Authenticating Mobile Agent Platforms Using Signature Chaining Without Trusted Third Parties

Amitabh Saxena and Ben Soh

Dept. Of Computer Science and Computer Engineering
La Trobe University
Bundoora, VIC, Australia 3086
Email:{asaxena, ben}@cs.latrobe.edu.au

Abstract—Mobile agents are mobile programs capable of maintaining their execution state as they migrate across different execution platforms. Large-scale deployment of mobile agent based applications has not been possible because some key technical and security issues remain unsolved.

In this paper we propose a method to protect mobile agent platforms from malicious agents using authentication primitives. In contrast to approaches for host protection based on sandbox environments, our method aims to validate the itinerary of an agent. We use the concept of one-way signatures to connect arbitrary hosts in a chain of trust, thus enabling ad-hoc trust relationships to be formed. Our technique does not require the use of any trusted third parties or tamper-proof hardware solutions.

I. INTRODUCTION

Mobile agents are agents that can physically travel across networks and perform tasks on machines that provide agent hosting capability. This allows processes to migrate from computer to computer, for processes to split into multiple instances that execute on different machines, and to return to their point of origin. A detailed discussion of mobile agents is beyond the scope of this paper and the reader is referred to [1]. Mobile agents have applications in areas like intrusion detection, network management and e-commerce [2].

Although there are many benefits, widespread use of this technology is not possible unless some key security issues are resolved. Security related challenges for mobile agents are described in [3], [4]. Of the various problems considered, two are foremost:

(a) Protecting an agent from a malicious host platform
(b) Protecting a host platform from a malicious agent

In this paper we focus only on the latter problem of protecting host platforms. For a discussion on methods for agent protection, the reader is referred to [5], [6], [7], [8].

Our approach to security is based on a notion of trust which is summarized as follows: If all entities involved with the agent can be authenticated, a level of trust can be established, which can then be used for granting or denying execution privileges. Current solutions for host protection rely on tamper-proof hardware, trusted third parties or a ‘sandbox’ model of execution [9], [10], [11]. Our method does not require any such measures.

II. BACKGROUND

A. Notation

Any entity that runs a mobile agent platform server is called a host. We assume that all such hosts are identified by a public directory. Any host that initiated an agent into the system is called the initiator of the agent. Agents can migrate autonomously between different host platforms. This act of migration is called agent transfer. We assume that agent transfer is done over a secure channel using a standard agent transfer protocol. An instance of an agent is a snapshot of its state at any point of execution on some platform. An itinerary is the ordered list of hosts already visited by an agent.

B. Agent Partitioning

A mobile agent is a set of interacting mobile objects with agent functionality. Since objects encapsulate data and methods, we use this idea to split the agent into two parts: a static part (consisting of object methods) which is fairly unchanging as the agent hops across platforms and a dynamic part containing data and the state information of the interacting objects that changes at each hop. We shall call the general process of splitting an agent into such parts as Agent Partitioning.

Though the exact method of partitioning will depend on specific implementations, we describe a general method which is suitable for our purpose. The static part contains executable code incorporating methods of the objects. All information that is state-dependent is in the dynamic part. Due to this constraint (with some additional constraints on the implementation), it is possible to ensure that our partitioning is unique. That is each instance of the same agent will have the same static part.

We assume that the static and dynamic parts can be made mutually inseparable. This means that the agent’s functionality is available if and only if both the static and dynamic parts correspond to the same agent. Mixing and matching between different agents is not possible. We say that the static and dynamic parts of an agent are mutually authenticating whenever they possess this property.
C. Authentication Requirements

Our approach to security is based on a model of trust. Execution privileges are granted or denied based on an authentication mechanism. In the context of mobile agents, authentication comes in two flavors:

1) Initiator authentication: Is the claimed initiator the same as the real initiator?
2) Itinerary authentication: Is the claimed itinerary the same as the real itinerary?

Our requirement for unconditional security is itinerary authentication. It is evident, however, that this will also always involve initiator authentication, since the initiator is the first host in the itinerary.

In this paper, we demonstrate that itinerary authentication is theoretically achievable if initiator authentication is possible. We first show how an arbitrary host can authenticate the initiator of an agent before describing our concept of signature chaining to authenticate itineraries.

III. INITIATOR AUTHENTICATION

The concept of undetectable signature [5], [7] combined with the partitioning scheme discussed earlier will enable initiator authentication as follows:

Undetectable signatures allow an initiator to include a signing function as part of the agent that can be used to create valid signatures only on messages of a predetermined structure. Since a host cannot generate arbitrary message-signature pairs [8], the signing function authenticates the initiator at the same time. If we assume that the agent is coded in a sufficiently high level language, with the static part containing the signing function, it will be hard to reverse-engineer the code and separate the signing function from the rest of the code. As a consequence, the entire static part of the agent can be authenticated with the initiator. Once we assume that the static and dynamic parts of an agent are mutually authenticating (see section II-B), we can obtain initiator authentication corresponding to the agent. This process can be essentially thought of as inserting the initiator’s ‘watermark’ inside the agent. Techniques for code-obfuscation and watermarking to this end are discussed in [8], [13], [14], [15], [16].

IV. ITINERARY AUTHENTICATION

Our concept of itinerary authentication is based on the underlying assumption that the initiator can somehow be authenticated. We introduce the concept of relative authentication to imply that the first host (the initiator) in an itinerary is unknown. On the other hand, absolute authentication implies that the initiator can be authenticated.

Represent the host platforms as points of a acyclic directed graph. As the agent hops, a new arc directed from the receiver to the sender is added to the graph. The edges of such a graph will represent a hop-by-hop path of the agent in the reverse direction from the current host to the initiator. In this notation the statements “a passed the agent to b” and “There is a path of unit length from b to a” are considered equivalent. We can consider this graph to describe the path by which trust is propagated in the system.

1. We say that a direct path exists from b to a if and only if b can prove (in the context of the agent) something about a that no other host can. That is, b has some extra information about a that others cannot extract from b’s proof.
2. Let \{h_0, h_1, \ldots, h_n\} be a set of hosts for some \( n \geq 1 \). We say a chained path exists from \( h_0 \) to \( h_0 \) if and only if there exists a direct path from \( h_x \) to \( h_{x-1} \) for each \( x \) from 1 to \( n \).
3. We say that there is a one-way chained path from b to a if and only if there is a chained path from b to a and there is no (direct or chained) path from a to any other host.

Assume that \( i \) is the initiator of the agent, a is any sending host and b is the receiving host. Also, excepting the act of agent transfer no other interaction is allowed between any hosts. Using this scenario, itinerary authentication can be defined in the context of b as follows:

(a) Relative: If b can determine that a chained path from a to i exists.
(b) Absolute: It b can determine that a one-way chained path from a to i exists.

In definition 1 above, the idea of ‘extra’ information can be formalized using zero knowledge proofs. This is discussed in the next section.

V. DEFINITIONS

A. Zero Knowledge Proofs

A zero knowledge protocol enables a prover to convince a verifier, the validity of a statement without revealing any extra information about the statement itself. That is, after the execution of the protocol, the verifier is convinced that the statement is indeed true without getting any information whatsoever as to why it is true [17]. In the simplest form, a zero knowledge protocol is interactive. That is, it requires many interactions (in the form of challenge-response) between the prover and verifier. After sufficiently many successive correct responses, the verifier is convinced about the correctness of the statement being proved.

B. Non-Interactive Zero Knowledge

In a Non-Interactive Zero Knowledge (NIZK) proof system, any honest prover can convince a verifier about the correctness of some statement by attaching a ‘short’ proof. The zero knowledge property is obtained by using a common random reference string [18], [19].

\[ \text{We intuitively define trust to propagate in the reverse direction of the agent. If the agent moves from a to b, we are interested to know if b trusts a. That is, if there is path from b to a. Moreover we are only interested in those hosts that modified the dynamic part.} \]
C. Identity Based Zero Knowledge

Intuitively, an identity based proof can be considered as a proof of ‘knowledge of knowledge’. The basic idea is the same; the proof convinces the verifier in a non-interactive way, the validity of a statement. The statement in this case also includes an identity. Essentially, the statement s in the previous definition is modified to say “Alice knows the value of u”. To understand zero knowledge in this new scenario, observe that the proof only validates the correctness of s. It does not give any information about the prover’s identity. The random string need not be pre-shared since the proof is verifier independent.

D. Additive Zero Knowledge

In this work, we consider the possibility of NIZK proofs based on multiple identities and propose the idea of additive zero knowledge. The word ‘additive’ implies that a verifier can construct a new identity based NIZK proof from an old proof using the same random string. If this process can go on many times, the participants in the protocol are said to be connected in a chain of trust.

In this new scenario, Alice makes a NIZK proof pA of the statement sA = “Alice knows the value of u” and sends {sA, pA} to Bob. Using this, Bob creates a NIZK proof pB of the statement sB = “Bob knows that statement sA is true”. He keeps pA secret and sends {sB, sA, pB} to another recipient, Carol. This enables Carol to validate sB without giving her the ability to validate sA on its own. In other words, it is not possible to compute pA just from the information sent to Carol. Finally, Carol ‘adds’ to the proof by creating the statement sC = “Carol knows that the statement sB is true” and a string pC such that the values {sC, sB, sA, pC} constitute a NIZK proof of sC. It can be seen that an additive proof system enables trust to propagate between multiple provers.

E. Trust Relationships

The connections between various provers and verifiers can be represented using trust relationships. We define a trust relationship to be the mapping τ : S × Z → {0, 1} where S is a set of statements (which can be either true or false) and Z is a set of proofs such that for some s ∈ S and some z ∈ Z, the value of τ indicates by the usage:

\[ \tau(s, z) = 1 \iff \text{z is a valid proof of s.} \]
\[ \tau(s, z) = 0 \iff \text{z is not a valid proof of s.} \]

F. Fixed Strings

Let L1 and L2 be any two languages. For some x ∈ L1 and some y ∈ L2, the ordered pair (x, y) is said to be fixed if and only if there exists a (polynomial-time computable) binary function σ : L1 × L2 → {0, 1} such that σ(x, y) = 1 and it is computationally intractable to find another string ŷ ∈ L2 such that σ(x, ŷ) = 1.

VI. OUR AUTHENTICATION PROTOCOL

We use the following notation in this section: Let P0 be the originator of the message. Denote by Pi the ith user who passed the message. We define the message passed from Pi to Pi+1 as the ordered pair (Mi, Di) where Mi is the static part and Di is the dynamic part. We assume that the static and dynamic parts are mutually authenticating. Since the dynamic part always changes, it is only validated up to the previous node. The path authentication is done entirely using the static part. Denote by Mi and Di, the signatures of Pi on M and Di respectively using any standard signature scheme.

A. Agent Initiation

For each message M the originator P0 generates a random secret authenticating string u and computes the corresponding public verification string v = f(u, M) where f is a one-way function such that it is it is computationally infeasible to find a solution for u just from v and M. To sign the message P0 creates the statement s0 and a zero knowledge proof z0 where:

1) s0 = “P0 KNOWS u CORRESPONDING TO (v, M).”
2) τ(s0, z0) = 1 if and only if s0 is true.

The signature of P0 on (M, Di) is \{s0, z0, v, M, Di\}. On receiving the message Pi follows the verification procedure which we describe next.

B. Agent Authentication

An arbitrary platform Pi (i > 0) will process the message as follows: On receiving the message from Pi−1, it first follows the verification procedure. Before passing on the message to Pi+1, it follows the signing procedure. We assume that the verification key v must always accompany the message.

Signing

To sign the message each Pi (i > 0) creates a statement si and a zero knowledge proof zi such that:
1) si = “Pi KNOWS STATEMENT si−1 IS TRUE.”
2) zi = g(zi−1, M) where g is a one-way function such that it is computationally infeasible to find a solution for zi−1 just from zi and M.
3) τ(s, z) = 1 \iff z is a valid proof of s.
4) τ(s, z) = 0 \iff z is not a valid proof of s.

The signature of any Pi on its message (Mi, Di) is the set Xi = \{s0, s1, ..., si, z, ν, {M0, M1, ..., Mi}, Di\}. The string “P0, P1, ..., Pi” with the list of names is also assumed to be part of the signature. Pi keeps zi−1 secret as evidence that the message was indeed received from Pi−1. If the the above properties are satisfied, we say that each Xi is a chained signature on M.
Verifications

For clarity, we describe the verification procedure to be followed by $P_{i+1}$. It consists of four tests and the message is rejected if any of them fail:

1) Verify $D_i$, the signature of $P_i$ on the dynamic part $D_i$.
2) Verify $M_j$, the signature of $P_j$ on the static part $M$ for each $j$ from 0 to $i$.
3) Verify that the static and dynamic parts belong to the same message.
4) Verify that $\tau(s_i, z_i) = 1$.

C. Analysis Of The Protocol

The above protocol is an example of an additive proof system since each $P_{i+1}$ can add more information to $z_i$ by computing $z_{i+1}$. Note that authentication is only relative since any other host could claim to be the initiator by generating a different verification string $\hat{v}$ corresponding to $M$ and there would be no way to tell which string is authentic. To enable absolute authentication, we require that at least one of $(M, P_0)$ or $(M, v)$ be fixed.

A possible approach for this is to involve a Trusted Third Party (TTP) to certify one of the pairs. The TTP ensures that the same pair cannot be reused again for a certain period of time. We note, however, that $(M, P_0)$ can be fixed without involving a TTP. This is done using the methods for initiator authentication described in section III.

For this section, we assume that $(M, P_0)$ can somehow be fixed. Due to this, the first name in the list necessarily has to be $P_0$. Since $y$ is a one-way function, computing any $z_i$ just from $z_i$ is considered infeasible. Also since $\tau(s_j, z_i) = 0$ if $j < i$, we see that users cannot delete or change order of names. It is clear that each $P_i$ must add its name before passing the message otherwise step 1 of the verification process will fail. Finally, arbitrary names cannot be added because it is not possible to compute signatures of other users required in step 2 of the verification process. In summary, the above protocol authenticates the itinerary.

We note that the size of signatures and the verification time increase linearly with the number of users. This cannot be avoided since the information contained in the signatures also increases linearly.

VII. Conclusion

In this paper, we proposed the concept of signature chaining and presented a protocol for mobile agent authentication. Our method is based on the notion of additive zero knowledge which enables trust to propagate between different provers. We demonstrated that signature chaining can be used to form ad-hoc trust relationships between multiple participants in a dynamic and non-interactive manner. A possible construction for our scheme based on cryptographic primitives is presented in [20]

We conclude the paper with an open question: Is it possible to construct chained signatures using existing methods?

REFERENCES