Protecting the Itinerary of Mobile Agents *

Uwe G. Wilhelm and Sebastian Staamann
Laboratoire de Systèmes d'Exploitation

Levente Buttyán
Institut pour les Communications informatiques et leurs Applications

Ecole Polytechnique Fédérale de Lausanne, 1015 Lausanne, Switzerland
e-mail: {Uwe.Wilhelm, Sebastian.Staamann, Levente.Buttyan}@epfl.ch

Abstract

Systems that support mobile agents are increasingly being used on the global Internet. An important application that is considered for these agents is electronic commerce, where agents roam the World Wide Web in search of goods for their owners. In these applications, an agent moves along some itinerary in order to search for the best offer for the good sought by the user. The problem with this approach is that malicious providers on the agent's itinerary can damage the agent, tamper with the agent so that the agent itself becomes malicious, or forward the agent to any arbitrary provider that might not be on the agent's itinerary.

In this presentation we will primarily address the question how an agent can safely follow some predefined itinerary. We will identify the problem of trust as a major issue in this context and describe a trusted and tamper-proof hardware that can be used to enforce a policy. Based on this policy, we will show how the agent can take advantage of it in order to achieve the desired goal.

1 Introduction

New approaches for distributed computing based on mobile agent[1] technology, such as *Java*, *Telescript*, or *Agent Tcl* become ever more pervasive and, with emerging standards for electronic commerce, constitute an interesting domain for research that may also have important economical implications.

A typical application of mobile agents in the domain of electronic commerce is an agent that roams the World Wide Web in search of some good for its owner. Such an agent is configured by its owner with all the relevant information about the desired good, the constraints that define under which conditions an offer from a provider is acceptable, and a list of some potential providers of the good, to which the agent will migrate in order to query for their offer. The path that defines the agent's journey between the providers is called the itinerary of the agent. Furthermore, the agent might hold information for one or several payment methods that it needs to finalize a purchase and which should not be available to any principal other than the one that receives the payment. Since the agent is vulnerable when it is executing at the provider, it is necessary that its owner obtains some guarantees concerning the execution of the agent. This is, for instance, necessary to prevent that a provider removes some information about a better offer from the memory of the agent, and thus tricks the agent into accepting this provider's offer. A similar problem exists for the agent's itinerary, since a provider can easily forward the agent to an arbitrary provider that can or cannot be on the agent's itinerary. The usual approach that is taken to provide some guarantees, is to assume that the providers are *trusted* principals [12] or at least to create a mechanism that allows to detect which of the providers on the itinerary misbehaved [18].

The notion of trust has long been recognized as being of paramount importance for the development of secure systems [6, 11, 23]. For instance, any conceivable system for authenticating users needs trusted functionality that holds the necessary authentication information (see e.g., [22, 16]). However, the meaning that is associated with trust or the notion of a trusted principal is hardly ever clearly defined in these approaches and the reader is left with his intuition.

In this paper we want to address the question of how trust in a certain principal can be motivated based on social or technical reasoning and we will discuss

*Research supported by a grant from the EPFL ("Privacy" project) and by the Swiss National Science Foundation as part of the Swiss Priority Programme Information and Communications Structures (SPP-ICS) under project number 5003-045364.

[1]The term mobile agent has been charged with many different meanings. We use it here in the context of distributed systems (as opposed to its use in artificial intelligence) to refer to an object that is capable to move between different agent platforms in order to accomplish some well-defined task on behalf of its owner.

our ideas in the context of protecting the itinerary of mobile agents. We believe that our findings can have important repercussions for the architecture of open systems for mobile agents, where everybody should be capable to easily become a service provider.

In the following Section 2, we will introduce our model for mobile agents and point out the problems related to trust within this model. Then, in Section 3, we will discuss the notion of trust and define its relation to policy, which will enable us to better assess the possible motivations for trust. In Section 4, we will introduce a piece of trusted hardware, the TPE, and a protocol, the CryPO protocol, which will allow us in Section 5 to show how these can be used to protect the itinerary of a mobile agent relying only on technical means. In Section 6, we discuss why we consider this to be a better way to approach the problem and what effects this has on the notion of open systems. Finally, Section 7 concludes the paper with a summary of the main contributions.

2 The mobile agent paradigm

The mobile agent paradigm (also referred to as mobile code, mobile computation, or program mobility) has been identified by many authors as a promising and innovative new approach to structure problems in distributed computing [3, 4, 8, 9, 19]. In [10], Harrison *et. al.* have pointed out that the mobile agent paradigm provides interesting solutions to many real-life problems, for instance in the context of:

- mobile users, where agents are sent out from a mobile computer in order to accomplish a well-defined task on behalf of the user while he is disconnected from the communication network. Once the user reconnects, the agent returns and reports the result of the task or the problems it encountered.
- high-bandwidth interactions, where an agent is sent to a database server that holds a large amount of unstructured data to search for some specific information for the user.
- resident agents, which are stationary agents that take residence at some service provider and handle simple routine actions for their owner (e.g., communication management for a mobile user, where the agent decides how to handle an incoming communication request).

In this presentation, we are not interested in the underlying technology that is used to implement the mobile agent paradigm, but we only require a rather simple model for our discussion. Therefore, we identify the following major abstractions that we associate with mobile agents. A mobile agent consists of code, data, and the current execution state, which can be marshaled by the *agent owner* in a transport format and subsequently sent to the *agent executor*. The agent can be confidentiality and integrity protected during transit to protect it against outside attackers through the use of cryptographic mechanisms. These mechanisms can also provide data origin authentication for the marshaled agent. The agent executor will then eventually unmarshal the agent and instantiate it on a special environment located at the agent executor, which is called *agent platform* (AP). Here, the mobile agent can interact with services of the local AP and other agents located at this AP and continue to accomplish the task it was given by the agent owner. The literature on agents (e.g., [3, 12]) distinguishes between two different approaches to agent mobility:

- weak mobility and
- strong mobility.

The former only allows a very restricted form of mobility, where the agent can only migrate once to another AP. When it has finished, the result of its remote execution will either be sent directly to the agent owner in form of a message, or the agent itself is returned to the agent owner who can then extract the result from the agent. Agents that exhibit this form of mobility are usually called one-hop or boomerang agents.

The latter approach does not impose such a general restriction on agent mobility, but allows agents to visit as many APs as is deemed necessary to accomplish the desired task. In this approach, the result of the agent execution can also be sent directly to the agent owner, for instance as an intermediate result before each migration, but usually it is kept in the execution state of the agent and transferred to the agent owner when the agent returns. Agents that exhibit this form of mobility are usually called multi-hop agents.

The reasons for restricting an agent's mobility are first that the execution environment in the AP and the agent transport format can be much simpler since the AP does not have to provide the current execution state (which is, for instance, not available from the Java virtual machine) and the transport format does not have to encode it. The second reason is that this approach has a simpler trust model, since any damage incurred by the agent to the AP (and thus to the agent executor) or by the AP to the agent (and thus to the agent owner) can easily be attributed to the other entity. If more than two principals are involved, the problem of accountability becomes much more difficult [12]. Each principal can easily defer any damage to actions by any of the other principals on the agent's itinerary. This problem has also been addressed by Vigna in [18], where he tries to resolve the problem of how to attribute the damage to one of the principals that the agent visited.

In this presentation, we are interested in the protection of the agent's itinerary, which we will more clearly define in the following section. Since this implies that an agent will generally visit more than two principals, we will only consider agent systems that support strong mobility. Furthermore, we will make the simplifying assumption that the complete itinerary of the agent is pre-defined in advance and given to the agent upon startup. We will illustrate the problem with a small example of a shopping agent, but first we want to discuss the notion of an itinerary in more detail.

2.1 The itinerary

We assume that the itinerary of an agent is given in the form of a list of AP descriptors, which identify the owner and the network address of the AP as well as other relevant information about the AP. Protecting this itinerary is important, since we assume that some agent executors are more trustworthy than others and it would be good if the execution of the agent could be limited to the trusted agent executors.

In order to achieve this protection, we have to ensure both the safety and the liveness of the agent. In order to ensure the safety, we have to make sure that the agent does not visit any AP that is not on its itinerary, that it does visit all the APs that are on its itinerary, and that it visits them in the correct order. In an actual implementation of this, it is sufficient to ensure that the agent starts on the first AP on its itinerary and that it takes each step according to its itinerary. Ensuring the liveness of an agent is a task that is even more difficult than ensuring the safety. It consists of mechanisms with which an agent can, for instance, survive the crash of the AP on which it is executing or cope with the unavailability of crucial communication links. In the following, we will only give some hints on how this might be accomplished, but we will mainly concentrate on ensuring the safety of the agent's itinerary.

2.2 The shopping agent

Consider the following task: a user, say Alice, tries to find the best offer for some item (e.g. a CD, an airline ticket, or a video on demand) from a well-defined set of providers for this item. This offer can be a quite sophisticated combination out of several variables, such as price, QoS, payment method, delivery method, etc. Alice configures her agent with all the relevant information in order to identify the best offer and she also provides the itinerary that the agent is supposed to follow. The agent also contains payment information that should not be disclosed to any principal other than the one that receives the payment. Alice launches the agent by sending it to the first provider on the itinerary, where it will negotiate an offer for the item and store this information in its current state. Then the agent will migrate to the next provider on its itinerary and negotiate an offer for the item from this provider. Once the agent has visited all the providers on its itinerary, it can decide which of them has provided the best offer and it can then migrate to this provider in order to finalize the purchase transaction[2]. This behaviour of the agent can be disturbed by the providers very easily. We will now discuss some of the threats that the agent has to cope with.

2.3 Threats to the shopping agent

There are two different forms of threats to the agent that we want to consider:

- threats to the agent's code or data and
- threats to the agent's itinerary.

The former are threats to the agent as an entity itself, where the agent executor gains access to internal data of the agent or manipulates code or data of the agent in order to change the behaviour of the agent[3] in such a way that it will not operate correctly or even behave maliciously when executing at another provider.

The latter are threats to the pre-defined itinerary of the agent, where the agent is forced to skip certain agent executors on its itinerary or is tricked into executing on others that are not on its itinerary. The result of such a manipulation may be similar to those of a direct manipulation of the agent, but the manipulation of the agent's itinerary might be more difficult to discover. As an illustration of the problem consider that a provider might simply not send the agent to its most fierce competitor, so that it can not obtain an offer from this competitor and will thus conclude that the manipulator made the best offer. In case this irregularity is discovered, the manipulator could simply pretend that the required communication link to its competitor was not available and that it did not act maliciously.

In a conventional agent system, when the agent owner sends a mobile agent to an agent executor in order to use some service, the agent owner loses all control over the code and data of the agent. The agent executor can:

- reverse engineer the agent's code,

[2] If we assume that the agent returns to the chosen provider in order to finalize the purchase transaction, then this constitutes a dynamic change to the agent's itinerary, which we do not allow in the basic model described above, but only in some relaxations discussed below.

[3] Since it can be assumed that the code of an agent is static, it is possible to apply some integrity protection, which makes it much more difficult for a malicious agent executor to manipulate the code. This assumption does, however, not hold for the agent's data.

- analyze the agent's data,
- arbitrarily change the agent's code or data, or
- send the agent to any arbitrary agent executor.

This constellation puts the agent executor in a much stronger position than the agent owner. The agent owner simply has to trust the agent executor not to use the methods described above to illicitly obtain confidential information from the agent that it has to carry in order to use the service or to protect it against changes to the agent's itinerary. There is no way for the agent owner to control or even know about the behaviour of the agent executor.

The reason for the imbalance between agent executor and agent owner in the mobile agent model as compared to service provider and service user in the client/server model is that in the former approach, the agent owner has no guarantees whatsoever concerning the execution of its agent. In the client/server approach, the service user relies on many assumptions that are so basic that one hardly ever thinks of them. Nevertheless, these guarantees allow to implement certain types of behaviour in the client part of the distributed application that can not be implemented in conventional agent systems (e.g., code will be executed at most once, code will be executed correctly, requests will be sent to servers in a particular and well-defined order, availability of a reasonably reliable time service, etc.). This advantage relies on the fact that the client implementation is under the physical control of the service user. Based on these assumptions, a service user can observe what is happening in the system and notice irregularities. Thus, he is able to react accordingly, for instance, to interrupt an ongoing transaction. Another possibility to take advantage of these assumptions would be to log any irregularities at the client side so that they can be provided as evidence in the case of a dispute with some service provider.

We intend to create an environment for mobile agents that allows them to base their execution on similar assumptions, so that it becomes possible for a mobile agent to protect itself from a malicious service provider.

3 The notion of trust

We already mentioned the importance of trust for security in distributed systems and pointed out the lack of a clear definition of what is meant by the terms trusted principal or trusted system. In the following, we present our analysis of possible trust relations between different principals.

A reason for the lack of a clear definition of trust could be that trust is more a social than a technical issue and consequently quite difficult to tackle entirely in a technical approach. The major problem stems from the fact that the notion of trust mixes the goals of a principal with its behaviour to achieve these goals. In order to trust some principal, it is usually necessary to concur with or at least approve of its goals (which are not always clearly stated) and to believe that it will behave accordingly. In our definition of trust, we will try to clearly separate these two issues by gathering the goals of a principal in a policy, which is a set of rules that constrains the behaviour of this principal for all conceivable situations. This policy has to be written down and made available to all other principals that interact with the issuer of the policy. Then, we define *trust in another principal* as the belief that it will adhere to its published policy.

The question of whether a certain principal can be trusted now consists of (a) checking its published policy in order to decide if it is acceptable and (b) to establish a motivation for the belief that it will adhere to its published policy. The former is quite difficult but can be supported by a formal specification of the security policy (similar to the approach in [14]). The latter, however, is a problem that is quite difficult to formalize.

We have identified two fundamentally different approaches to the problem of trust: the *optimistic approach*, where we give an entity the benefit of the doubt and assume that it will behave properly and try to punish any violation of the published policy afterwards; and a *pessimistic approach*, in which we try to prevent any violation of the published policy in advance, by effectively constraining the possible actions of a principal to those conforming to the policy. Both of these approaches have their advantages and disadvantages.

The optimistic approach is easy to implement, since it does not require any special measures to make some interaction possible. This is probably why it is the basis for most business done today. However, it requires some possibility to discover a policy violation after it has occurred. If such a possibility does not exist, then the approach degenerates to *blind trust*, which indicates that there is no particular motivation to believe that a principal will adhere to its published policy other than its own assertion. Blind trust is obviously a very weak foundation for trust and not recommended for any important or financially valuable transaction. It is therefore important to make the risk that a policy violation is discovered as high as possible by improving controls and establishing checkpoints.

Once a policy violation is discovered and if it can further irrefutably be attributed to one of the participants in the corresponding transaction, this principal should be punished according to the appropriate laws and the damage caused by the policy violation. The goal of this punishment is primarily to deter potential violators from committing such a policy violation in the first place. Depending on how this punishment is enacted, we identify the following two motivations for the belief that an entity will adhere to its published policy:

- trust based on (a good) reputation
- trust based on control and punishment

Trust based on reputation stems from the fact that the principal in question is well known and has very little to gain through a violation of its own policy but a lot to lose in case a policy violation is discovered. This loss is supposed to transpire from the lost revenue due to customers taking their business to another principal. Reputation is an asset that is very expensive to build up and that is invaluable for any company. Thus, a principal would not risk to lose its good reputation for a small gain and will consequently rather adhere to its policy.

Trust based on control and punishment means that we do not trust the principal at all, but rather the underlying technical and legal framework to ensure the principal's proper behaviour. Here, we explicitly introduce the same tradeoff by normal disciplinary actions such as fines or imprisonment, depending on the severity of the offence. The short term gain that might be achieved through a policy violation is supposed to be negated by appropriate punishment.

Obviously, there are many other problems with this approach, such as the enforcement of laws, which can be very expensive, is usually quite slow, and is sometimes very complex (in particular if the laws of different countries are applicable) or the different perceptions of punishment. A person who has not much to lose might readily risk some years of imprisonment for the possibility of a relatively large gain.

Another problem in the approach stems from the fact that many abuses of confidential information are not necessarily conducted for the purposes of the company that holds this information, but rather by malicious insiders of such a company, who do it for strictly personal reasons or financial benefits [17, 21]. Such abuses are even more difficult to discover (there are less people involved) and to punish (it has to be decided if only the employee for malicious behaviour, only the company for negligence, or both have to be pursued).

The problem to reliably discover a policy violation could be resolved by requiring a very high degree of transparency. However, this is difficult to achieve and it is quite likely that even trustworthy principals with a very good reputation might not accept it. We therefore assume that complete transparency is not a very useful tool for supervision. A better approach would be to designate specialized companies that execute frequent in-depth controls of the conduct of companies.

Finally, neither of the two approaches can prevent malicious behaviour, but they only try to compensate for it after it has been discovered. For many situations in real-life, where an offence might have an irreparable effect (e.g., the collapse of the Barings Bank, where the Bank officials trusted their trader [7]) or where a proper functioning of the system is absolutely essential, this guarantee might not be strong enough.

We would like to remark that most of these problems are also present in our every-day life and therefore quite well understood. However, the question stands if we can do better than that.

The pessimistic approach removes all these disadvantages by simply preventing any violation of the published policy. This would clearly be the best foundation for trust since we can solely rely on a principal's policy to verify that its behaviour will be acceptable. The behaviour of the principal becomes completely transparent as far as it is constrained by its policy without the need to actually supervise any particular action. If the policy prescribes a particular action for some event and if the policy is enforced then it is guaranteed that the action will take place. Unfortunately, this policy enforcement can not be realized in its full generality, but is limited to those policies (or rules of a policy) that can effectively be enforced with some uncircumventable mechanism. For non-enforceable policies, we still have to rely on optimistic approaches to trust.

There is no possibility to enforce rules within a policy without relying on some piece of trusted and tamper-proof hardware [4]. In the following section, we will describe such a piece of hardware and the requirements that have to be met so that it can be used to enforce certain rules of a policy.

4 Tamper-proof hardware and the CryPO protocol

We will first present the execution environment (TPE) that we rely on and then describe the CryPO (cryptographically protected objects[4]) protocol that uses it. Figure 1 gives an overview of the principals in the system.

The TPE manufacturer produces the TPEs, which can be bought by any agent executor. An agent owner has to trust the TPE manufacturer to design and produce its TPEs properly (see Section 6). The broker is basically a directory service to locate the other principals and to obtain their credentials.

4.1 Notation

The described approach relies on public key cryptography [5] (such as RSA [13]). A detailed description of cryptography and the corresponding notations is not

[4]We have originally chosen the term object since it is more general than the term agent.

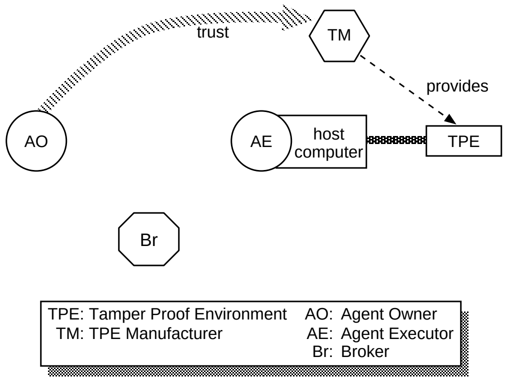

Figure 1: Overview of the Principals in the CryPO protocol

within the scope of this presentation, for information on this topic see, for instance [2, 15]. The notation we will use is as follows.

A principal P has a pair of keys (K_P, K_P^{-1}) where K_P is P's public key and K_P^{-1} its private key. Given these keys and the corresponding algorithm, it is possible to encrypt a message m, denoted $\{m\}_{K_P}$, so that only P can decrypt it with its private key. A signed message, including a digital signature on the message m, generated by P is denoted $\{m\}_{S_P}$.

In the following we assume the usage of optimization schemes such as encrypting a large message with a symmetric session key, which in turn is encrypted using public key cryptography and prepended to the message as well as the use of hash algorithms to reduce the amount of data that has to be signed. However, for ease of presentation, we will not make this explicit.

4.2 The execution environment

As we have noted above, there is no way to enforce any particular behaviour from another principal without a piece of trusted and tamper-proof hardware. We will discuss the problem of trust in the tamper-proof hardware in Section 6. The concept of tamper-proofedness usually applies to a well-defined module, sometimes called black-box, that executes a given task. The outside environment cannot interfere with the task of this module other than through a restricted interface that is under the complete control of the tamper-proof module (see Section 4.4). We will call this device *tamper-proof environment* (TPE). The TPE provides a complete agent platform that can not be inspected or tampered with. Any agent residing on the TPE is thus protected by the TPE both from disclosure and manipulation.

The TPE is a complete computer that consists of a CPU, RAM, ROM, and non-volatile storage (e.g. harddisk or flash RAM). It runs a virtual machine (VM) that provides the platform for the execution of agents and an operating system that provides the external interface to the TPE and controls the VM (e.g., protection of agents from each other). Furthermore, the TPE contains a private key K_{TPE}^{-1} that is known to no principal other than the TPE – also the physical owner of the TPE has no information concerning this private key. This can be achieved by generating the private key on the TPE[5]. Using this approach, the key is never available outside of the TPE and, thus, protected by the operating system and the tamper-proofedness of the TPE. The secrecy of the private key is a crucial requirement for the usage of the TPE to enforce a particular behaviour.

The TPE is connected to a host computer that is under the control of the TPE owner. This host computer can access the TPE exclusively through a well defined interface that allows, for instance, the following operations on the TPE:

- upload, migrate, or remove agents;
- facilitate interactions between host and agent or between agents on the TPE;
- verify certain properties of the TPE (such as which agents are currently executing).

Due to its implementation as a tamper-proof module and the restricted access via the operating system, it is impossible to directly access the information that is contained on the TPE.

This property is ensured and guaranteed by the TPE manufacturer (TM), which also provides the agent executor (AE) with a certificate (signed by TM). The certificate contains information about the TPE, such as its manufacturer, its type, the guarantees provided, and its public key. The agent owner (AO) has to trust the TM (see Section 6) that the TPE actually does provide the protection that is claimed in the certificate.

4.3 CryPO protocol

The CryPO (cryptographically protected objects) protocol transfers agents exclusively in encrypted form over the network to a TPE. Therefore, it is impossible for anyone who does not know the proper key to obtain the code or data of such a protected agent.

The protocol is divided into two distinct phases. The first phase consists of an initialization, which has to be executed once before the actual execution of the protocol. The second phase is concerned with the usage of the TPE and the actual transfer of the agent. The protocol is based on the interactions given in Figure 2.

[5]Other, more sophisticated approaches to create the pair of keys could be envisaged, which could also incorporate key recovery mechanisms (e.g., escrowed key shares).

4.3.1 Initialization: In the initialization phase, the participants establish the trust relations that are associated with the different keys:

- the TM publishes its certification key K_{TM}.
- the TM sends the certificate $Cert_{TPE} = \{K_{TPE}\}_{S_{TM}}$ to the AE.
- the AE registers its reference[6] with one or several brokers.

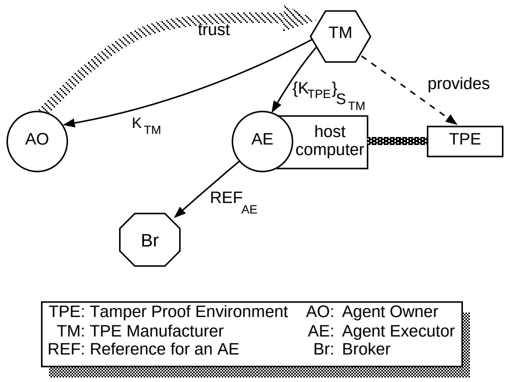

Figure 2: Initialization of the CryPO protocol

4.3.2 TPE usage: After the participants have finished the initialization, they can execute the usage part of the CryPO protocol:

- The AO queries the broker for the reference to the AE with which it wants to interact (or it already holds this reference from a previous interaction).
- The AO verifies the certificate $Cert_{TPE}$ to check the manufacturer and the type of the TPE, in order to decide if it satisfies the security requirements of the AO. If it is not satisfied with these checks, it will abort the protocol.
- The AO sends the agent encrypted with the public key of the TPE, $\{A\}_{K_{TPE}}$ to the AE.
- The AE cannot decrypt $\{A\}_{K_{TPE}}$ nor can it do anything other than upload the agent to its TPE.
- The TPE decrypts $\{A\}_{K_{TPE}}$ using its private key K_{TPE}^{-1} and obtains the executable agent A, which will eventually be started and can then interact with the local environment of the AE or other agents on the TPE.
- The agent can, after it has finished its task, send a message back to its owner or migrate back to its owner or to another AE to which it holds a reference.

[6] A reference to an AE consists of its name, its physical address in the network, its policy, and the certificate $Cert_{TPE}$ for its TPE. The broker can also verify that the AE actually controls the corresponding TPE by executing a challenge-response protocol with the TPE via the AE.

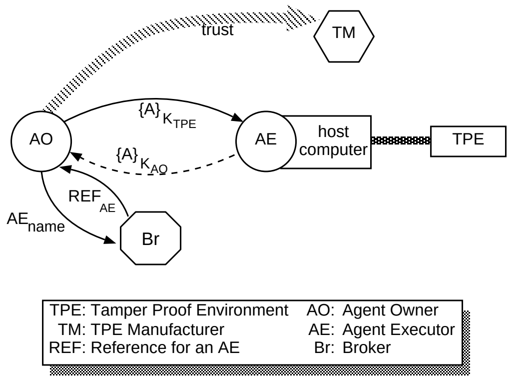

Figure 3: Usage of the CryPO protocol

The obvious problem of protecting the TPE from malicious agents is independent of the described approach and has to be tackled with additional mechanisms, such as code signing. The problem of protecting the TPE from tampered agents can easily be solved by concatenating the agent with a well known bit-pattern (magic number, MN) before encrypting it $\{A, MN\}_{K_{TPE}}$. The TPE simply has to verify the correct MN before starting the agent.

4.4 Notes on feasibility

The actual construction of a tamper-proof module in the real world is difficult; nevertheless, there are many applications that rely on them (e.g., payphones, debit cards, or SIM cards for GSM). Given sufficient time and resources, it becomes very probable that an attacker can violate the protection of such a module [1].

We believe that the actual realization of the presented TPE with reasonably strong guarantees in real-world settings is also quite difficult, but nonetheless feasible. Especially, since we only require the detection of tampering for most envisioned applications.

We imagine the TPE as a regular computer with a special operating system. It is physically protected with a special hardware that can effectively be sealed to detect tampering, is under continuous video surveillance similar to the systems used to supervise ATMs, and is subject to challenge inspections by the TM or an independent appraisal and inspection organization. As explained in [1], such an installation is conceivable and can even resist massive attacks. A thorough analysis of the remaining risks has to be undertaken, but this is not within the scope of this presentation.

5 Usage of the TPE to guarantee an agent's itinerary

The CryPO protocol and the TPE described above guarantee the integrity of the agent platform to the AO and protect the code and data of an agent against manipulation and disclosure, both in transit and during execution. These guarantees are based on the trust relation between the AO and the TM, where the AO trusts the TM to properly manufacture its TPEs and to control them regularly (if necessary) so that the claimed guarantees hold. The certificate enables the AO to ensure that it really deals with a TPE from a certain manufacturer.

The above guarantees can then be extended to formulate rules of a policy that can effectively be enforced by a TPE. In [20], we have discussed these issues in more detail and have shown how this approach can be used to implement limited lifetime (i.e., an agent can only be executed up to a certain point in time) and at-most-once execution (i.e., an agent can be executed once, but after it has finished, it can never be executed again) of an agent. If the enforced policy rules provide sufficient protection for a given agent, then the user does not need to trust the AE, but it suffices to trust the TM. The problem of why the agent owner should trust the TM is discussed in section 6.

Here, we want to investigate how a similar approach can be used to assure an agent owner that his agent will follow a pre-defined itinerary. As we have already seen, this problem consists of ensuring that the agent will neither migrate to a TPE that is not on its itinerary nor skip a TPE that is on its itinerary. Therefore, it suffices to make sure that the agent will migrate exactly to the TPE that is next on its itinerary. This resolves the problem of ensuring the safety part of this problem. The complementary problem of ensuring liveness, i.e., to make sure that the agent will actually continue its execution on the next TPE is another difficult task, for which we will only sketch a solution.

We will start by identifying the policy that a TPE has to enforce, to which we will refer to as *itinerant safe* policy. Then we will describe the *send-agent* approach, which takes advantage of this policy to ensure that the agent will migrate exactly to the next TPE on its itinerary.

For reasons of simplicity, we assume that the AO provides the agent with a pre-defined itinerary upon its start and that the AO has also verified that all the TPEs on the itinerary enforce the itinerant safe policy, which is encoded in the TPE's certificate.

This limitation can easily be removed by having the agent perform the policy verification itself before migrating to a new TPE. This might possibly require a slightly stronger protection by a TPE that supports this (i.e., some new policy rules). Since the agent is executing on a TPE that enforces an itinerant safe policy at its start – the AO verifies this, before sending the agent – and since it will only migrate to a TPE that also enforces an itinerant safe policy, by induction, we can conclude that the agent will never execute on a TPE that does not enforce an itinerant safe policy.

5.1 An itinerant safe policy

We assume that the policy rules enforced by the TPE are the following:

<a> the code and data of an agent will never be disclosed by the TPE.

<b> an agent is protected from any interference from other agents executing on the same TPE (other than calls on its public interface).

<c> an agent can create new agents on the TPE.

<d> an agent can explicitly pass a capability to another agent on the same TPE in order to allow it to perform special management operations on this agent.

<e> the TPE provides a capability protected management operation to freeze[7] an active agent.

<f> the TPE provides a capability protected management operation to marshal a frozen agent into a sequence of bytes.

<g> the TPE provides a cryptographic interface to encrypt a sequence of bytes with a public key[8].

<h> the TPE provides an I/O operation to send a sequence of bytes to some network address.

Apart from rule <a>, which is a basic rule for almost any TPE, all the other rules are regular operating system functionality that is well understood. The rules <b>, <e>, and <f> that require protection of agents from one another are readily implementable using normal memory protection mechanisms. The capability that is required in rules <d>, <e>, and <f> can be a simple random bit-string that is sufficiently long to prevent guessing attacks from other agents.

5.2 The send-agent approach

The use of the itinerant safe policy to guarantee that an agent will migrate from a TPE T_1 to a specific TPE

[7] The agent that will be frozen will be informed about this action, so that it can perform any necessary cleanup operations before being frozen.

[8] For performance reasons, the TPE might also use a hybrid encryption scheme, where it only encrypts a session key with the public key and the bulk of the data with the session key using a symmetric encryption scheme.

T_2 is now straightforward. We assume that the original agent A_O carries the code for the send-agent A_S with it. A_O will now take advantage of rule <c> and <d> to instantiate A_S on the TPE (using a primitive similar to `fork` on the UNIX operating system) and provides it with a capability to access the management functionality on A_O as well as the certificate for T_2, which contains T_2's policy, the public key of T_2, and a network address for T_2.

A_S will now take the necessary actions to freeze A_O (rule <e>) and to marshal A_O into a sequence of bytes (rule <f>). For this, A_S needs the capability that it received from A_O. Then A_S will encrypt the marshaled A_O (rule <g>) using the key in the certificate for T_2 and send it to the network address given in the certificate.

In order to guarantee the liveness of an agent that is executing on a TPE, the TPE needs to implement some form of checkpointing mechanism that allows the agent to survive a possible crash of the TPE (or another mechanism with a similar functionality). Solving this problem is in itself a complex task and we will not discuss it here any further. However, in order for A_S to ensure that A_O will actually continue its execution on T_2, it is necessary that A_S waits for a confirmation stating that A_O has successfully created such a checkpoint on T_2. If A_S does not receive such a message after a certain timeout, it might simply re-instantiate A_O on T_1 and inform it about the unsuccessful migration attempt. A_O can then take appropriate actions to recover from this error.

No other agent on T_1 is capable to execute the operations that give access to A_O's code or data in any way (rule <b>). Furthermore, since T_1 will never disclose any information on any agent executing on it (rule <a>), the only way that any information about an agent on a TPE can ever become available outside of this TPE is through proper actions of the agent itself (by sending messages or by using the send-agent approach for migration). Therefore, we conclude that the presented approach allows to guarantee that an agent will migrate exactly to the next TPE on its itinerary and will, thus, follow its pre-defined itinerary.

To simplify this approach to agent mobility, the code of A_S could also be integrated in the operating system of the TPE, which could offer access to this functionality in the form of a system call that performs the migration (e.g. jump, go, etc.). However, the underlying policy and the necessary actions that are required to achieve this, are the ones described above.

6 Trust in the TPE manufacturer

We have just introduced the mechanism, with which an agent can take advantage of the policy enforced by a TPE. However, as we have mentioned above, in order for a user to trust in the proper enforcement of this policy, it is necessary that he also trusts the TPE manufacturer to properly design, implement, and produce its TPEs. Since there is no way (to the knowledge of the authors) to enforce a correct behaviour of the TPE manufacturer, it seems that the presented approach simply replaces one required trust relationship with another one. This is a correct observation from a theoretical point of view. Nevertheless, we believe that this replacement of trust in an arbitrary service provider with trust in a TPE manufacturer has several more subtle implications. We will briefly discuss the following advantages that we identified:

- better understanding of security and privacy problems
- centralized control
- resources to build reputation
- separation of concern

The TPE manufacturer is a specialized service provider, which primarily deals in the field of the provision of security devices. Therefore it has a better understanding of security and privacy problems, which makes it a much more capable entity to ensure this service since it is more aware of the potential problems and pitfalls.

We assume that there will be relatively few TPE manufacturers (on the order of several hundreds) compared to the number of possible operators of the TPE (on the order of several millions). This makes the control of their behaviour much easier for expert appraisal organizations. Also, it is quite conceivable that a TPE manufacturer might invite external experts to control its internal operation, in order to obtain a better position in the market (similar to the approach for quality assurance in the ISO-9000 or the approach taken by Intermind[9]).

The production of TPEs is considered to be a difficult task (see Section 4.4). Therefore, we assume that it will be undertaken by major corporations, which have the necessary resources to build a good reputation and which have an incentive to protect this reputation. This allows us to rely on good reputation as foundation for trust in the TPE manufacturer.

[9] Intermind was evaluated by a Big 6 accountants firm to verify the implementation of its privacy policy. For further information see press release from June 6, 1997 on `http://www.intermind.com/`.

The TPE manufacturer that is responsible for the enforcement of the proper policy rules on the TPE, has nothing to gain by not accomplishing its task. Since the TPE will be operated independent from the TPE manufacturer by a completely different principal and since the TPE manufacturer has no means to access the data that is processed on the TPE (no physical connection), there is no possibility for the TPE manufacturer to draw a direct benefit from a TPE that does not properly enforce its policy[10].

We assume that the above arguments of high expertise, effective controllability, good reputation, and lack of incentive are sound reasons to trust a TPE manufacturer to build reliable and powerful TPEs. The main advantage of the approach lies in the possibility to leverage this trust in the TPE manufacturer onto a completely different principal in the role of a service provider, which

- does not have the proper expertise to ensure a secure operation of its hardware and to guarantee the protection of the processed data.
- is quite difficult to control, due to the sheer number of such service providers.
- has no particular reputation (and therefore none to lose).
- might have short term goals that (in its point of view) justify a policy violation.

With the presented approach, such a service provider can easily define the policy rules that it would like its TPE to enforce (by selecting from the options offered by the TPE manufacturer) and buy the appropriate TPE from a reputable TPE manufacturer. The service provider can then immediately benefit from the trust that users have in the TPE manufacturer of its TPE to convince them that it will not maliciously abuse an agent sent by the users.

With this, the approach favours the open systems philosophy, where any principal can possibly become a provider of services. Such a service provider simply has to obtain a TPE from some reputable manufacturer and can then easily convince a client that the client's confidential information is sufficiently protected. Thus, it becomes much easier for a new service provider to establish itself in the market.

7 Conclusion

In this paper, we have discussed the notion of trust in the context of mobile agent systems and introduced a structuring for this problem domain. Starting from this structure, we have proposed an approach that relies on trusted and tamper-proof hardware, which allows to prevent malicious behaviour rather than correct it. We believe this to be the better form of protection for confidential data. We have shown how the approach can be used to effectively protect the itinerary of a mobile agent. Finally, we identified the positive implications that the presented approach can have on the construction of open mobile agent systems, where any principal can become a service provider and receive mobile agents.

In real-life, there are limitations to the approach. Given sufficient time and resources, a TPE operator might succeed in breaking the system and it would thus be possible for him to violate even those parts of the policy that should be enforced by the TPE. Our goal is to make this approach so costly that it would negate a possible gain (there may be many different implementations of TPEs that provide different security guarantees). As further deterrent, we assume that a non-repudiable proof for a policy violation of an enforced policy or of an attempted or successful breaking of a TPE might be punished much more severely than a mere policy violation since it proves a much larger determination to commit a criminal offence.

[10] There is the possibility that a TPE operator bribes a TPE manufacturer to provide an incorrect TPE. We assume that such a behaviour is a severe offence that is subject to criminal investigation and not within the scope of this discussion.

References

[1] R. Anderson and M. Kuhn. Tamper resistance — a cautionary note. In *The Second USENIX Workshop on Electronic Commerce Proceedings*, pages 1–11, Oakland, California, November 1996.

[2] G. Brassard. *Modern Cryptology – A Tutorial*, volume 325 of *Lecture Notes in Computer Science*. Springer Verlag, 1988.

[3] A. Carzaniga, G. P. Picco, and G. Vigna. Designing distributed applications with mobile code paradigms. In R.Taylor, editor, *Proceedings of the 19th International Conference on Software Engineering (ICSE'97)*, pages 22–32. ACM Press, 1997.

[4] D. M. Chess, B. Grosof, C. G. Harrison, D. Levine, C. Parris, and G. Tsudik. Itinerant agents for mobile computing. *IEEE Personal Communications*, 2(3):34–49, October 1995.

[5] W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Trans. Inform. Theory*, IT-22(6):644–654, 1976.

[6] DoD. Trusted Computer System Evaluation Criteria (TCSEC). Technical Report DoD 5200.28-STD, Department of Defense, December 1985.

[7] The Economist. The collapse of Barings, March 4 1995.

[8] J. Gosling and H. McGilton. The java language environment. White paper, Sun Microsystems, Inc., 1996.

[9] R.S. Gray. Agent Tcl: A transportable agent system. In *Proceedings of the CIKM Workshop on Intelligent Information Agents*, Baltimore, MD, December 1995.

[10] C. G. Harrison, D. M. Chess, and A. Kershenbaum. Mobile agents: Are they a good idea? In J. Vitek and C. Tschudin, editors, *Mobile Object Systems: Towards the Programmable Internet*, volume 1222 of *Lecture Notes on Computer Science*, pages 25–47. Springer, 1997. Also available as IBM Technical Report RC 19887.

[11] ITU. *ITU-T Recommendation X.509: The Directory – Authentication Framework*. International Telecommunication Union, 1993.

[12] J. Ordille. When agents roam, who can you trust? Technical Report Technical Report, Computing Science Research Center, Bell Labs, 1996.

[13] RSA Data Security, Inc. *PKCS #1: RSA Encryption Standard*. RSA Data Security, Inc., November 1993.

[14] R. A. Rueppel. A formal approach to security architectures. In *EuroCrypt*, pages 387–398, Brighton, England, 1991.

[15] B. Schneier. *Applied cryptography*. Wiley, New York, 1994.

[16] J. G. Steiner, C. Neuman, and J. I. Schiller. Kerberos: An authentication service for open network systems. In *Proceedings of the USENIX Winter 1988 Technical Conference*, pages 191–202. USENIX Association, Berkeley, USA, February 1988.

[17] New York Times. U.S. workers stole data on 11,000, agency says, April 6, 1996.

[18] G. Vigna. Protecting mobile agents through tracing. In *Proceedings of the Third Workshop on Mobile Object Systems*, Finland, June 1997.

[19] J. E. White. Telescript technology: The foundation for the electronic market place. White paper, General Magic, Inc., 1994.

[20] U. G. Wilhelm, L. Buttyàn, and S. Staamann. On the problem of trust in mobile agent systems. In *Symposium on Network and Distributed System Security*. Internet Society, March 1998. (to appear).

[21] I. S. Winkler. The non-technical threat to computing systems. *Computing Systems, USENIX Association*, 9(1):3–14, Winter 1996.

[22] T. Y. C. Woo and S. S. Lam. Authentication for distributed systems. *IEEE Computer*, 25(1):39–52, January 1992.

[23] P. Zimmermann. *PGP User's Guide*. MIT Press, Cambridge, 1994.