the phrase being searched for is found. Furthermore, Alice would then generate a hash value from the nonce and phrase. This hash value would be used to compare if a successful match is found in the database. If a successful match is found, the key is generated from the hash function and the hidden code is decrypted then executed. In the case of Alice's agent searching through the database, her code will only be executed when a match is found. On host where no match is found, they will have no idea of the intention of Alice's search.

The mobile agent would be constructed using the following steps [6]:

- N := a random nonce
- K := H("smoke detector with a snooze alarm")
- M := E_k("report findings to alice@whatever.com")
- O := H(N xor "smoke detector with a snooze alarm")

When the mobile arrives at the host's database it executes the following algorithm:

- **for** six word sequence (x) **in** the database **do**
- If H(N xor (x)) = O then **execute** $D_{H(x)}(M)$

7 A Comparison of Prevention Techniques

Each of these methods discussed in this paper takes a slightly different approach in protecting the mobile code of an agent. The table below gives a comparison of the methods used to hidden the mobile code of an agent.

	Time-Limited Black Box	Computing with Encrypted Functions	Environmental Key Generation
Method used for Hiding code and data	Obfuscation	Homomorphic Encryption	Keys are generated from one way hash functions
Limited by a time interval	Yes	No	No
Code is executable without decrypting	Yes	Yes	No
Entire program hidden	Yes	Yes	No
Know implementations	Yes	No	Yes
Protection is mathematically provable	No	Yes	Yes

Table 2. Comparison of methods

7.1 Consequences of Time Limit on Protection

The benefit of clueless agents and CEF is they impose no time limit on how long the protection is valid. On the other hand, the time-limited black box approach restricts the life of an agent. By an agent having a time limit, the applications which the agent can be used in are limited. For example, if an agent was created that needed to exist for months or years in the system, then they would not be able to use a time-limited black box. Encrypted functions and environmental key generation do not suffer from this problem.

7.2 Time-Limited Black Boxes Have Restrictions on Payload

Another problem with the black box approach is that it limits the type of information that can be contained in the black box since it is only made hard to determine the content of the data and code. For instance, the creator of an agent would not want to put something in the agent that they would never want discovered such as a social security number. A time-limited black box is restricted by the type of data it can carry.

7.3 Benefit of Executing Hidden Code

One benefit of using CEF and time-limited black boxes over clueless agents is that the code can be given to a host and executed without the having to decrypt the code. The benefit of this is it shields the true identity of an agent. CEF and time-limited black boxes both create a black box of an agent. While a portion of the code of a clueless agent remains hidden from a host, when the environmental condition becomes true the code will be decrypted. If the code is decrypted on a hostile host, the host could then attack the code. Neither, CEF or time-limited black boxes suffer from this problem since the code and data they contain is always hidden from the host and it doesn't need to be decrypted to be executed.

7.4 Reduced Threat of Alteration

A benefit of each of the solution is they greatly reduce the threat of alteration attacks. Each agent's content is either encrypted or obfuscated. There is nothing to stop a malicious host from changing individual bits in the program, but the host would have little knowledge what the actual bits are they are changing since they are either encrypted or obfuscated. While it is still possible for a host to alter an agent, the host would not know the effects the alteration will have.

7.5 Possibility of Attacks on Unencrypted Code of a Clueless Agent

Another problem with clueless agents is that a portion of code contained by the agent is always exposed. While it is difficult to attack the hidden code contained in the agent, it is possible to attack the code of the agent that is not encrypted. For example, a malicious host could add or remove instructions to the plain text code of a clueless agent. In this case some additional security features are need such as those used to detect attacks on mobile agents. One possibility is for the clueless agent to have a digital signature of its code.

7.6 Black Box Attacks Are Possible

An additional problem with all three solutions is that a malicious host could run black box attacks on the agent. There is nothing to prevent a host from executing an agent multiple times. The host could continually execute the agent giving the agent different input each time. The host could then observe the output of the agent and any changes to the agent and try to infer information about the agent [1].

7.7 The Security Gained by the Agent Come at a Cost to the Host

An additional dilemma with each of the proposed solutions is that the agent is hiding the meaning of code from a host. The host is no longer able validate that the code is safe for execution. Each of these solutions contradicts the security requirements of the host. What is gained in security of an agent is lost in the security of a host.

7.8 Choices of Solution

The application being developed also has an impact on the choice of security solution to use. In the case of the searching for air line tickets the use of a time-limited black box may be preferred over using a clueless agent. It would be hard to create clueless agent that where a range of possible values are used as the key (i.e. creating a clueless agent that took some action for a range of ticket prices). On the other hand, clueless agents would be more applicable to situation where the value being searched for was already known as the case of the blind search of the patent database. The needs of the application should be evaluated before choosing one of these security solutions.

8 Implementation

There are very few commercial mobile agent toolkits. The majority of mobile agent toolkits to date have been created by the academic community. Some of the popular mobile agent toolkits from the academic community are: Mole, D'Agents, and Ajanta. Also, a number of open source toolkits have been developed, the most popular being Aglets. The majority of the work being done in the field of mobile agents is in the area of research as opposed to creating commercial systems.

Each of the solutions presented in this paper are also currently in the research stage of development. The most promising solution is the use of computing with encrypted functions. The benefit of this solution is that the code and data of the agent always remains secret and the level of protection is mathematically provable. The drawback of this approach is at the present time there is no known way to implement this encryption method, so this solution has yet to be implemented.

The use of a time-limited black box at the current time could only be used in applications that require a medium level of security. Much work still must be done to use this approach in a production environment. Further research needs to be conducted to create techniques to verify the level of protection offered by obfuscation. Also, further research must be conducted on how to determine the time length of the black box. Overall, it would not be hard to implement the time-limited black box. A number of programs are currently available that will obfuscate the code of a Java program and must current mobile agent systems use Java. The problem as mentioned is determining the length of time the agent is valid.

Implementing a mobile agent as a clueless agent is the solution that is the applicable at the present time. This solution does not have any major obstacles to prevent its implementation as is the case with computing with encrypted functions and time-limited black boxes. To implement clueless agents in there simplest form all that is need is a hash function and a symmetric encryption protocol.

9 Conclusion

Having security features to protect mobile agents in the past was considered a nice feature to have for a mobile agent system. Much of the attention in the past focused on creating security solutions to protect a host in a mobile agent system. Without security mechanisms to protect a mobile agent many of the proposed uses for agents would not be possible. In the case of the scenario of purchasing an airline ticket, a number of attacks on agents are possible if appropriate security measures aren't in place. Digital signatures can help to protect an agent against some attacks. For example, if an agent's code is modified, the owner of the agent is able to detect the attack through the use of digital signatures. Digital signatures and many previous solutions fail to protect a malicious host from determining the data and algorithms used by agent.

A difficult problem is finding a way to keep an agent's code and data private. There are times when an agent's owner doesn't want the code or data of their agent exposed for various reason. Three possible solutions were explored: time-limited black boxes, computing with encrypted functions and environmental key generation. Each can be used to protect an agent from this form of attack. While each of these solutions is not perfect they provide some form of protection to a mobile agent. None of these solutions solve all of the problems needed to keep a mobile agent secure. These methods may be combined with other methods to provide a secure environment for mobile agents.

This paper explored the importance of preventing attacks on mobile agents. In the future, each of the solutions presented in this paper should be further researched to overcome the problems associated with each method.

References

[1] Peter Braun and Wilhelm Rossak. Mobile Agents: Basic Concepts, Mobility Models, & the Tracy Toolkit. Morgan Kaufmann Publishers, San Fransisco, CA, 2005.

[2] Fritz Hohl. Time Limited Blackbox Security: Protecting Mobile Agents from Malicious Hosts. G. Vinga (Ed.), Mobile Agents and Security, pages 92-112, Springer-Verlag, Lecture Notes in Computer Science No. 1419, 1998.

[3] W. Jansen. Countermeasures for Mobile Agent Security. In Computer Communications, Special Issue on Advances in Research and Application of Network Security, November 2000. http://citeseer.ist.psu.edu/article/jansen00countermeasures.html

[4] W. Jansen and T. Karygiannis. NIST Special Publication 800-19 - Mobile Agent Security. National Institute of Standards and Technology, 2000. http://citeseer.ist.psu.edu/jansen00nist.html

[5] R. Oppliger. Security Issues Related to Mobile Code and Agent-Based Systems. Computer Communications 22 pages 1165-1170, 1999.

[6] J. Riordan and B. Schneier. Environmental key generation towards clueless agents. Mobile agents and security 19, 1998. http://citeseer.ist.psu.edu/639981.html

[7] Tomas Sander and Christan F. Tschudin. Protecting Mobile Agents Against Malicious Hosts. Mobile agents and security, volume 1419 of Lecture Notes in Computer Science, pages 44--60. Springer-Verlag, New York, NY, 1998.