Deutsches Forschungszentrum für Künstliche Intelligenz GmbH

Research Report
RR-02-03

Proceedings of the 2nd International Workshop on Security in Mobile Multiagent Systems

associated to AAMAS-2002

Bologna, Italy

Klaus Fischer and Dieter Hutter (Eds.)

16. July 2002

Deutsches Forschungszentrum für Künstliche Intelligenz GmbH

Postfach 20 80
67608 Kaiserslautern, FRG
Tel.: + 49 (631) 205-3211
Fax: + 49 (631) 205-3210
E-Mail: info@dfki.uni-kl.de

Stuhlsatzenhausweg 3
66123 Saarbrücken, FRG
Tel.: + 49 (681) 302-5252
Fax: + 49 (681) 302-5341
E-Mail: info@dfki.de

WWW: http://www.dfki.de

Deutsches Forschungszentrum für Künstliche Intelligenz

DFKI GmbH

German Research Center for Artificial Intelligence

Founded in 1988, DFKI today is one of the largest nonprofit contract research institutes in the field of innovative software technology based on Artificial Intelligence (AI) methods. DFKI is focusing on the complete cycle of innovation — from world-class basic research and technology development through leading-edge demonstrators and prototypes to product functions and commercialization.

Based in Kaiserslautern and Saarbrücken, the German Research Center for Artificial Intelligence ranks among the important “Centers of Excellence” worldwide.

An important element of DFKI’s mission is to move innovations as quickly as possible from the lab into the marketplace. Only by maintaining research projects at the forefront of science can DFKI have the strength to meet its technology transfer goals.

DFKI has about 165 full-time employees, including 141 research scientists with advanced degrees. There are also around 95 part-time research assistants.

Revenues for DFKI were about 30 million DM in 2000, half from government contract work and half from commercial clients. The annual increase in contracts from commercial clients was greater than 20% during the last three years.

At DFKI, all work is organized in the form of clearly focused research or development projects with planned deliverables, various milestones, and a duration from several months up to three years.

DFKI benefits from interaction with the faculty of the Universities of Saarbrücken and Kaiserslautern and in turn provides opportunities for research and Ph.D. thesis supervision to students from these universities, which have an outstanding reputation in Computer Science.

The key directors of DFKI are Prof. Wolfgang Wahlster (CEO) and Dr. Walter Olthoff (CFO).

DFKI’s five research departments are directed by internationally recognized research scientists:

- Knowledge Management (Director: Prof. A. Dengel)
- Intelligent Visualization and Simulation Systems (Director: Prof. H. Hagen)
- Deduction and Multiagent Systems (Director: Prof. J. Siekmann)
- Language Technology (Director: Prof. H. Uszkoreit)
- Intelligent User Interfaces (Director: Prof. W. Wahlster)

In this series, DFKI publishes research reports, technical memos, documents (eg. workshop proceedings), and final project reports. The aim is to make new results, ideas, and software available as quickly as possible.

Prof. Wolfgang Wahlster
Director

Proceedings of the 2nd International Workshop on Security in Mobile Multiagent Systems

Klaus Fischer and Dieter Hutter (Eds.)

DFKI-RR-02-03

This work has been supported by a grant from The Federal Ministry of Education, Science, Research, and Technology (FKZ ITW-5064).

ISSN 0946-008X

Proceedings of the 2^{nd} International Workshop on Security in Mobile Multiagent Systems

Klaus Fischer and Dieter Hutter (Eds.)

June 27, 2002

Preface

This report contains the Proceedings of the Second Workshop on Security of Mobile Multiagent Systems (SEMAS'2002). The Workshop was held in Bologna, Italy on July 16, 2002, as a satellite event to the 1st International Conference on Autonomous Agents and Multiagent Systems 2002. The First Workshop on Security of Mobile Multiagent Systems (SEMAS'2001) was held in Montreal, Canada as a satellite event to the 5th International Conference on Autonomous Agents in 2001.

The far reaching influence of the Internet has resulted in an increased interest in agent technologies, which are poised to play a key role in the implementation of successful Internet and WWW-based applications in the future. While there is still considerable hype concerning agent technologies, there is also an increasing awareness of the problems involved. In particular, that these applications will not be successful unless security issues can be adequately handled. Although there is a large body of work on cryptographic techniques that provide basic building-blocks to solve specific security problems, relatively little work has been done in investigating security in the multiagent system context. Related problems are secure communication between agents, implementation of trust models/authentication procedures or even reflections of agents on security mechanisms. The introduction of mobile software agents significantly increases the risks involved in Internet and WWW-based applications. For example, if we allow agents to enter our hosts or private networks, we must offer the agents a platform so that they can execute correctly but at the same time ensure that they will not have deleterious effects on our hosts or any other agents / processes in our network. If we send out mobile agents, we should also be able to provide guarantees about specific aspects of their behaviour, i.e., we are not only interested in whether the agents carry-out their intended task correctly. They must defend themselves against attacks initiated by other agents, and survive in potentially malicious environments.

Agent technologies can also be used to support network security. For example in the context of intrusion detection, intelligent guardian agents may be used to implement active protection strategies on a firewall or intelligent monitoring agents can be used to analyse the behaviour of agents migrating through a network. Part of the inspiration for such multi-agent systems comes from primitive animal behaviour, such as that of guardian ants protecting their hill or from biological immune systems.

Progam Committee

The papers in this report were reviewed by the program committee consisting, besides the workshop chairs, of

- Sahin Albayrak, TU Berlin, Germany
- David Basin, University of Freiburg, Germany
- Hans-Juergen Buerckert, DFKI GmbH Saarbruecken, Germany
- Ciaran Bryce, University of Geneve, Switzerland
- Giuseppe Castagna, Ecole Normale Superieure, France
- Luc Moreau, University of Southampton, UK
- Stefan Poslad, Queen Mary University of London, UK
- Volker Roth, Fraunhofer IGD, Germany
- Helmut Schwigon, BSI Bonn, Germany
- Vipin Swarup, The MITRE Corp, USA
- Christian Tschudin, Uppsala University, Sweden

The Workshop Chairs

Klaus Fischer and Dieter Hutter

Contents

Modal Specifications of Trace-Based Security Properties

Rafael Accorsi David Basin Luca Viganò
Institut für Informatik, Albert-Ludwigs-Universität Freiburg
Georges-Köhler-Allee 52, D-79110 Freiburg, Germany
{accorsi,basin,luca}@informatik.uni-freiburg.de

ABSTRACT

We introduce a multi-modal logic that combines complementary features of authentication logics and trace-based approaches. Our logic contains two kinds of modalities: implicit belief, which formalizes the view of an external agent reasoning about interleaved protocol executions, and explicit belief, which uses awareness to model the resource-bounded reasoning of the agents involved in the executions. We employ these modalities to formalize extensional and intensional specifications of protocols and their properties, and use these formalizations to characterize and reason about attacks. As an example, we consider the Needham-Schroeder Public Key protocol and use our logic to demonstrate the existence of the well-known man-in-the-middle attack, and also show the equivalence of our modal specification to one based on an interleaved trace semantics.

1. INTRODUCTION

Security protocols describe how agents should exchange messages to achieve security goals such as confidentiality and integrity of data, or authentication of the identity of agents in a network. A number of approaches have been proposed for rigorously analyzing security protocols. Some of these are based on specialized security logics, such as the foundational BAN logic for authentication protocols [6] and its extensions, e.g. [1, 4, 7, 13, 20, 21]. These logics work by formalizing the doxastic or epistemic reasoning of agents executing a protocol, and security properties are formalized and reasoned about in terms of the way the beliefs or the knowledge of the agents evolve as messages are exchanged. Although effective for finding some kinds of flaws, the logics' semantics are often lacking or restrictive (e.g. the logics are designed to reason about a *single* protocol execution).

An alternative way of reasoning about security protocols is to consider protocols as sets of possibly interleaved communication traces. For example, given a protocol and an attacker model, Paulson [18] turns these into an inductive definition (of the trace set) in higher-order logic. The resulting set can be used to inductively establish security properties by showing that they hold for all traces. Similar inductive definitions are used by Basin in [3] to provide a basis for finding attacks (traces violating security properties) using infinite-state model-checking. The strengths and weaknesses of trace-based methods are in some sense complementary to security logics. Although trace-based methods provide a simple and expressive theory for formalizing the semantics of protocols and security properties in terms of interleaved executions, characterizing attacks as properties of traces can be tricky, whereas BAN-like specifications are generally simpler and more abstract.

In this paper, we introduce a multi-modal logic that combines complementary features of authentication logics and trace-based approaches. Our account is semantic: traces are used to build a Kripke structure upon which a modal logic is defined to reason about actions occurring in interleaved protocol executions. The modalities are used to formalize the *implicit* and *explicit beliefs* of agents, allowing modal specifications of security properties while being based on an underlying interleaved trace semantics. The two belief modalities give us considerable flexibility in our specifications. Roughly speaking, using the implicit belief modality we can model what agents would believe had they seen all messages exchanged by all agents, and using explicit belief we can model what agents believe based on what they have actually seen. We define implicit belief as the standard modality of belief logics [14]. To formalize explicit belief and express the local reasoning of an agent based on the actions he has participated in, we adapt the notion of *awareness* (introduced in logics for artificial intelligence [10, 11, 22] to address the problem of agents having unbounded reasoning power and thus being logically omniscient, which is a characteristic of most doxastic/epistemic logics).

We present two applications of our logic. First, we apply it to formally characterize different kinds of specifications of security properties. It has been observed that specifications are generally either *intensional*, i.e. based on details of the protocol steps, or *extensional*, i.e. formulated independently of message exchanges. We use the explicit belief modality to characterize intensional specifications, and the implicit belief modality to characterize extensional specifications.

Second, we show how to use these modalities to characterize and reason about attacks in interleaved protocol executions. Our specifications of security properties combine intensional and extensional specifications: the intensional part is used to represent the completion (or commitment) of agents in protocol executions, and the extensional part formalizes properties such as message secrecy. We illustrate this using the Needham-Schroeder Public Key protocol as a running example and show how the semantics can be used to demonstrate the existence of attacks. Afterwards we show the equivalence of our modal specification to those based on interleaved trace semantics.

We proceed following the structure given above. §2 gives the semantic foundations of our logic, and §3 and §4 discuss the two applications. We compare with related work in §5 (as explained there, this work supersedes our previous work on awareness-based security logics [2]), and conclude in §6.

2. A MULTI-MODAL SECURITY LOGIC

2.1 Syntax

We start by defining the set of messages, which are built from primitive terms by pairing and encryption. Based on this, we define a multi-modal language extended with operators expressing, e.g., possession and secrecy of messages.

Definition 1 Let the set $\mathcal{T}$ of *primitive terms* consist of three disjoint subsets: $\mathcal{T}_I$ of *agent identifiers*, $\mathcal{T}_K$ of *cryptographic keys*[1], and $\mathcal{T}_N$ of *nonces*. The set $\mathcal{M}$ of *messages* is the smallest set closed under the following rules: (i) $M \in \mathcal{M}$ if $M \in \mathcal{T}$; (ii) $M \circ M' \in \mathcal{M}$ if $M, M' \in \mathcal{M}$; and (iii) $\{\!|M|\!\}_K \in \mathcal{M}$ if $M \in \mathcal{M}$ and $K \in \mathcal{T}_K$.

The set $\mathcal{F}$ of *formulas* is the smallest set closed under the following rules: (i) $\bot \in \mathcal{F}$; (ii) $\varphi \to \psi \in \mathcal{F}$ if $\varphi \in \mathcal{F}$ and $\psi \in \mathcal{F}$; (iii) $\mathsf{says}_A(B, M)$, $\mathsf{sees}_A(M)$, $\mathsf{has}_A(M)$, $\mathsf{sec}_{\mathcal{G}}(M)$, $\mathsf{comml}_A(B, M)$, $\mathsf{commR}_A(B, M) \in \mathcal{F}$ if $A, B \in \mathcal{T}_I$, $\mathcal{G} \subseteq \mathcal{T}_I$, and $M \in \mathcal{M}$; (iv) $\mathcal{X}_A\,\varphi$, $\mathcal{B}_A\,\varphi \in \mathcal{F}$ if $A \in \mathcal{T}_I$ and $\varphi \in \mathcal{F}$. ■

The formulas express properties of message exchanges between the agents engaged in a protocol execution (also called *run*). Intuitively, the formula $\mathsf{says}_A(B, M)$ denotes agent A *saying* M to B, $\mathsf{sees}_A(M)$ denotes A *seeing* M, $\mathsf{has}_A(M)$ denotes A *possessing* M, and $\mathsf{sec}_{\mathcal{G}}(M)$ denotes that M is a *secret* possessed only by the agents in the *group* $\mathcal{G}$. The formula $\mathsf{comml}_A(B, M)$ (respectively, $\mathsf{commR}_A(B, M)$) expresses that agent A uses message M to *commit* as the *initiator* (respectively, *responder*) in a protocol execution with agent B.[2] The modalities $\mathcal{X}_A$ and $\mathcal{B}_A$ denote the *explicit belief* and the *implicit belief* of an agent A. Other connectives and modalities are defined in the usual manner, e.g. negation $\neg\varphi \equiv \varphi \to \bot$ and conjunction $(\varphi \wedge \psi) \equiv \neg(\varphi \to \neg\psi)$.

To distinguish between the variables appearing in a protocol description and the actual values with which these variables are instantiated in a protocol execution, variables ranging over agent identifiers are denoted by capital letters $A, B, C, \ldots$, and the concrete values by lowercase letters $a, b, c, \ldots$, where the special constant *spy* denotes the *attacker*. We use the same convention also for keys and nonces, and write K and k, and N and n. We write $\mathcal{G}$ to denote a group of agents, α to denote an awareness set, and φ and ψ to denote formulas.

[1] We assume an underlying algebra where $(K^{-1})^{-1} = K$ for all keys $K \in \mathcal{T}_K$, and the function $\cdot^{-1} : \mathcal{T}_K \to \mathcal{T}_K$ maps a key K to its inverse key K^{-1}. For protocols employing (symmetric) shared keys we also have $K^{-1} = K$.

[2] The two predicates are used to express commitment of agents executing protocols with two roles, initiator and responder. It is straightforward to generalize the syntax and subsequent semantics with families of predicates like $\mathsf{comm}^k_j(A_1, \ldots, A_k, M)$, which formalizes commitment for the agent in the j-th role for a protocol with k roles.

2.2 Model of computation

Our model of computation combines ideas from trace-based methods for protocol verification [3, 18] with ideas from authentication logics [1, 6] and from approaches to reasoning about knowledge in multi-agent systems [10, 11, 22].

Trace-based foundations

An *event* e is a message exchange of the form $A \to B : M$ and a *trace* is a sequence $e_1, \ldots, e_k$ of *events* (where $\langle\rangle$ denotes the empty trace). A protocol is modeled as a set of traces; namely, the smallest set of traces closed under rules that formalize the effects of protocol steps and the possible actions by an attacker.

In Fig. 1 we show the NSPK protocol and its definition as an inductively defined set P of traces. The rules nspk_1, nspk_2, and nspk_3 formalize the three protocol steps. For example, nspk_2 models the second step of the protocol and says that a trace $t \in P$ can be extended with $B \to A : \{\!|N_A \circ N_B|\!\}_{K_A}$ whenever N_B has not been used in t (i.e. it is a *fresh* nonce) and t contains an event $A' \to B : \{\!|A \circ N_A|\!\}_{K_B}$. The attacker rule formalizes the attacker model of Dolev and Yao [9]: the *spy* can say anything that he can synthesize from the analyzable parts of the messages he spies, where $\mathsf{spies}(t)$ is the set consisting of all messages that have been sent in a trace t (which formalizes the assumption that the attacker has control over the network). The attacker rule uses the auxiliary functions synth and analz, which we now define (along with parts) as they will be needed in our model.

Definition 2 Let $\mathbf{M}$ be a set of messages. Using the corresponding rules in Fig. 2, we build the following three sets: the set $\mathsf{parts}(\mathbf{M})$ is the smallest extension of $\mathbf{M}$ obtained by adding the components of compound messages and the bodies of encrypted messages; the set $\mathsf{analz}(\mathbf{M})$ is the smallest extension of $\mathbf{M}$ closed under projection and decryption by keys in $\mathsf{analz}(\mathbf{M})$; and the set $\mathsf{synth}(\mathbf{M})$ is the smallest extension of $\mathbf{M}$ closed under pairing and encryption. ■

Modal foundations

The *local state* of an agent $A \in \mathcal{T}_I$ is a pair consisting of the set of actions that A has performed and the set of messages in A's possession. A *global state* w is an n-tuple of local states, where n is the number of agents in the system, including the attacker. In our model, the *actions* that an agent A can perform are *sending* a message M to another agent B, in symbols $\mathtt{send}_A(B, M)$, and *receiving* a message M, in symbols $\mathtt{rec}_A(M)$, where the identity of the sending agent is not known a priori.

We combine the notions of trace and state by defining functions that, given a trace, compute the local state of each agent participating in the (possibly partial, interleaved) protocol executions in the trace.

Definition 3 Given $t \in P$, the sets of actions and possessions of an agent A are defined by the functions $Ac_A(t)$ and

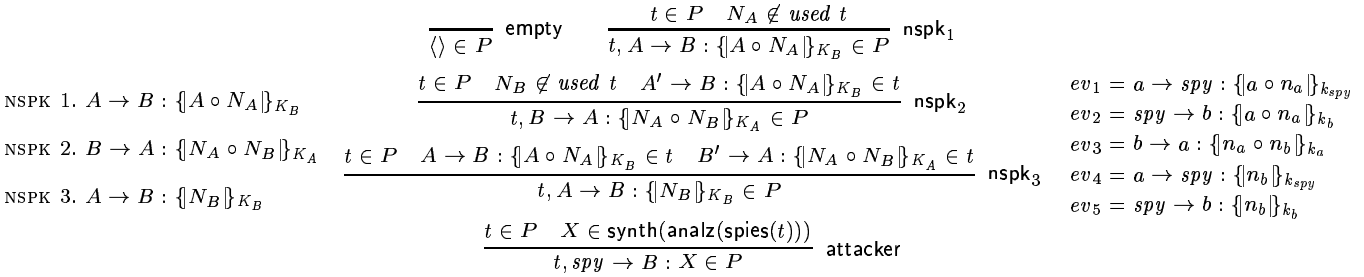

Figure 1: The NSPK protocol (L), the rules defining it inductively (C), and the MITM attack on it (R)

$$\frac{M \in \mathbf{M}}{M \in \mathsf{parts}(\mathbf{M})}\ \mathsf{parts\text{-}inj} \qquad \frac{M_1 \circ M_2 \in \mathsf{parts}(\mathbf{M})}{M_i \in \mathsf{parts}(\mathbf{M})}\ \mathsf{parts}\text{-}i\ (i \in \{1,2\}) \qquad \frac{\{|M|\}_K \in \mathsf{parts}(\mathbf{M})}{M \in \mathsf{parts}(\mathbf{M})}\ \mathsf{parts\text{-}body}$$

$$\frac{M \in \mathbf{M}}{M \in \mathsf{analz}(\mathbf{M})}\ \mathsf{analz\text{-}inj} \qquad \frac{M_1 \circ M_2 \in \mathsf{analz}(\mathbf{M})}{M_i \in \mathsf{analz}(\mathbf{M})}\ \mathsf{analz}\text{-}i\ (i \in \{1,2\}) \qquad \frac{\{|M|\}_K \in \mathsf{analz}(\mathbf{M}) \quad K^{-1} \in \mathsf{analz}(\mathbf{M})}{M \in \mathsf{analz}(\mathbf{M})}\ \mathsf{analz\text{-}dec}$$

$$\frac{M \in \mathbf{M}}{M \in \mathsf{synth}(\mathbf{M})}\ \mathsf{synth\text{-}inj} \qquad \frac{M_1 \in \mathsf{synth}(\mathbf{M}) \quad M_2 \in \mathsf{synth}(\mathbf{M})}{M_1 \circ M_2 \in \mathsf{synth}(\mathbf{M})}\ \mathsf{synth\text{-}pair} \qquad \frac{M \in \mathsf{synth}(\mathbf{M}) \quad K \in \mathsf{synth}(\mathbf{M})}{\{|M|\}_K \in \mathsf{synth}(\mathbf{M})}\ \mathsf{synth\text{-}enc}$$

Figure 2: The rules defining the sets parts, analz and synth

$Po_A(t)$ as follows: $Ac_A(\langle\rangle) = \emptyset$ and $Ac_A(B \to C : M, ts)$ is

$$\begin{cases} \{\mathtt{send}_B(C, M)\} \cup Ac_A(ts) & \text{if } A = B \\ \{\mathtt{rec}_C(M)\} \cup Ac_A(ts) & \text{if } A = C \\ \{\mathtt{send}_B(C, M), \mathtt{rec}_C(M)\} \cup Ac_A(ts) & \text{if } A = spy \\ Ac_A(ts) & \text{otherwise} \end{cases}$$

and $Po_A(\langle\rangle) = initState(A)$ and $Po_A(B \to C : M, ts)$ is $\{M\} \cup Po_A(ts)$ if $A \in \{B, C, spy\}$ and $Po_A(ts)$ otherwise, where ts ranges over event sequences and *initState* is a protocol-dependent function returning the message items that an agent initially possesses (e.g. his private and public keys, and the public keys and identifiers of other agents).

Thus, given a trace $t \in P$, the local state $s_A(t)$ of an agent A is simply $\langle Ac_A(t), Po_A(t)\rangle$, and the global state w is the n-tuple of the local states $s_A(t)$ for all n agents. Given a global state w, we will (overloading notation) write $s_A(w)$ to denote the local state of an agent A at w, and $Ac_A(w)$ and $Po_A(w)$ to denote the two components of $s_A(w)$. ■

Hence, the *spy*'s local state contains the actions performed by all the agents, as well as the messages they exchange, while the local state $s_A(w)$ of an agent A different from the *spy* is built only from the events that A participated in. Since the *spy* possesses all the messages sent in the network, $Po_{spy}(w)$ captures the same information as the set spies used in Fig. 1 to formalize the attacker's control over the network.

Let t be a trace and ts be the sequence of all prefixes of t. The set W^t of global states (or *worlds*) relative to t is obtained by computing, for each prefix of t, the corresponding sets of actions and possessions for all agents A. Formally, $W^t = wrl(ts)$, where

$$wrl(ts) = \begin{cases} \{(Ac_A(t'), Po_A(t'))\} \cup wrl(ts') & \text{if } ts = t', ts' \\ \{(Ac_A(\langle\rangle), Po_A(\langle\rangle))\} & \text{if } ts = \langle\rangle \end{cases}$$

Modeling resource-bounded agents

In the artificial intelligence literature, resource-bounded agents have limited computational resources, such as memory or time. In our approach, we model resource-bounded agents where the limitations are both in (1) the propositions an agent may reason about (his language) and (2) his deductive ability (what he can conclude). As an example of (1), if a nonce N is a secret between A and B, then another agent C should not even be able to formulate propositions about it. As an example of (2), when C learns the nonce N, he can then conclude that he possesses it, but not necessarily that some other agent D possesses it (even when this is the case).

Our first step in limiting resources is to restrict an agent's language by making the messages he can construct a function of the information he possesses at a state.

Definition 4 The set $\mathcal{M}_A(w)$ of messages that an agent A can construct at a global state w is defined as $\mathcal{M}_A(w) = \{M \mid M \in \mathsf{synth}(\mathsf{analz}(Po_A(w)))\}$. The set $\mathcal{F}_A(w)$ of formulas of an agent A at a global state w is the smallest set of formulas closed under the following rules: (i) $\bot \in \mathcal{F}_A(w)$; (ii) $\varphi \to \psi \in \mathcal{F}_A(w)$ if $\varphi, \psi \in \mathcal{F}_A(w)$; (iii) $\mathsf{says}_B(C, M)$, $\mathsf{sees}_B(M)$, $\mathsf{has}_B(M)$, $\mathsf{sec}_{\mathcal{G}}(M)$, $\mathsf{comml}_B(C, M)$, $\mathsf{commR}_B(C, M) \in \mathcal{F}_A(w)$ if $B, C \in \mathcal{M}_A(w) \cap \mathcal{T}_I$, $M \in \mathcal{M}_A(w)$, and $\mathcal{G} \subseteq \mathcal{M}_A(w) \cap \mathcal{T}_I$; and (iv) $\mathcal{X}_A\,\varphi \in \mathcal{F}_A(w)$ if $\varphi \in \mathcal{F}_A(w)$. ■

Clause (iii) expresses that each agent has its own language for the predicates says, sees, has, sec, comml, and commR, which depends on the messages that an agent possesses at some state w. In comparison with rule (iii) in Def. 1, here we simply require that the message items belong to the set of messages of the agent. For example, $\mathsf{says}_A(B, M) \in \mathcal{F}_A(w)$ if A and B are agent identifiers in $\mathcal{M}_A(w)$ (denoted by

$A, B \in \mathcal{M}_A(w) \cap \mathcal{T}_I$) and M is a message in the set of messages of A (in symbols $M \in \mathcal{M}_A(w)$).

With respect to (iv), note that since the agents' languages do not include the modality $\mathcal{B}$ for implicit belief, an agent can reason about neither his own nor other agents' implicit beliefs, nor can he have explicit beliefs about the explicit beliefs of other agents (as is standard in belief logics, e.g. [15]).

2.3 Semantics

We begin by fixing a set $\overline{\mathcal{T}}_I$ of *agent names*, where, for notational simplicity, we identify its elements with the previously defined set $\mathcal{T}_I$ of agent identifiers; thus, from now on we will simply talk of agents. Similarly, for keys and nonces.

Given a trace $t \in P$, we obtain the corresponding model $\mathfrak{M}^t = (W^t, \sim, \alpha)$, where W^t is a non-empty set of worlds, $\sim$ is an agent-indexed family of equivalence relations on W^t, and α is an agent-indexed family of awareness sets, where the set $\alpha_A(w)$ consists of the formulas that agent A is aware of at world w. The family of equivalence relations $\sim$ captures *indistinguishability*: two global states are indistinguishable to an agent A iff the local state of A is the same at these two global states. Formally, $w \sim_A w'$ iff $s_A(w) = s_A(w')$, i.e. $Ac_A(w) = Ac_A(w')$ and $Po_A(w) = Po_A(w')$. Note that our model does not contain a valuation function as we do not have propositional symbols.

A protocol execution results from agents taking actions and corresponds to a multi-agent system. We can view this system from two perspectives: that of an external agent who observes the system from the outside and does not interact with the agents executing the protocol, and that of the internal agents engaged in the execution. The former view is formalized using a *global truth relation*, denoted by $\models$. The latter is formalized by a *local truth relation*, which is a family of truth relations $\models_A$, indexed by agents A.

2.3.1 Global truth

The global truth relation formalizes what an external observer can conclude from the system. By design, this agent is not resource-bounded and has access to all communication and can reason about the local states of individual agents. In particular, he ascribes implicit belief to the agents, i.e. he can compute whether an agent A would implicitly believe in some formula φ, had A enough information about the overall communication that is taking place.

In order to formalize these ideas, and to define the semantics for predicates such as says and sees, we need to express specific relationships between agents and messages at a global state. For example, an agent should only be entitled to say the messages he is able to compose from the information he possesses. Similarly, he should be entitled to see the submessages that he can obtain from a message he receives. To this end, we introduce the operators comp and submsg to define two abbreviations that will be useful in the semantic definitions in §2.3; assuming that M is a message at w, the set $\mathsf{comp}_A(w, M)$ contains all the sub-messages that A used to construct the message M at w, i.e. $\mathsf{comp}_A(w, M) =$

$$\begin{cases} Po_A(w) \cap \mathsf{parts}(\{M\}) & \text{if } M \in \mathcal{M}_A(w) \\ \emptyset & \text{otherwise} \end{cases},$$

and the set $\mathsf{submsg}_A(w, M)$ consists of all sub-messages that A can obtain from M given the keys he possesses at w, i.e.

$$\mathsf{submsg}_A(w, M) = \mathsf{analz}(Po_A(w)) \cap \mathsf{parts}(\{M\}).$$

We use *commit sets* $\mathcal{C}$ to define the semantics of the commI and commR formulas. During one execution of a protocol *Prot*, an agent A can take either the initiator role or the responder role. Intuitively, within an execution, each role is identified by some message M, where M is, or contains, a nonce. The set $\mathcal{C}_I^{Prot}(A, B, M)$ contains the actions that A performed in order to commit as initiator to a responder B using message M. Similarly, $\mathcal{C}_R^{Prot}(A, B, M)$ contains the actions that B performed in order to commit as responder to an initiator A using message M. Both sets are obtained directly from the description of the protocol. We illustrate this by means of our running example.

Example 5 In an execution of the NSPK protocol (Fig. 1), the initiator A commits to the responder B using N_A after performing the actions corresponding to the steps encoded by the rules nspk_1, nspk_2 and nspk_3. Hence, the commit set $\mathcal{C}_I^{\text{NSPK}}(A, B, N_A) = \{\mathtt{send}_A(B, \{\!|A \circ N_A|\!\}_{K_B}), \mathtt{rec}_A(\{\!|N_A \circ N_B|\!\}_{K_A}), \mathtt{send}_A(B, \{\!|N_B|\!\}_{K_B})\}$. Similarly, the responder's "view" is formalized by the set $\mathcal{C}_R^{\text{NSPK}}(A, B, N_B) = \{\mathtt{rec}_B(\{\!|A \circ N_A|\!\}_{K_B}), \mathtt{send}_B(A, \{\!|N_A \circ N_B|\!\}_{K_A}), \mathtt{rec}_B(\{\!|N_B|\!\}_{K_B})\}$. ■

We are now ready to define the global truth relation.

Definition 6 The truth of a formula φ at a global state w in a model $\mathfrak{M} = (W, \sim, \alpha)$, in symbols $\mathfrak{M}, w \models \varphi$, is the smallest relation satisfying:

$\mathfrak{M}, w \models \mathsf{says}_A(B, M)$	if	$\mathtt{send}_A(B, M') \in Ac_A(w)$ and $M \in \mathsf{comp}_A(w, M')$ for some M'
$\mathfrak{M}, w \models \mathsf{sees}_A(M)$	if	$\mathtt{rec}_A(M') \in Ac_A(w)$ and $M \in \mathsf{submsg}_A(w, M')$ for some M'
$\mathfrak{M}, w \models \mathsf{has}_A(M)$	if	$M \in \mathsf{analz}(Po_A(w))$
$\mathfrak{M}, w \models \mathsf{sec}_{\mathcal{G}}(M)$	if	$\mathfrak{M}, w \models \mathsf{has}_A(M)$ for all $A \in \mathcal{G}$ and $\mathfrak{M}, w \not\models \mathsf{has}_B(M)$ for all $B \notin \mathcal{G}$
$\mathfrak{M}, w \models \mathsf{commI}_A(B, M)$	if	$\mathcal{C}_I^{Prot}(A, B, M) \subseteq Ac_A(w)$
$\mathfrak{M}, w \models \mathsf{commR}_A(B, M)$	if	$\mathcal{C}_R^{Prot}(B, A, M) \subseteq Ac_A(w)$
$\mathfrak{M}, w \models \varphi \to \psi$	if	$\mathfrak{M}, w \not\models \varphi$ or $\mathfrak{M}, w \models \psi$
$\mathfrak{M}, w \models \mathcal{B}_A \varphi$	if	$\mathfrak{M}, w' \models \varphi$ for all w' such that $w \sim_A w'$
$\mathfrak{M}, w \models \mathcal{X}_A \varphi$	if	$\mathfrak{M}, w \models_A \varphi$ and $\varphi \in \mathcal{F}_A(w)$

We write $\mathfrak{M} \models \varphi$ iff $\mathfrak{M}, w \models \varphi$ for all $w \in W$. ■

In other words, at a global state w an agent A *says* M to an agent B iff he sent an M' to B such that he used M in composing M', A *sees* M iff he received an M' such that M is a readable sub-message of M', and A *has* M iff M is an analyzable message in A's set of possessions. A message M is a *secret* shared among the agents in a group $\mathcal{G}$ at w iff at w all the agents in $\mathcal{G}$ possess M and no agent outside the group possesses M. Moreover, A *commits* to an agent B as an initiator (respectively, responder) using M iff A has performed the actions in the initiator's (respectively, responder's) commit set. Furthermore, an agent A *implicitly believes* in φ at

w iff φ holds in all the worlds indistinguishable to A from w, which is the standard interpretation of the belief of logically omniscient agents. We employ the explicit belief modality (and awareness) to formalize the formulas in which a non-omniscient, resource-bounded agent believes: We start by restricting the formulas φ that an agent might explicitly believe in at a global state w to those in his language, which is expressed by $\varphi \in \mathcal{F}_A(w)$ (see Def. 4), and then further restrict these formulas to those he can prove using the information he currently possesses, which is captured using the local truth relation $\models_A$.

2.3.2 Local truth

$\mathfrak{M}, w \models_A \varphi$ captures the truth of a formula φ relative to an agent A at a global state w. Since there are situations in which φ expresses properties of A himself, and situations in which φ expresses properties of other agents, we will distinguish between these two forms of reasoning in our definition below. In particular, different forms of reasoning require different kinds of information. For example, if A has to check whether he possesses M, he will check whether his possession set contains M. But, to check whether an agent B has M, A cannot just access the set of B's possessions. In our formalization, A uses his awareness set to determine whether B used M to compose a message B has sent, or that B received M in some message M' that B can analyze.

Modeling an agent reasoning about his own local state is straightforward: we use the sets comp, submsg and analz to define the semantics for the says, sees and has predicates, respectively, as in Def. 6.

Modeling an agent reasoning about other agents is more complicated. Here we employ the agent's awareness set to define the semantics of the formulas. To accomplish this, we define "meta-versions" of the sets comp and submsg, expressing the messages that some other agent may have used to compose a message he has sent, as well as the submessages he might be able to obtain from a message he has received. These capabilities are formalized by means of the sets m-comp and m-submsg, respectively. The set $\mathsf{m\text{-}comp}_A(B, C, w, M)$ consists of the messages that, at global state w, A expects B to have used to send the message M to some agent C. The set $\mathsf{m\text{-}submsg}_A(B, w, M)$ consists of the sub-messages of M that, according to A's awareness set at w, agent B might be able to possess. The rules defining these sets are given in Fig. 3.

We explain the intuition behind some these rules. Rule mc-inj formalizes that if an agent A is aware that an agent B sent a message M, then M is among the messages that A expects B to have sent. In ms-pk, if A observed that B received a message M' such that M encrypted with B's public key K_B is part of M', then A concludes that B has M. Note that, although an agent reasons about messages that he may be unable to analyze, there will not be any secrecy violation following from these rules: reasoning about the existence of a message does not correspond to possessing it.

The awareness set of an agent encodes the actions that he expects other agents to have performed. To reason about commitment, we have to check whether a set of actions, i.e. a commit set, is a subset of the awareness set of an agent. To this end, we introduce a function that maps actions ac in a set $\mathcal{C}$ to the corresponding set $form(\mathcal{C}) = \{a2f(ac) \mid ac \in \mathcal{C}\}$ of formulas, where $a2f(ac) = \mathsf{says}_A(B, M)$ if $ac = \mathtt{send}_A(B, M)$ and $a2f(ac) = \mathsf{sees}_A(M)$ if $ac = \mathtt{rec}_A(M)$.

We now turn to the formal definition of $\models_A$.

Definition 7 The truth of a formula φ relative to an agent A at a global state w in a model $\mathfrak{M} = (W, \sim, \alpha)$, in symbols $\mathfrak{M}, w \models_A \varphi$, is the smallest relation satisfying:

$$\mathfrak{M}, w \models_A \varphi \to \psi \quad \text{if} \quad \mathfrak{M}, w \not\models_A \varphi \text{ or } \mathfrak{M}, w \models_A \psi$$

For an agent reasoning about himself:[3]

$$\begin{aligned}
\mathfrak{M}, w \models_A \mathsf{says}_A(B, M) \quad &\text{if} \quad \mathtt{send}_A(B, M') \in Ac_A(w) \\
&\text{and } M \in \mathsf{comp}_A(w, M') \text{ for some } M' \\
\mathfrak{M}, w \models_A \mathsf{sees}_A(M) \quad &\text{if} \quad \mathtt{rec}_A(M') \in Ac_A(w) \\
&\text{and } M \in \mathsf{submsg}_A(w, M') \text{ for some } M' \\
\mathfrak{M}, w \models_A \mathsf{has}_A(M) \quad &\text{if} \quad M \in \mathsf{analz}(Po_A(w)) \\
\mathfrak{M}, w \models_A \mathsf{comml}_A(B, M) \quad &\text{if} \quad \mathcal{C}_I^{Prot}(A, B, M) \subseteq Ac_A(w) \\
\mathfrak{M}, w \models_A \mathsf{commR}_A(B, M) \quad &\text{if} \quad \mathcal{C}_R^{Prot}(B, A, M) \subseteq Ac_A(w)
\end{aligned}$$

For an agent A reasoning about an agent $B \neq A$:

$$\begin{aligned}
\mathfrak{M}, w \models_A \mathsf{says}_B(C, M) \quad &\text{if} \quad \mathsf{says}_B(C, M') \in \alpha_A(w) \\
&\text{and } M' \text{ such that } M \in \mathsf{m\text{-}comp}_A(B, C, w, M') \\
\mathfrak{M}, w \models_A \mathsf{sees}_B(M) \quad &\text{if} \quad \mathsf{sees}_B(M') \in \alpha_A(w) \\
&\text{and } M' \text{ such that } M \in \mathsf{m\text{-}submsg}_A(B, w, M') \\
\mathfrak{M}, w \models_A \mathsf{has}_B(M) \quad &\text{if} \quad \mathfrak{M}, w \models_A \mathsf{says}_B(C, M) \text{ for} \\
&\text{some } C \text{ or } \mathfrak{M}, w \models_A \mathsf{sees}_B(M) \\
\mathfrak{M}, w \models_A \mathsf{comml}_B(C, M) \quad &\text{if} \quad form(\mathcal{C}_I^{Prot}(B, C, M)) \subseteq \alpha_A(w) \\
&\text{for some } C \\
\mathfrak{M}, w \models_A \mathsf{commR}_B(C, M) \quad &\text{if} \quad form(\mathcal{C}_R^{Prot}(C, B, M)) \subseteq \alpha_A(w) \\
&\text{for some } C
\end{aligned}$$

The semantics for secrecy (where the agent identifiers range over the identifiers in A's possession set $Po_A(w)$) is:

$$\mathfrak{M}, w \models_A \mathsf{sec}_{\mathcal{G}}(M) \quad \text{if} \quad \mathfrak{M}, w \models_A \mathsf{has}_B(M) \text{ for all } B \in \mathcal{G} \text{ and } \mathfrak{M}, w \not\models_A \mathsf{has}_C(M) \text{ for all } C \notin \mathcal{G}. \quad \blacksquare$$

Let us give the intuition behind the definitions for an agent A reasoning about another agent B. We define that, for an agent A at global state w, B *says* M to an agent C iff A is aware that B has sent an M' to C such that M was (expected to be) used by B to compose M'. Similarly, agent B *sees* a message M iff A is aware that B has received a message M' such that M is a sub-message that B is (expected to be) able to see from M'. Agent B *has* M iff either B says or sees M. From the point of view of A, an agent B has committed to an agent C as an initiator of an execution identified by M iff A is aware that B has performed the actions in the initiator's commit set. Similarly, for the commR formula. As we observed above, there is no clause for explicit belief since an agent cannot reason about what another agent may explicitly believe.

Note that an agent reasoning about his own state (local truth) coincides with an external agent reasoning about this

[3] We do not define the $\models_A$ relation in the case of the $\mathcal{X}_A$ since this reduces trivially to $\models_A$.

$$\frac{\mathsf{says}_B(C,M) \in \alpha_A(w)}{M \in \mathsf{m\text{-}comp}_A(B,C,w,M)}\ \text{mc-inj} \qquad \frac{\{|M|\}_{K_B^{-1}} \in \mathsf{m\text{-}comp}_A(B,C,w,M')}{M \in \mathsf{m\text{-}comp}_A(B,C,w,M')}\ \text{mc-sig} \qquad \frac{M_1 \circ M_2 \in \mathsf{m\text{-}comp}_A(B,C,w,M)}{M_i \in \mathsf{m\text{-}comp}_A(B,C,w,M)}\ \text{mc-}i\ (i \in \{1,2\})$$

$$\frac{\mathsf{sees}_B(M) \in \alpha_A(w)}{M \in \mathsf{m\text{-}submsg}_A(B,w,M)}\ \text{ms-inj} \qquad \frac{\{|M|\}_{K_B} \in \mathsf{m\text{-}submsg}_A(B,w,M')}{M \in \mathsf{m\text{-}submsg}_A(B,w,M')}\ \text{ms-pk} \qquad \frac{M_1 \circ M_2 \in \mathsf{m\text{-}submsg}_A(B,w,M)}{M_i \in \mathsf{m\text{-}submsg}_A(B,w,M)}\ \text{ms-}i\ (i \in \{1,2\})$$

Figure 3: The rules defining the sets m-comp and m-submsg

agent (global truth). Hence, as shown in the appendix, it follows straightforwardly from Def. 6 and Def. 7 that:

Lemma 8 For all agents A and B, global states w, and formulas $\varphi \in \mathcal{F}_A(w)$ such that $\varphi \in \{\mathsf{says}_A(B,M), \mathsf{sees}_A(M), \mathsf{has}_A(M)\}$, we have that $\mathfrak{M}, w \models \mathcal{B}_A\,\varphi$ iff $\mathfrak{M}, w \models \mathcal{X}_A\,\varphi$. ■

To summarize, our formalization expresses that there are two sources of information (local states and awareness sets), which provide different levels of reliability (certainties and expectations) and are employed differently (for reasoning about oneself or about other agents).

2.4 Defining awareness

We use awareness to represent the expectations of an agent with respect to the actions of the agents with whom he is communicating. Each step of a protocol gives rise to (i) a rule capturing the expectations of the sender with respect to the `send` action he has performed, and (ii) a rule capturing the expectations of the receiver regarding the corresponding `rec` action. Note that an agent's expectations may not correspond to reality, as he might be aware of, and thus explicitly believe in, false statements (as is the case in the man-in-the-middle attack on the NSPK protocol, which we consider below).

The rules representing the sender perspective are obtained from the protocol steps in a straightforward manner. Given the n-th step $A \to B : M$ of a protocol *Prot*, the sender A, who has the $\mathtt{send}_A(B,M)$ action recorded in his local state, expects the receiver B to get the message M; thus, the rule $Prot\_s_n$ adds the formula $\mathsf{sees}_B(M)$ to A's awareness set:

$$\frac{\mathtt{send}_A(B,M) \in Ac_A(w)}{\mathsf{sees}_B(M) \in \alpha_A(w)}\ Prot\_s_n\,.$$

The rules capturing the expectations of the receiver depend on the protocol the agents are executing, and thus cannot be given in a general form like the sender rule. Instead, we consider a concrete example and give the receiver rules for the NSPK protocol in Fig. 4.

The intuition behind the rule nspk_r_1 is that, upon the receipt of the first message, agent B expects that it has been sent by agent A. The rule nspk_r_2 formalizes that when A receives his nonce N_A back, he may conclude that B sent it. The intuition behind nspk_r_3 is similar.

Note that the expectations of the agents do not always correspond to what is actually happening. In fact, attacks run counter to the expectations of the agents (as these are based on incomplete information).

Although illustrated only for the NSPK protocol, the ideas presented here are general. We have used our logic to reason about a number of other protocols, e.g. the full NSPK protocol and the Otway-Rees protocol with shared keys.

3. MODALITIES AND SPECIFICATIONS

In this section, we use our modalities to formally characterize different kinds of specifications of security properties. Furthermore, we show how to use them to reason semantically about attacks in interleaved protocol executions.

3.1 Extensional and intensional specifications

A number of researchers, e.g. [5, 12, 19], have observed that there are two different kinds of security specifications: extensional specifications, which are, in some sense, independent of the details of a particular protocol, and intensional specifications, where statements of properties are based on the protocol itself. For example, consider the definitions given by Roscoe [19, pages 31 and 34]:

> We classify a specification as *extensional* when it is independent of the details of the protocol and would apply to any other protocol designed to achieve the same effect. Thus, inevitably, it cannot mention the actual messages passing between nodes during a protocol since these vary from one to another. Instead, it will test the states of mind (knowledge, belief, etc.) of the various participants including the spy.
>
> A specification is classified as *intensional* when its primary purpose is to assert a property of the way, in terms of communications within a protocol, a particular state is reached.

Until now, these definitions have lacked a formal status. One of the contributions of our work is to characterize these notions in terms of our modalities.

We begin by observing that the implicit belief modality has an extensional character as the properties it formalizes are independent of the particular message exchanges. Intuitively, this is because implicit belief captures the view of an external, resource-unbounded observer following the protocol execution. In order to check whether a property denoted by a formula φ holds, such an observer need not be aware of the particular message exchanges of the protocol execution; rather, he simply checks whether the local states of the agents satisfy φ.

$$\frac{\mathtt{rec}_B(\{|A \circ N_A|\}_{K_B}) \in Ac_B(w)}{\mathsf{says}_A(B, \{|A \circ N_A|\}_{K_B}) \in \alpha_B(w)}\ \mathsf{nspk\_r}_1 \qquad \frac{\begin{array}{c}\mathtt{send}_A(B, \{|A \circ N_A|\}_{K_B}) \in Ac_A(w) \\ \mathtt{rec}_A(\{|N_A \circ N_B|\}_{K_A}) \in Ac_A(w)\end{array}}{\mathsf{says}_B(A, \{|N_A \circ N_B|\}_{K_A}) \in \alpha_A(w)}\ \mathsf{nspk\_r}_2 \qquad \frac{\begin{array}{c}\mathtt{send}_B(A, \{|N_A \circ N_B|\}_{K_A}) \in Ac_B(w) \\ \mathtt{rec}_B(\{|N_B|\}_{K_B}) \in Ac_B(w)\end{array}}{\mathsf{says}_A(B, \{|N_B|\}_{K_B}) \in \alpha_B(w)}\ \mathsf{nspk\_r}_3$$

Figure 4: Receiver rules for the NSPK protocol

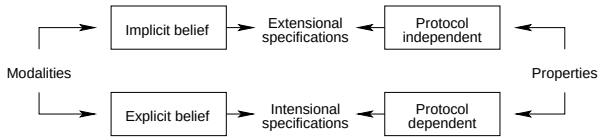

Figure 5: Relation between modalities, kinds of specification and security properties

In contrast, the explicit belief modality has an intensional character. This modality is based on the agents' awareness sets, which model the agents' local, resource-bounded views of the expected results of their actions. The awareness sets are in turn determined by the protocol rules, and hence statements about explicit belief are statements about the results of particular protocol steps. Fig. 5 summarizes these relationships.

This logical characterization of the two definitions has the status of a thesis: since the definitions are informal (natural language), our thesis cannot be formally proven. However, we can support it by showing that it holds for different commonly considered kinds of specifications. In what follows, we will consider two examples: *secrecy* of a message (which can be specified extensionally) and an agent's *completion* of a protocol execution (which is inherently intensional).

To illustrate how these properties can be modally specified, we return to the NSPK protocol, our running example. We will start from an informal specification of a correctness requirement for a responder of the protocol, expressed in terms of secrecy and completion. We then show how a progressive refinement of the informal specifications of these properties leads to their formal specifications in terms of implicit and explicit belief, supporting our thesis. Moreover, in §3.3 and §4, we will use the resulting specification to show how we can reason about modal specifications, and in what sense this specification is equivalent to one stated directly in terms of properties of interleaved traces.

3.2 Formalizing secrecy and completion

A message is a secret between a group of agents at some state of a protocol execution when, at that state, all the agents in the group have the message, and all the remaining agents do not have it. The way in which the state is reached is irrelevant to the secrecy of a message. Only the possession of the agents at that particular state matters! On the other hand, the completion of a protocol execution by an agent refers to a sequence of actions performed by the agent, not to the properties of an individual state. Thus, a specification of completion involves the details of the protocol and is therefore intensional.

Consider again the NSPK protocol. Agent B uses his nonce N_B as a challenge to authenticate agent A. One way of expressing the correctness requirement for a responder B executing an instance of the NSPK protocol with an initiator A is by means of the following two properties that must hold in a trace t:

P1. *Completion*: B completes an execution with A using the nonce N_B.

P2. *Secrecy*: The responder's nonce N_B is a secret between B and the initiator A.

Note that secrecy is necessary, but not sufficient, for correctness; we should check whether the responder's nonce N_B is a secret only when the corresponding protocol execution is completed. That is, property 1 should imply property 2.

We refine these informal specifications and formalize them within our logic. We start with the completion property, which we state in a more refined but still informal way.

P1'. There exists a state in the trace such that B completes an execution as a responder with an initiator A using a nonce N_B.

Formalizing the meaning of "B completes an execution with an initiator A using a nonce N_B", we obtain:

P1". There exists a state w of $\mathfrak{M}^t$ such that $\mathfrak{M}^t, w \models \mathcal{X}_B\, \mathsf{commR}_B(A, N_B)$.

More specifically, the predicate commR formalizes that B has completed the execution as a responder, and the intensionality is represented by the use of the explicit belief modality.

We now turn to the secrecy property, which (bringing out its inherent extensionality) we can state in a more refined but still informal way as:

P2'. There exists a state in the trace such that N_B is a secret between agents B and A.

The formalization of this extensional specification is based on the fact that, in order to specify what a secret is, we do not need to refer to the protocol: N_B is a secret between B and A iff B and A are the only agents who possess N_B. We can directly formalize this using the sec predicate and the implicit belief modality to obtain:

P2". There exists a state w of $\mathfrak{M}^t$ such that $\mathfrak{M}^t, w \models \mathcal{B}_B\, \mathsf{sec}_{\{A,B\}}(N_B)$.

The correctness requirement for the responder of an execution of the NSPK protocol combines completion and secrecy by requiring that if, for some trace t, B completes an execution identified by N_B as a responder of A, then N_B must be a secret between B and A. Formally, the responder's requirement for the protocol is: for all $t \in P$, models $\mathfrak{M}^t$, agents A and B, and nonces N_B,

$$\mathfrak{M}^t \models \mathcal{X}_B\, \mathsf{commR}_B(A, N_B) \rightarrow \mathcal{B}_B\, \mathsf{sec}_{\{A,B\}}(N_B)\,. \qquad (1)$$

3.3 Reasoning about the NSPK protocol

We now show how to use our semantics to reason about specifications. There are two possibilities, depending on the relationship of the specification to the set of intended models (i.e. traces). The first possibility is verification, which is establishing the correctness of a protocol by showing that it holds for all "protocol conform" models. For instance, we could show that all models $\mathfrak{M}^t$, resulting from all possible NSPK protocol traces t, satisfy the agents' requirements, such as that for B in (1). In a manner similar to Paulson's inductive method, such verification could be carried out by induction over the set of all models $\mathfrak{M}^t$, corresponding to those t in the inductively defined set of traces.

A second possibility is falsification. We will illustrate this here by giving a model where B's requirement (1) fails to hold. That is, to falsify (1), we give a particular model $\mathfrak{M}^t_{\text{MITM}}$ that models an execution trace corresponding to the man-in-the-middle (MITM) attack.

Theorem 9 There exist an NSPK execution trace t, a model $\mathfrak{M}^t$, agents A and B, and a nonce N_B such that $\mathfrak{M}^t \not\models \mathcal{X}_B\, \mathsf{commR}_B(A, N_B) \rightarrow \mathcal{B}_B\, \mathsf{sec}_{\{A,B\}}(N_B)$. ■

The MITM attack on the NSPK protocol [16] consists of the sequence of events shown in Fig. 1. Thus, consider the model $\mathfrak{M}^t_{\text{MITM}} = (W^t, \sim, \alpha)$ obtained from the trace $t = \langle ev_1, ev_2, ev_3, ev_4, ev_5\rangle$, which represents the smallest sequence of events containing this attack. The components of $\mathfrak{M}^t_{\text{MITM}}$ that are relevant for our analysis are:

- $W^t = \{w_0, w_1, w_2, w_3, w_4, w_5\}$,
- $\sim_a = \{(w_0,w_0),(w_1,w_1),(w_1,w_2),(w_2,w_1),(w_2,w_2),$ $(w_3,w_3),(w_4,w_4),(w_4,w_5),(w_5,w_4),(w_5,w_5)\}$,
- $\sim_b = \{(w_0,w_0),(w_0,w_1),(w_1,w_0),(w_1,w_1),(w_2,w_2),$ $(w_3,w_3),(w_3,w_4),(w_4,w_3),(w_4,w_4),(w_5,w_5)\}$,
- $\alpha_a(w_5) = \{\mathsf{sees}_{spy}(\{\!|a \circ n_a|\!\}_{k_{spy}}), \mathsf{says}_{spy}(a, \{\!|n_a \circ n_b|\!\}_{k_{spy}}),$ $\mathsf{sees}_{spy}(\{\!|n_b|\!\}_{k_{spy}})\}$,
- $\alpha_b(w_5) = \{\mathsf{says}_a(b, \{\!|a \circ n_a|\!\}_{k_b}), \mathsf{sees}_a(\{\!|n_a \circ n_b|\!\}_{k_a}),$ $\mathsf{says}_a(b, \{\!|n_b|\!\}_{k_b})\}$.

We focus on w_5 since it is the global state obtained after the last event in t. The local states of the agents a and b and the possessions of the *spy* at w_5 are:

$$s_a(w_5) = \begin{cases} Po_a(w_5) = \{a, k_a, k_a^{-1}, spy, k_{spy}, n_a, n_b\}, \\ Ac_a(w_5) = \{\mathtt{send}_a(spy, \{\!|a \circ n_a|\!\}_{k_{spy}}), \\ \quad \mathtt{rec}_a(\{\!|n_a \circ n_b|\!\}_{k_a}), \mathtt{send}_a(spy, \{\!|n_b|\!\}_{k_{spy}})\} \end{cases},$$

$$s_b(w_5) = \begin{cases} Po_b(w_5) = \{b, k_b, k_b^{-1}, a, k_a, n_a, n_b\} \\ Ac_b(w_5) = \{\mathtt{rec}_b(\{\!|a \circ n_a|\!\}_{k_b}), \\ \quad \mathtt{send}_b(a, \{\!|n_a \circ n_b|\!\}_{k_a}), \mathtt{rec}_b(\{\!|n_b|\!\}_{k_b})\} \end{cases},$$

$$Po_{spy}(w_5) = \{spy, k_{spy}, k_{spy}^{-1}, a, k_a, b, k_b, n_a, n_b\},$$

where $initState(a) = \{a, k_a, k_a^{-1}, spy, k_{spy}\}$, $initState(b) = \{k_a, b, k_b, k_b^{-1}\}$ and $initState(spy) = \{spy, k_{spy}, k_{spy}^{-1}, a, k_a, b, k_b\}$. We can then use the semantics to demonstrate the existence of the attack (as shown in the proof of Theorem 9 in the appendix).

4. MODAL VERSUS TRACE-BASED SPECIFICATIONS

We now show that our modal specification for B's correctness requirement for the NSPK protocol is equivalent to a trace-based specification of the same protocol requirement. We establish this by showing the logical equivalence of both specifications with respect to our semantics.

As noted above, a trace-based interleaved semantics can be used both for interactive verification [18] and for falsification based on infinite-state model-checking [3]. The specifications in both approaches are intensional and specify what must (or cannot) hold after certain occurrences of events. For example, for verification, in the case of NSPK one might specify B's requirement by formalizing that N_B is a secret after the last two steps of the protocol have occurred:

$$(\mathsf{sees}_B(\{\!|N_B|\!\}_{K_B}) \wedge \mathsf{says}_B(A, \{\!|N_A \circ N_B|\!\}_{K_A})) \rightarrow \neg \mathsf{has}_{spy}(N_B)\,. \qquad (2)$$

For falsification, one formalizes the negation of (2), i.e.

$$(\mathsf{sees}_B(\{\!|N_B|\!\}_{K_B}) \wedge \mathsf{says}_B(A, \{\!|N_A \circ N_B|\!\}_{K_A})) \wedge \mathsf{has}_{spy}(N_B)\,, \qquad (3)$$

and searches for a trace with this property.

The specification (3) is a direct translation of the Haskell program used in [3] to specify an attack.[4] This can be directly expressed as a formula in our logic and proved for some A, B, and N_B, at some world w of some model $\mathfrak{M}^t$ resulting from some execution trace t.

Showing that this is equivalent to the statement of Theorem 9 establishes that the attack in the trace-based specification is equivalent to the attack in our modal specification with respect to our semantics. The equivalence between the two specifications can be shown alternatively (in terms of "verification" rather than "falsification") by showing the

[4]Paulson's verification specification is similar to (2). He formalizes an intensional specification of secrecy for the nonce N_B by stating that if there is an event $B \rightarrow A : \{\!|N_A \circ N_B|\!\}_{K_A}$ in the set of traces modeling the NSPK protocol, then the *spy* does not possess the nonce N_B.

equivalence of (1) and of a formula representing the correctness of B's requirement. As shown in the appendix, for non-compromised agents we have:

Theorem 10 For all traces t of the NSPK protocol, models $\mathfrak{M}^t$, agents A, B such that $A \neq spy$, $B \neq spy$, $\mathfrak{M}^t \models \neg \mathsf{has}_{spy}(K_A^{-1})$ and $\mathfrak{M}^t \models \neg \mathsf{has}_{spy}(K_B^{-1})$, and nonces N_B,

$$\mathfrak{M}^t \models \mathcal{X}_B \, \mathsf{commR}_B(A, N_B) \rightarrow \mathcal{B}_B \, \mathsf{sec}_{\{A,B\}}(N_B)$$

iff

$$\mathfrak{M}^t \models (\mathsf{sees}_B(\{\!|N_B|\!\}_{K_B}) \wedge \mathsf{says}_B(A, \{\!|N_A \circ N_B|\!\}_{K_A})) \rightarrow \neg \mathsf{has}_{spy}(N_B)\,.$$

5. RELATED WORK

We now compare our work with related approaches to specifying and classifying security properties. Abadi and Tuttle [1] define a possible-worlds semantics for an extension of BAN that models interleaved protocol executions. However, details and examples are lacking so that a thorough comparison is difficult. Although their logic lacks an explicit notion of awareness, their *hide* operator conceals the contents of unreadable messages, and thus provides a basis for modeling "belief as a form of resource-bounded, defeasible knowledge" [1, p. 202]. It thereby captures some of the notions that our explicit belief modality formalizes.

In [2], we initially investigated how to use awareness to model resource-bounded reasoning in interleaved protocol executions. The multi-modal logic that we have given here differs considerably from [2]: while both are based on the explicit and implicit beliefs of the agents, here we modified and systematized the semantics for the modalities, the method how the awareness sets are computed, and how the logic is employed to specify properties and reason about attacks.

Interleaved trace-based semantics is a standard approach to modeling distributed computation. Paulson [18] has championed its use for inductive verification of security protocols, and the same semantic model can directly be used for model checking as well, e.g., as in [3]. Specifications in this setting (whether for verification or model checking) tend to be intensional as they are formalized in terms of sequences of protocol specific events. Our results in §4 illustrate how we can employ our modal specification to provide more abstract, high-level specifications of security properties with similar expressive power based on this semantic model.

Our definitions of intensional and extensional specifications come from Roscoe [19]. He also introduces the notion of *canonical intensional specification*, which "simply asserts that the protocol runs as expected" [19, p. 34], i.e. no agent can believe a protocol execution has completed unless the correct series of messages has occurred (consistent with all the various parameters) up to and including the last message the agent communicates. In our approach, this intensional character is directly formalized by the commit sets $\mathcal{C}$, and specified with the explicit belief modality. Note, however, that since we model action sets instead of action sequences, we cannot formalize the order in which the actions occur. However, it is straightforward to modify our framework to capture this idea.

A number of other authors, e.g. [5, 12, 17, 21], have looked at classifying and relating specifications. Notable in this regard is the work of Lowe [17], who uses CSP to formalize a hierarchy of authentication specifications, in which each level of the hierarchy expresses one possible meaning of "entity authentication". These specifications are all intensional; abstract notions such as secrecy are not accounted for. Using explicit belief it should be possible to formalize similar hierarchies in our setting. Moreover, using implicit belief it should be possible to extend these hierarchies, for example combining the intensional notion of "injective agreement" with the extensional requirement that some of the messages exchanged should remain secret.

6. CONCLUSIONS AND OUTLOOK

We have defined a multi-modal security logic with a trace-based semantics. Our logic combines the simple expressive semantics of trace-based approaches with the use of modalities to support high-level, trace-independent specifications of security properties based on different notions of belief. The logic also sheds light on, and allows us to give a logical characterization of, extensional and intensional specifications of security properties.

There is considerable work ahead and many interesting problems are still open. First, the account we have given is semantic. Via a semantic embedding, for example in higher-order logic, we could mechanize deductions in Isabelle (we have already carried out some initial work in this direction). More interesting though is to derive, from the semantics, higher-level proof rules for reasoning about the modalities.

Second, we have illustrated the logical equivalence between trace-based specifications (translated into our setting) and modal specifications. What is missing is a general statement about such equivalences. Such a statement is difficult as it requires the definition of a general class of trace-based specifications, and circumscribing such a class is problematic due to their intensional nature. One possible solution, which we would like to investigate, is to show equivalence for particular classes of specifications. For example, the semantics of the commit formulas captures an idea that is very close to the one of *matching histories* [8], except that, since we use sets of actions instead of sequences, we cannot talk about their ordering.

Finally, in our example in reasoning about attacks (i.e. the man-in-the-middle attack on the NSPK protocol) we knew of its existence in advance. One of the advantages of logics like BAN is that, in some cases, they allow for a kind of abductive reasoning as they provide a way of finding attacks by identifying missing assumptions required for proofs. When a deductive system for our logic is in place, we will also have the chance to explore these possibilities.

7. REFERENCES

[1] M. Abadi and M. R. Tuttle. A semantics for a logic of authentication. In *Proceedings of the 10th Annual ACM Symposium on Principles of Distributed Computing*, pages 201–216. ACM Press, 1991.

[2] R. Accorsi, D. Basin, and L. Viganò. Towards an awareness-based semantics for security protocol

analysis. In J. Goubault-Larrecq, editor, *Proceedings of the Post-CAV Workshop on Logical Aspects of Cryptographic Protocol Verification*, ENTCS 55(1). Elsevier, 2001.

[3] D. Basin. Lazy infinite-state analysis of security protocols. In R. Baumgart, editor, *Secure Networking: CQRE'99*, LNCS 1740, pages 30–42. Springer-Verlag, 1999.

[4] A. Bleeker and L. Meertens. A semantics for BAN logic. In *Proceeding of DIMACS Workshop on Design and Formal Verification of Crypto Protocols*. 1997.

[5] C. Boyd. Extensional goals in authentication protocols. In *Proceedings of DIMACS Workshop on Design and Formal Verification of Crypto Protocols*. 1997.

[6] M. Burrows, M. Abadi, and R. Needham. A logic of authentication. *ACM Transactions on Computer Systems*, 8(1):18–36, 1990.

[7] I. Cervesato and P. F. Syverson. The logic of authentication protocols. In R. Focardi and R. Gorrieri, editors, *Foundations of Security Analysis and Design*, LNCS 2171, pages 63–136. Springer-Verlag, 2001.

[8] W. Diffie, P. C. van Oorschot, and M. J. Wiener. Authentication and authenticated key exchanges. *Designs, Codes and Cryptography*, 2:107–125, 1992.

[9] D. Dolev and A. C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 2(29):198–208, March 1983.

[10] R. Fagin and J. Y. Halpern. Belief, awareness and limited reasoning. *Artificial Intelligence*, 34(1):39–76, 1987.

[11] R. Fagin, J. Y. Halpern, Y. Moses, and M. Y. Vardi. *Reasoning about knowledge*. MIT Press, 1995.

[12] D. Gollmann. Authentication – myths and misconceptions. In *Progress in Computer Science and Applied Logic*. Birkhäuser Verlag, 2001.

[13] L. Gong, R. Needham, and R. Yahalom. Reasoning about belief in cryptographic protocols. In *Proceedings of the 1990 IEEE Symposium on Security and Privacy*, pages 234–248. IEEE Computer Society Press, 1990.

[14] G. E. Hughes and M. J. Cresswell. *A new introduction to modal logic*. Routledge, 1996.

[15] H. J. Levesque. A logic of implicit and explicit belief. In *Proceedings of AAAI'84*, pages 198–202. 1984.

[16] G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Proceedings of TACAS'96*, LNCS 1055, pages 147–166. Springer-Verlag, 1996.

[17] G. Lowe. A hierarchy of authentication specifications. In *Proceedings of the 10th IEEE Computer Security Foundations Workshop: CSFW'97*, pages 31–43. IEEE Computer Society Press, 1997.

[18] L. C. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, 6:85–128, 1998.

[19] A. W. Roscoe. Intensional specifications of security protocols. In *Proceedings of the 9th IEEE Computer Security Foundations Workshop: CSFW'96*, pages 28–38. IEEE Computer Society Press, 1996.

[20] P. F. Syverson. Knowledge, belief, and semantics in the analysis of cryptographic protocols. *Journal of Computer Security*, 1:317–334, 1992.

[21] P. F. Syverson and P. C. van Oorschot. On unifying some cryptographic protocol logics. In *Proceedings of the 1994 IEEE Computer Society Symposium on Research in Security and Privacy*. IEEE Computer Society Press, 1994.

[22] E. Thijsse. On total awareness logics. In M. de Rijke, editor, *Defaults and Diamonds*, pages 309–347. Kluwer Academic Publishers, 1993.

APPENDIX

PROOF OF LEMMA 8. We begin by observing that, for a formula $\varphi \in \mathcal{F}_A(w)$ such that $\varphi \in \{\mathsf{says}_A(B, M), \mathsf{sees}_A(M), \mathsf{has}_A(M)\}$, we trivially have that

$$\mathfrak{M}, w \models \varphi \text{ iff } \mathfrak{M}, w \models_A \varphi \tag{4}$$

since the definitions of $\mathfrak{M} \models \varphi$ and $\mathfrak{M} \models_A \varphi$ are in this case identical. The proof then proceeds as follows.

(Left-to-right) Assume that $\mathfrak{M} \models \mathcal{B}_A \varphi$. By definition, $\mathfrak{M}, w' \models \varphi$ for all w' such that $w \sim_A w'$, and thus $\mathfrak{M}, w \models \varphi$ as $w \sim_A w$ by definition. From (4), we then have that $\mathfrak{M}, w \models_A \varphi$, and thus, by definition, $\mathfrak{M}, w \models \mathcal{X}_A \varphi$.

(Right-to-left) Assume that $\mathfrak{M}, w \models \mathcal{X}_A \varphi$. By definition, $\mathfrak{M}, w \models_A \varphi$ and thus from (4) we have that $\mathfrak{M}, w \models \varphi$. Since φ characterizes a property of A's local state, if $\mathfrak{M}, w \models \varphi$ then $\mathfrak{M}, w' \models \varphi$ for all worlds w' in the equivalence class induced by A's indistinguishability relation $\sim_A$. Thus, by definition, $\mathfrak{M}, w \models \mathcal{B}_A \varphi$. □

PROOF OF THEOREM 9. We first show that $\mathfrak{M}^t_{\text{MITM}}, w_5 \models \mathcal{X}_b\, \mathsf{commR}_b(a, n_b)$ and $\mathfrak{M}^t_{\text{MITM}}, w_5 \not\models \mathcal{B}_b\, \mathsf{sec}_{\{a,b\}}(n_b)$. By definition of explicit belief, $\mathfrak{M}^t_{\text{MITM}}, w_5 \models \mathcal{X}_b\, \mathsf{commR}_b(a, n_b)$ iff $\mathfrak{M}^t_{\text{MITM}}, w_5 \models_b \mathsf{commR}_b(a, n_b)$ and $\mathsf{commR}_b(a, n_b) \in \mathcal{F}_b(w_5)$. From $Po_b(w_5)$ it follows that $\mathsf{commR}_b(a, n_b) \in \mathcal{F}_b(w_5)$ holds. By definition, $\mathfrak{M}^t_{\text{MITM}}, w_5 \models_b \mathsf{commR}_b(a, n_b)$ holds iff $\mathcal{C}_R^{\text{NSPK}}(a, b, n_b) \subseteq Ac_b(w_5)$, which holds because $\mathcal{C}_R^{\text{NSPK}}(a, b, n_b) = \{\mathtt{rec}_b(\{\!|a \circ n_a|\!\}_{k_b}), \mathtt{send}_b(a, \{\!|n_a \circ n_b|\!\}_{k_a}), \mathtt{rec}_b(\{\!|n_b|\!\}_{k_b})\}$. We can thus conclude that $\mathfrak{M}^t_{\text{MITM}}, w_5 \models \mathcal{X}_b \mathsf{commR}_b(a, n_b)$.

To show that b does not implicitly believe in $\mathsf{sec}_{\{a,b\}}(n_b)$ at w_5, observe that by definition $\mathfrak{M}^t_{\text{MITM}}, w_5 \not\models \mathcal{B}_b\, \mathsf{sec}_{\{a,b\}}(n_b)$ iff $\mathfrak{M}^t_{\text{MITM}}, w' \not\models \mathsf{sec}_{\{a,b\}}(n_b)$ for some w' such that $w_5 \sim_b w'$. Since w' can only be w_5 by the definition of $\sim_b$ in $\mathfrak{M}^t_{\text{MITM}}$, we check whether $\mathfrak{M}^t_{\text{MITM}}, w_5 \models \mathsf{sec}_{\{a,b\}}(n_b)$, which holds iff $\mathfrak{M}^t_{\text{MITM}}, w_5 \models \mathsf{has}_C(n_b)$ for all agents $C \in \{a, b\}$, and $\mathfrak{M}^t_{\text{MITM}}$, $w_5 \not\models \mathsf{has}_D(n_b)$ for all agents $D \notin \{a, b\}$. $Po_a(w_5)$ and $Po_b(w_5)$ tell us that both a and b possess n_b. Since we are only considering agents a, b and the *spy*, D can only be the *spy*. Since $\mathfrak{M}^t_{\text{MITM}}, w_5 \models \mathsf{has}_{spy}(n_b)$, we conclude that $\mathfrak{M}^t_{\text{MITM}}, w_5 \not\models \mathsf{sec}_{\{a,b\}}(n_b)$. □

PROOF OF THEOREM 10. (Left-to-right) We assume $\mathfrak{M}^t \models \mathcal{X}_B\, \mathsf{commR}_B(A, N_B) \rightarrow \mathcal{B}_B\, \mathsf{sec}_{\{A,B\}}(N_B)$ and $\mathfrak{M}^t, w \models \mathsf{sees}_B(\{\!|N_B|\!\}_{K_B}) \wedge \mathsf{says}_B(A, \{\!|N_A \circ N_B|\!\}_{K_A})$ for an arbitrary w, and show that $\mathfrak{M}^t, w \models \neg\mathsf{has}_{spy}(N_B)$.

$\mathfrak{M}^t, w \models \mathsf{sees}_B(\{\!|N_B|\!\}_{K_B})$ implies that there exists a message M such that $\mathtt{rec}_B(M) \in Ac_B(w)$ and $\{\!|N_B|\!\}_{K_B} \in \mathsf{submsg}_B(w, M)$. By the inductive definition of the protocol, M can only be $\{\!|N_B|\!\}_{K_B}$, which implies that $\mathtt{rec}_B(\{\!|N_B|\!\}_{K_B}) \in Ac_B(w)$.

$\mathfrak{M}^t, w \models \mathsf{says}_B(A, \{\!|N_A \circ N_B|\!\}_{K_A})$ implies that there exists an M such that $\mathtt{send}_B(A, M) \in Ac_B(w)$ and $\{\!|N_A \circ N_B|\!\}_{K_A} \in \mathsf{comp}_A(w, M)$. By the inductive definition of the protocol, M can only be $\{\!|N_A \circ N_B|\!\}_{K_A}$, which implies that $\mathtt{send}_B(A, \{\!|N_A \circ N_B|\!\}_{K_A}) \in Ac_B(w)$.

From $\mathtt{send}_B(A, \{\!|N_A \circ N_B|\!\}_{K_A}) \in Ac_B(w)$ and from $\mathtt{rec}_B(\{\!|N_B|\!\}_{K_B}) \in Ac_B(w)$ it follows, again by the inductive definition of the protocol, that $\mathtt{rec}_B(\{\!|A \circ N_A|\!\}_{K_B}) \in Ac_B(w)$. Thus, $\mathcal{C}_R^{\text{NSPK}}(A, B, N_B) \subseteq Ac_B(w)$. This implies that $\mathfrak{M}^t, w \models_B \mathsf{commR}_B(A, N_B)$. Since it is straightforward to show that $\mathsf{commR}_B(A, N_B) \in \mathcal{F}_B(w)$, we have that $\mathfrak{M}^t, w \models \mathcal{X}_B\, \mathsf{commR}_B(A, N_B)$.

The assumption and $\mathfrak{M}^t, w \models \mathcal{X}_B\, \mathsf{commR}_B(A, N_B)$ imply $\mathfrak{M}^t, w \models \mathcal{B}_B\, \mathsf{sec}_{\{A,B\}}(N_B)$, i.e. $\mathfrak{M}^t, w' \models \mathsf{sec}_{\{A,B\}}(N_B)$ for all w' such that $w \sim_B w'$. By the reflexivity of $\sim_B$ we have $\mathfrak{M}^t, w \models \mathsf{sec}_{\{A,B\}}(N_B)$, and by the definition of secrecy we have $\mathfrak{M}^t, w \not\models \mathsf{has}_C(N_B)$ for all $C \notin \{A, B\}$, so we can conclude that $\mathfrak{M}^t, w \models \neg\mathsf{has}_{spy}(N_B)$.

(Right-to-left) We assume $\mathfrak{M}^t \models (\mathsf{sees}_B(\{\!|N_B|\!\}_{K_B}) \wedge \mathsf{says}_B(A, \{\!|N_A \circ N_B|\!\}_{K_A})) \rightarrow \neg\mathsf{has}_{spy}(N_B)$ and $\mathfrak{M}^t, w \models \mathcal{X}_B\, \mathsf{commr}(B, A, N_B)$ for an arbitrary w, and show that $\mathfrak{M}^t, w \models \mathcal{B}_B\, \mathsf{sec}_{\{A,B\}}(N_B)$, i.e. $\mathfrak{M}^t, w' \models \mathsf{sec}_{\{A,B\}}(N_B)$ for all $w' \in \mathfrak{M}^t$ such that $w \sim_B w'$.

$\mathfrak{M}^t, w \models \mathcal{X}_B\, \mathsf{commR}_B(A, N_B)$ implies, by definition, that $\mathfrak{M}^t, w \models_B \mathsf{commR}_B(A, N_B)$. From the definition of commitment, it follows that $\mathcal{C}_R^{\text{NSPK}}(A, B, N_B) \subseteq Ac_B(w)$. Since $w \sim_B w'$, we have that $Ac_B(w) = Ac_B(w')$ and thus $\mathcal{C}_R^{\text{NSPK}}(A, B, N_B) \subseteq Ac_B(w')$. This implies both (i) $\mathfrak{M}^t, w' \models \mathsf{says}_B(A, \{\!|N_A \circ N_B|\!\}_{K_A})$, since $\mathtt{send}_B(A, \{\!|N_A \circ N_B|\!\}_{K_A}) \in Ac_B(w')$ and $\{\!|N_A \circ N_B|\!\}_{K_A} \in \mathsf{comp}_B(w', \{\!|N_A \circ N_B|\!\}_{K_A})$, and (ii) $\mathfrak{M}^t, w' \models \mathsf{sees}_B(\{\!|N_B|\!\}_{K_B})$, since $\mathtt{rec}_B(\{\!|N_B|\!\}_{K_B}) \in Ac_B(w')$ such that $\{\!|N_B|\!\}_{K_B} \in \mathsf{submsg}_B(w', \{\!|N_B|\!\}_{K_B})$. The assumption, together with (i) and (ii), implies $\mathfrak{M}^t, w' \models \neg\mathsf{has}_{spy}(N_B)$. Since B sent a message in the form of step two to A, we also have that $\{\!|N_A \circ N_B|\!\}_{K_A} \in Po_A(w')$ and $N_B \in \mathsf{analz}(Po_A(w'))$, which implies that $\mathfrak{M}^t, w' \models \mathsf{has}_A(N_B)$. Moreover, $\{\!|N_B|\!\}_{K_B} \in Po_B(w')$ and $N_B \in \mathsf{analz}(Po_B(w'))$, imply that $\mathfrak{M}^t, w' \models \mathsf{has}_B(N_B)$. It thus follows that $\mathfrak{M}^t, w' \models \mathsf{sec}_{\{A,B\}}(N_B)$ for an arbitrary w' such that $w \sim_B w'$, and hence $\mathfrak{M}^t, w \models \mathcal{B}_B\, \mathsf{sec}_{\{A,B\}}(N_B)$. □

An agent digital signature in an untrusted environment *

K. Cartrysse
Delft University of Technology
P.O Box 5031
2600 GA Delft
The Netherlands
K.Cartrysse@ITS.TUDelft.nl

J.C.A. van der Lubbe
Delft University of Technology
P.O Box 5031
2600 GA Delft
The Netherlands
J.C.A.vdlubbe@ITS.TUDelft.nl

ABSTRACT

As agent technology is evolving, security becomes more important. This paper addresses the problem of computing with secret data, in particular the application of a digital signature. In case an agent must sign a document at a foreign host, it is not desirable to give the agent's private key to this host. The private key can be seen as the only information that can proof the identity of the agent, hence this information should not be revealed to anyone. This paper gives several solutions how an agent can sign a document without giving its private key to the host. Double signing is a problem here, and also for this some solutions are given that makes it less attractive for the host to sign documents under the agent's name without having permission for it. The solutions can be seen as a new privacy enhancing technology for agent applications.

General Terms

Digital signatures, blind signatures, hidden private key, privacy enhancing technologies, agent security

1. INTRODUCTION

Agent technology is nowadays seen as one of the technologies that will play a key role for future IT-applications. As this technology evolves, awareness about security of this kind of technology is increasing. When looking from the user's point of view, an agent is sent out to be executed at some foreign host. Sometimes, information with respect to security about this host is known and appropriate measures can be taken. However, in many cases the user does not know anything about the foreign host and therefore wants it to be protected against this host and against other agents present at that host.

Providing security solutions for agents is even more complex than in conventional systems for the simple reason that the agent uses resources from a host it does not know whether it can be trusted. Even in the case that a good working trust model is present, there is still some data that an agent should not provide to any entity in the system, including the host. Such kind of data is the agent's private key. The agent's private key is the information that enables the agent to proof its identity to other parties. It is clear that no other entity than the user of the agent should have access to it.

*This research has been performed within the framework of PISA, an interdisciplinary project related to privacy enhancing technologies in intelligent agent systems supported by the IST-programme of the European Commission.

One of the security mechanisms an agent should have access to is the digital signature.During its life cycle it will need to be able to sign documents or authenticate itself by using digital signatures. The problem is that signing a document involves the agent's (or user's) private key and as said before this is the most privacy-critical information of the agent, hence in case the agent platform cannot be trusted how can an agent sign a document without the platform being able to use its private key for other purposes. This paper proposes several solutions to this specific problem of how an agent can sign some document without anybody (except the user) can have access to the agent's private key. It can be seen as a special case of computing with secret data.

This paper is organized as follows. In the following section a solution outline is proposed in combination with related research. Section 3 describes several solutions and finally conclusions are written in section 4.

2. SOLUTION OUTLINE

In [14] several problems are stated that should be solved in order to provide security to mobile agents. One of these problems is how computing with secrets in public is possible. An example of computing with secrets in public is the question of how an agent can remotely sign a document without disclosing the private key. Sander proposes an outline how these signatures should be constructed. In [10] a solution is proposed based on RSA, where not only the private key is hidden but the complete signature procedure. Several other solutions are proposed in [6] and [9].

The approach taken in this paper is different in a way that not the entire function is hidden, but only the data (private key), while maintaining the property that a signature can only be set once. The advantage is that the signature and verification formulas can be used with a hidden key, but also like a conventional signature with a normal private key. The idea for the solution is that a transformation on the private key is needed, which results in a hidden private key. The original private key is stored at the user's trusted computer and the agent only has access to the hidden private key. It is clear, that it must not be possible to calculate the private key directly from the hidden private key.

In several applications, such as electronic voting [13] and anonymous electronic cash [1] [12], the idea of computing with secret data is already well established, but is not always seen as such. These applications are based on blind

signatures, first introduced by Chaum [3]. A blind signature allows a person to let somebody else digitally sign a message, without this signer knowing the content. The secret in this signature function is the original message. In order to make this possible, the message is blinded and this is used by the signer to perform the signature operation [4].

In agent technology this idea of blinding data can be used. Instead of blinding the message, the private key should be blinded. This signature will then exist out of the following steps:

1. Key generation
2. Blinding operation on private key
3. Signature operation
4. Activation of signature
5. Verification

Steps 1, 3 and 5 are necessary in any conventional digital signature algorithm. Step 2 is the transformation from private key into a blinded version and in step 4, the signature is activated. This step is necessary because in step 3 a signature is set using a blinded private key. This signature cannot yet be verified by using the agent's public key, because the blinded private key and the agent's public key are not related as such. Hence an activation procedure must be added.

Steps 1 and 2 must be performed in a trusted environment, e.g. the user's computer. Step 3 and 4 are done at the foreign host . Finally the verification can be done anywhere in the system.

3. AGENT DIGITAL SIGNATURE

3.1 Introduction

The various digital signatures as presented in this paper are based on the elliptic curve digital signature algorithm [8]. A small modification, similar to the one in [2] is done in order to be able to transform the digital signature function into one where a parameter is private.

The security in elliptic curve cryptographic systems is based on the hardness of the discrete logarithm problem for elliptic curves. This problem can be informally described as follows [15]: Given two points P and Q on an elliptic curve such that $Q = dP$, find the integer d. This problem is generally believed to be infeasible in case the space in which d is chosen, is large enough. Certain curves are believed not so secure, such as supersingular curves [11]. To provide clarity in notation, for a point on the curve a capital letter is used and integers are shown in lowercase letters.

In a conventional system, the digital signature exists out of three steps: key generation, signature operation and signature verification (steps 1,3 and 5 in paragraph 2). The signer owns two kinds of keys; the private and public key. The private key, which is only known to the signer is used to sign a digital message and the public key is used to verify the validity of the signature [15]. In practice, the public key is certified by a Trusted Third Party (TTP), such that the public key is connected to an identity. Hence, the verification of the signature using the public key can proof who signed the document and this entity can not deny its signature afterwards. This property provides non-repudiation.

1. **Key generation**
   The key generation starts with the signer choosing an elliptic curve E defined over Z_p. The number of points on $E(Z_p)$ should be divisible by a large prime n. The signer selects a point P on $E(Z_p)$ of order n and selects a random integer d in the interval $[1, n-1]$. Parameter d is the private key of the signer and must be kept secret. Using the private key, the signer can compute its public key:
   $$Q = dP \text{ over } E(Z_p). \quad (1)$$
   The public key is formed by the parameters (E, n, P, Q)
2. **Signature generation**
   In order to sign a message m, the signer selects a random secret integer $k \in [1, n-1]$ and computes:
   $$R = kP = (x_0, y_0), \quad (2)$$
   $$r = x_0 \bmod n,$$
   $$s = km + rd \bmod n. \quad (3)$$
   The signature exists out of the parameters (r, s) and this is sent to the verifier in combination with the message m. Parameter k must be different for each signature based on one private key d. In case k is equal for two different messages, d can easily be computed [5].
3. **Verification**
   By obtaining the right signature parameters, public key and digital signature, the verifier can check the validity of the signature by computing:
   $$T = (sP - rQ)m^{-1} = (x_1, y_1) \quad (4)$$
   $$t = x_1 \bmod n$$
   The signature is valid if and only if:
   $$t = r$$

3.2 Agent digital signature

In case a software agent needs to sign a document, the agent cannot follow the above procedure, because there are some fundamental differences:

- The agent is the signer of the document, but does not own the resources to be able to compute the signature.
- The signer does not know whether it is located in a trusted environment.

In case the agent lets the host compute a signature for it, using the algorithm described in 3.1, the host would have access to the agent's private key d and that gives it the possibility to sign other messages out of the agent's name or it could pretend to be the agent (during authentication). In this case the property of non-repudiation would not be present anymore, because multiple entities have now access

to the agent's private key, hence there is no guarantee that the agent signed the document.

As is described in the solution outline, the idea is to sign a document using the hosts resources by using an agent's hidden private key. After the signature is computed, the host activates the signature.The user generates several blinding factors, which hide the private key. These factors are then needed to activate the signature. This can be accomplished by storing a part of the blinding factors in the agent and a part securely at the user's computer. How this can be achieved such that the activation can take place at the host and the private key cannot be computed by the host, is shown in the description of the algorithm.

1. **Key generation**
   An elliptic curve is defined over Z_p, of which the number of points on $E(Z_p)$ is divisible by a large prime n. The user selects a random integer d in the interval $[1, n-1]$. Parameter d is the agent's private key and is securely stored at the user's computer. The user computes the agent's public key:

   $$Q = dP \text{ over } E(Z_p). \tag{5}$$

   The public key is formed by (E, P, n, Q) and this is stored in the agent and at the user's computer. Besides calculating a regular public key, the user computes a "temporary" public key:

   $$\delta = d\gamma \bmod n, \tag{6}$$
   $$\Gamma = \delta P, \text{ over } E(Z_p). \tag{7}$$

   and here γ is a blinding factor and δ can be seen as a temporary private key. The parameters γ and d are stored securely at the user and not given to any other element in the system. Parameter Γ is also stored in the agent for verification purposes.

   Two extra parameters, α and λ, are chosen at random in the interval $[1, n-1]$ and the following is computed:

   $$c = \alpha\gamma \bmod n, \tag{8}$$
   $$\Lambda = \lambda cP \text{ over } E(Z_p). \tag{9}$$

   In the next steps it will become clear why these parameters are necessary. The parameters Λ and c are stored in the agent, while α, γ and λ are kept secret at the user's computer.

2. **Blinding operation on private key**
   This step, the hiding of the private key, is completed at the user's computer. In order to obtain a blinded private key, the user selects one other blinding factor β at random in the interval $[1, n-1]$. As in the digital signature algorithm, the user also selects a parameter k at random in $[1, n-1]$ and computes:

   $$\tilde{R} = kP = (x_0, y_0), \tag{10}$$
   $$\tilde{r} = x_0 \bmod n,$$
   $$R = c\tilde{R} + \beta P = (x_1, y_1,) \tag{11}$$
   $$r = x_1 \bmod n,$$
   $$\tilde{d} = \alpha^{-1}d + \lambda \bmod n, \tag{12}$$

   in which $\tilde{d}$ is the blinded private key. The blinding factors α, λ and γ must be kept secret at the user's computer, just like the agent's private key. Parameters $\tilde{d}, k, r, \beta$ and c are stored in the agent and therefore known to the host. These parameters in combination with the system parameters give the agent the opportunity to sign a document m while located at a foreign host and using its computational resources without letting this host having access to its private key. Also, knowing these parameters, d cannot be computed by the host.

3. **Signature generation**
   During the signature generation, the agent is located at the host. The signature on message m is then computed by:

   $$\tilde{s} = km + r\tilde{d} \bmod n \tag{13}$$

   It can be seen in equation (13) that the signature operation that the host must execute for the agent is equal to the signature operation in equation (3). It only differs in the fact that in (13) the hidden private key is used instead of the original private key.

   In paragraph 3.1 it is said that parameter k must be kept secret in order to prevent the revealing of the private key. Here, that would mean that the host must either not have access to it or the user must trust the host not to abuse this knowledge. Fortunately, this does not matter here, because the private key is not used. By using the same k twice, the host would not gain any more knowledge about the private key.

4. **Activation of signature**
   Because some of the blinding factors are stored in the agent, the host is able to transform the signature towards a valid signature. With valid, it is meant that the signature must be verified using the agent's public key as is registered at a Trusted Third Party (in case a PKI is used). The activation of the signature can be performed by computing:

   $$s = c\tilde{s} + \beta m \bmod n. \tag{14}$$

   Parameters c and β are known by the host, but this is not sufficient to compute d or γ. Substitution of parameters gives the following signature:

   $$s = (\alpha\gamma k + \beta)m + \gamma r d + \alpha\gamma r\lambda \bmod n. \tag{15}$$

   From (15), it can be seen that this signature is of the same form as (3). Out of (15) it is seen that it is not important whether k is kept secret or not. In case k is known and kept at the same value for multiple signatures, it depends still on the factor γ whether d can be calculated. Because the factor λ is unknown by the host, it is impossible for the agent platform to calculated d or δ. Hence, neither the private key or the temporary private key can be computed.

   However, it is possible for the host to compute $\epsilon = \gamma d + c\lambda$, but during the verification process it will be shown that this does not make it less secure. In case parameter Λ would not be used, e.g. no λ would occur in (15), it would be possible for the host to calculate the temporary private key δ.

5. **Verification of signature**
   The verification formulas are the same as in a conventional digital signature algorithm based on elliptic

curves, only now the temporary public key must be used in combination with parameter Λ:

$$T = (sP - r(\Lambda + \Gamma))m^{-1} = (x_1, y_1) \quad (16)$$
$$t = x_1 \bmod n$$

The signature is valid if and only if:

$$t = r.$$

For the verification process it is important that Γ and Λ are given to the verifier as two distinct parameters instead of $(\Lambda + \Gamma)$, because for the host it is possible to calculate $\epsilon = \gamma d + c\lambda$ and hence $\epsilon P = \Lambda + \Gamma$, but the host cannot calculate γd and $c\lambda$ separately and therefore cannot pretend to be the agent.

By introducing this temporary public and private key, something extra is also achieved besides making it possible to activate the signature at the host. This temporary key pair can be seen as a pseudo-identity of the agent. Parameter Γ must then be registered at the Trusted Third Party (TTP), just as Γ. Giving the agent multiple temporary key pairs is actually giving the agent more identities. Hence, this algorithm can be seen as a privacy enhancing technology [7] for agent specific applications. Using these types of pseudonyms is an advantage over the simple solution of registering a temporary public key with the TTP and using the corresponding private key without blinding it, because the host cannot compute the temporary private key in the above proposed solution, hence the host or another agent cannot pretend to be an agent it is not. Depending on the amount of pseudonyms an agent wishes, extra overhead is added to the TTP for key distribution and revocation.

Out of the parameters known by the host, it is impossible to calculate the private key, because of the hardness of the discrete logarithm problem for elliptic curves. Therefore, the identity of the agent, d, is protected. However, this algorithm does not give control to the user about what the agent signs or how many times the host executes this algorithm. For the host it is possible to repeat the algorithm with different messages and all the signatures will be valid. This drawback makes this algorithm only suitable in a trusted environment. It does not mean it is useless. On the contrary, it is preferred to the conventional digital signature in trusted environments, because the private key is not revealed at any time. That means the host can sign out of the agent's name, but cannot pretend to be the agent. The problem of multiple signing can be compared to the double spending problem in applications as digital anonymous cash [1]. Several solutions can be found to this problem, of which two are presented in the next paragraph.

3.3 Agent digital signature and solutions to double signing problem

An idea to prevent hosts from performing an agent signature multiple times, is to include the host's identity in the verification of the signature. Each time a signature is verified, the verifier can see at what location the document has been signed. This solution does not make the double signing operation impossible, but it will be an extra threshold to do so. Two algorithms are proposed in this paper to accomplish the idea. The first is one without a signature from the host, only its identity is added to the verification formula. The second algorithm gives two signatures on the message.

3.3.1 *Agent signature combined with host's identity*

The host's identity must be added in the verification formula. In order to obtain this, the public key is added to this formula. This means that the public key or the private key must also be added in the signature. This solution does not include a host's signature and therefore the host's public key, which represents its identity, is added during the blinding operation on the agent's private key. Adding the host's identity must occur at the user's, because this will make it impossible for the host to change its identity on the signature at a later stage.

1. **Key generation**
   As in the previous algorithms, the key generation starts with selecting an elliptic curve E over Z_p, of which the number of points on $E(Z_p)$ is divisible by a large prime n. The user selects a random integer d_a in the interval $[1, n-1]$. Parameter d_a is the agent's private key and is securely stored at the user's computer. The user computes the agent's public key:

   $$Q_a = d_a P \text{ over } E(Z_p). \quad (17)$$

   Besides calculating a regular public key, the user also selects the first blinding factor α and computes a "temporary" public key and as in the previous paragraph parameter Λ:

   $$\delta = \gamma d_a \bmod n, \quad (18)$$
   $$\Gamma_a = \delta P \text{ over } E(Z_p), \quad (19)$$
   $$c = \alpha\gamma \bmod n, \quad (20)$$
   $$\Lambda = \lambda c P \text{ over } E(Z_p). \quad (21)$$

   and here γ is a blinding factor and the combination γd_a can be seen as a temporary private key. The parameters α, γ, λ and d_a are stored securely at the user and not given to any other element in the system.

   In this algorithm, also the host generates a key pair:

   $$Q_h = d_h P \text{ over } E(Z_p), \quad (22)$$

   where d_h is the host's private key and Q_h is its corresponding public key.

2. **Blinding operation on private key**
   This step is equal to the blinding operation in the previous paragraph. Only to the parameter R, the host's identity is added in the form of its public key Q_h:

   $$\tilde{R} = kP = (x_0, y_0), \quad (23)$$
   $$\tilde{r} = x_0 \bmod n,$$
   $$R' = \alpha\gamma\tilde{R} + \beta P \quad (24)$$
   $$R = R' + Q_h = (x_1, y_1) \quad (25)$$
   $$r = x_1 \bmod n,$$
   $$\tilde{d_a} = \alpha^{-1} d_a + \lambda \bmod n. \quad (26)$$

   The parameters that are stored in the agent and therefore known to the host are $r, \tilde{d_a}, k, c$ and β and for verification purpose Γ.

3. **Signature generation**
   Again the signature operation is equal to (3), with the exception that d is replaced by $\tilde{d}_a$:

$$\tilde{s} = km + r\tilde{d}_a \bmod n. \quad (27)$$

4. **Activation of signature** The activation does not involve the host's identity, and therefore is equal to the activation in the previous algorithm:

$$s = c\tilde{s} + \beta m \bmod n. \quad (28)$$

   Again parameters c and β are known by the host, but this is not sufficient to compute d_a or γ. Substitution of parameters gives the following signature:

$$s = (\alpha\gamma k + \beta)m + \gamma r d_a + \alpha\gamma r\lambda \bmod n. \quad (29)$$

   Again the signature is of an equal form as (3).

5. **Verification**
   The idea was to add the host's identity in the verification formula, such that it is always possible to know where the signature operation was executed. Adding the host's identity is possible, because the public key of the host is already used in the blinding operation:

$$\begin{aligned} T &= (sP - r(\Lambda + \Gamma_a))m^{-1} + Q_h = (x_1, y_1) \quad (30) \\ t &= x_1 \bmod n \end{aligned}$$

   The signature is valid if and only if:

$$t = r.$$

This algorithm has the advantage that the host's identity is attached to the agent's signature, which makes it for the host less attractive to sign documents in the agent's name without having permission for it.

A disadvantage, however, is that the agent must know the identities of the hosts it is planning to visit. An easy measure to overcome this is by storing several R' parameters in the agent and before roaming to another platform, it lets the current host add the next host's identity to form parameter R. The signature as proposed here is only from the agent and not the host, because the host's private key is not used.In the next paragraph it is shown how this can be achieved.

3.3.2 *Combined agent and host signature*

This signature is similar to the previous one. In various stages extra information is added about the host, such that also the host signs the document.

1. **Key generation**
   As in the previous algorithms, the key generation starts with selecting an elliptic curve E over Z_p, of which the number of points on $E(Z_p)$ is divisible by a large prime n. The user selects a random integer d_a in the interval $[1, n-1]$. Parameter d_a is the agent's private key and is securely stored at the user's computer. The user computes the agent's public key:

$$Q_a = d_a P \text{ over } E(Z_p). \quad (31)$$

   Besides calculating a regular public key, the user also computes the "temporary" public key for the agent and the parameter Λ:

$$\begin{aligned} \delta &= \gamma d_a \bmod n, & (32) \\ \Gamma_a &= \delta P \text{ over } E(Z_p), & (33) \\ c &= \alpha\gamma \bmod n, & (34) \\ \Lambda &= \lambda c P \text{ over } E(Z_p). & (35) \end{aligned}$$

   and here $\gamma \in [1, n-1]$ is a blinding factor and δ can be seen as a temporary private key for the agent. Again, the parametersα, γ, $d_a$and λ are stored securely at the user and not given to any other element in the system.

   In this algorithm, also the host generates a key pair:

$$Q_h = d_h P \text{ over } E(Z_p), \quad (36)$$

   where d_h is the host's private key and Q_h is its corresponding public key.

2. **Blinding operation on private key**
   This step is equal to the blinding operation in the previous paragraph. Only to the parameter R, the host's identity is added:

$$\begin{aligned} \tilde{R} &= kP = (x_0, y_0), & (37) \\ \tilde{r} &= x_0 \bmod n, & \\ R' &= \alpha\gamma\tilde{R} + \beta P & (38) \\ R &= R' + Q_h = (x_1, y_1) & (39) \\ r &= x_1 \bmod n, & \\ \tilde{d}_a &= \alpha^{-1} d_a + \lambda \bmod n, & (40) \\ c &= \alpha\gamma \bmod n. & (41) \end{aligned}$$

   The user also computes a temporary public key for the host:

$$\Gamma_h = cQ_h \text{ over } E(Z_p). \quad (42)$$

   The host can check whether the right public key is used, by performing the same operation as in (42). The parameters that are stored in the agent and therefore known to the host are $r, \tilde{d}_a, k, c, \Gamma_h$ and β and for verification purpose Γ_a.

3. **Signature generation**
   In this step the signature operation is executed at the host and the signature should involve the private keys of the agent and the host. This can be accomplished by the following operation:

$$\tilde{s} = km + r\tilde{d}_a + rd_h \bmod n. \quad (43)$$

4. **Activation of signature**
   The activation does not involve the host's identity, and therefore is equal to the activation in the previous algorithm:

$$s = c\tilde{s} + \beta m \bmod n. \quad (44)$$

   Parameters c and β are known by the host, but this is not sufficient to compute d_a or γ. Substitution of parameters gives the following signature:

$$s = (\alpha\gamma k + \beta)m + \gamma r d_a + \alpha\gamma r d_h + \alpha\gamma r\lambda \bmod n. \quad (45)$$

   Again the signature is of the same form as (3), only now signed by two parties.

5. **Verification** The verification formula is here a little different, because also the private key of the host is used to sign the message:

$$T = \frac{sP - r(\Gamma_a + \Lambda + \Gamma_h)}{m} + Q_h = (x_1, y_1) \quad (46)$$
$$t = x_1 \bmod n$$

The signature is valid if and only if:

$$t = r.$$

This algorithm, like the previous one, makes it less attractive for a host to sign messages in the agent's name without having its permission. Here it is accomplished that both the agent and the host signed the document and the host cannot deny afterwards that it signed this document.

4. CONCLUSIONS AND DISCUSSION

In this paper, several solutions are presented for a special case of computing with secret data in public, namely the digital signature for agents. The secret data in agent digital signatures is the private key, as is this the information that proofs the agent's identity. The agent should not reveal this information to anyone, not even a trusted host.

Several agent digital signatures are described in this paper, all based on one principle; the private key is transformed to a blinded private key. The blinded private key is stored in the agent and the original private key is stored securely at the user's computer. The first described digital signature for mobile agents has the problem that the host is capable of signing multiple messages in the agent's name without having its permission. This agent signature can also be seen as a new privacy enhancing technology, as pseudo-identities can easily be introduced here, without adding extra complexity of the scheme.

Two solutions are proposed to prevent the problem of double signing, such that the problem is not solved, but it makes it less attractive to the host to sign in the agent's name without having its permission. The first of these solutions use the host's identity at time of signature verification, such that the verifier can locate where the signature operation has been executed. The second solution let not only the agent sign the message, but also the host. This method makes it impossible for the host to deny the signature afterwards.

The advantage of all the proposed algorithms is that they can be used specific for agents (where the private key is hidden), but also for non-agent systems, where the computation is done in a trusted environment. In the latter case, the private key is simply not hidden and the rest of the algorithms is the same.

Some solutions for a digital signature in agent technology are proposed in this paper, but this is only a first step towards securing intelligent software agents. The next step in these digital signatures would be to provide non-repudiation not only for the host but also for the agent. Non-repudiation for the agent is not provided yet, because in case a malicious host has the purpose to sell products, it can double sign an order and the agent is not capable of proving whether it gave permission for this order, or it cannot be proven that the agent gave its permission, because in this case it's in the host's advantage to have its name on the signature. A solution must be found to this problem in order to provide full functionality of a digital signature.

5. REFERENCES

[1] S. Brands. *Untraceable off-line cash in wallet with observers.* Advances in cryptology - Crypto'93. Springer-Verlag. pp. 302-318.

[2] J.L. Camenisch, J-M. Piveteau, M.A. Stadler. *Blind signatures based on the discrete logarithm problem.* Advances in Cryptology - Eurocrypt'94, Springer-Verlag, pp 428-32.

[3] D. Chaum. *Blind signatures for untraceable payments.* Advances in Cryptology - Crypto'82, Springer-Verlag, pp 428-432.

[4] D. Chaum. *Achieving electronic privacy.* Scientific American. August 1992. pp 96-101.

[5] T. ElGamal. *A public key cryptosystem and a signature scheme based on discrete logarithms.* IEEE Transactions on Information Theory. Vol. IT-31, No. 4. July 1985. pp. 469-72.

[6] U. Feige, J. Kilian, M. Naor. *A minimal model for secure computation.* STOC 94 Montreal, Canada. ACM. pp.554-563.

[7] R. Hes, J. Borking. *Privacy-Enhancing Technologies, The path to anonymity.* Achtergrondstudies en Verkenningen 11, Registratiekamer, The Hague, 2000.

[8] D.B. Johnson, A.J. Menezes. *Elliptic curve DSA (ECDSA): An enhanced DSA.* http://www.certicom.com/research.html

[9] S. Loureiro, R. Mova. *Privacy for mobile code.*Proceedings of the Distributed Object Security Workshop, OOPSLA'99, Denver, US, November 1999.

[10] P. Kotzanikolaou, M. Burmester, V. Chrissikopoulos. *Secure transactionswith mobile agents in hostile environments.* Information security and privacy. Proceedings of the 5th Australasian conference. ACISP2000. Springer-Verlag, 2000. Volume 1841. pp. 289-297.

[11] A.J. Menezes. *Elliptic curve public key cryptosystems.* Kluwer Academic Publishers, 1993.

[12] S. Miyazaki, K. Sakurai. *A more efficient untraceable e-cash system with partially blind signatures based on the discrete logarithm problem.* Financial cryptography'98. Springer-Verlag. pp. 296-308.

[13] A. Riera, J. Rifà, J. Borrell. *Efficient construction of vote-tags to allow open objection to the tally in electronic elections.* Information Processing Letters 75. Elsevier, 2000. pp. 211-215

[14] T. Sander, C. F. Tschudin. *Towards mobile cryptography.* IEEE symposium on Security and privacy. 1998. pp. 215-224.

[15] B. Schneier. *Applied cryptography, second edition.* John Wiley & Sons, Inc. 1996.

An Intrusion Response Scheme: Tracking the alert source using a stigmergy paradigm

Noria Foukia
Centre Universitaire d'Informatique
University of Geneva
24, rue du Général Dufour,
Switzerland
CH-1211 Geneve 4
noria.foukia@cui.unige.ch

Salima Hassas
L.I.S.I.
Nautibus, 8 Bd Niels Bohr
Campus de la Doua, 43 Bd du 11 Novembre
69622 Villeurbanne, France
hassas@bat710.univ-lyon1.fr

Serge Fenet
L.I.S.I.
Nautibus, 8 Bd Niels Bohr
Campus de la Doua, 43 Bd du 11 Novembre
69622 Villeurbanne, France
sfenet@bat710.univ-lyon1.fr

ABSTRACT

Today, the security community is in search of novel solutions to achieve efficient responses to intrusions. This is particularly needed because attackers intervene in an automated way, at computer speed. There also is a need to respond according to the nature of the detected attack. That is why Intrusion Detection Systems (ID Systems) and Intrusion Response Systems (IR Systems) have to cooperate and work in parallel. To this end, it is more efficient to design the IR System in function of the ID System. This paper describes an IR System based on Mobile Agents (MAs) distributed throughout the network. This IR System is strongly adjustable to its partner ID System, also based on MAs. Both the ID System and the IR System are designed in quite similar ways, since both are mappings of the behavior of natural systems. We present our approach to building these two systems based on natural life. We particularly stress the design of our IR System and present some simulations to demonstrate its efficiency.

Keywords

Intrusion Detection and Response Systems, Mobile Agents, Natural Systems.

1. INTRODUCTION

As attacks against information systems are growing and becoming more and more sophisticated, there is a need to intensify research on responding to intrusions. This statement is in line with a report issued by the *Computer Emergency Response Team* (CERT), which announces each year an increase of computer security incidents and, in many cases, a complete lack of measures to counter them [1]. This is principally due to the fact that people in charge of security focus their attention essentially on the deployment of Intrusion Detection Systems (ID Systems) and control systems, without dealing with the preparation of intrusion responses. Formerly, this approach was justified as long as the number of elements to supervise (devices, connections) throughout the networks was not too large. It was quite sufficient to let a system administrator intervene manually to stop an attack in a small scale network without harming too much the global integrity of the system. However, today, the size of information networks has grown considerably and, even if a given ID System is quite effective, it is not reasonable to let a human administrator proceed manually to stop an attack, to evaluate the extent of damage and to regenerate a safe state in accordance with the security policy in force. The more reasonable solution is to have corrective or defensive actions generated automatically as soon as a suspicious activity is detected, and not to be delayed by the reaction time of a human operator.

Our research work encompasses both Intrusion Detection (ID) and Intrusion Response (IR) and follows in its principles the behavior of natural systems:

- For the detection, the human natural immune system provides a source of inspiration for today's computer security when building ID Systems, because the immune systems evolves many interesting mechanisms to defend our body against external attacks and aggressions.
- For the response, we also took our inspiration from a social insects paradigm, namely the collective behavior of foraging ants. This behavior expresses the way ants gather food collectively: they use an indirect communication mechanism where each ant's motion is influenced by a chemical information (called *pheromone*) deposited by the other ants in the environment. Our Intrusion Response System (IR System) maps the collective behavior of a population of foraging ants, using mobile agent (MA) technology and an electronic version of pheromone.

In this paper we describe an Intrusion Detection and Response System (IDR System), combining both of these mechanisms and we focus on the response scheme to intrusion in local networks using MA technology. This IDR System is based on a social insects paradigm to trace the source back to where the intrusion alert was generated. We specially study and describe the tuning of such a mechanism, to allow an effective scheme for intrusion response. Section 2 provides an overview of the different kinds of IR Systems and related work. Section 3 presents our approach for intrusion response based on MA technology as well as the social insects paradigm. We show in this section how systems inspired by nature provide a suitable paradigm for IDR Systems. Section 4 describes the simulation performed with Starlogo [2] to validate our algorithm for tracing the route

back to the source of an alert and to evaluate the effectiveness of established parameters; the results are also summarized in this section. Section 5 investigates future work and draws a conclusion.

2. RELATED WORK IN INTRUSION RESPONSE

Fortunately, today the security community seems to realize the lack of efficient intrusion response schemes and some of its members are starting to investigate possible solutions. These research activities are based on the use of either static or mobile components, the latter offering the widest possibilities.

2.1 Static Systems

Static systems were initially built in the frame of centralized ID Systems. Only the emergence of distributed architectures gave way to mobile components. However, some distributed systems implement a response scheme based on static software components. This is for instance the case in [3] where a technique for adapting responses to intrusions is proposed. Such a system has one or more intrusion detection systems that identify intrusions. Separate modules classify the nature of the intrusion and determine appropriate responses based on an intrusion response taxonomy. This taxonomy refers to a previous work described in [4], which provides a categorization of possible offensive and defensive responses. As also reported in [3], a number of systems have been developed for responding to intrusive actions. They are classified as notification systems, manual response systems or automatic response systems. Table 1 borrowed from [3] summarizes this classification of static response systems.

Intrusion Response Classification	Number of Systems
Notification	31
Manual Response	8
Automatic Response	17
Total	56

Table 1: Classification of Static Response Systems

Notification systems only generate a report or an alarm and wait for the system administrator to provide the answer. The manual response systems give the administrator the capability to launch a manual response from a limited pre-programmed set of responses. Automatic response systems immediately answer to an attack with pre-programmed responses. In this case, a taxonomy such as the one mentioned in [4] will lead to an indication of generic response categories and is useful to launch automated responses. As far as we are concerned, we are targeting automated response schemes where *Intrusion Response Agents* (IR Agents) identify the kind of alert reported in the pheromone and execute an automated process to undertake corrective actions. Meanwhile, our architecture is based on a population of MAs interacting in the network environment. As we will see now, the use of mobility brings a lot of advantages from a security point of view.

2.2 Systems Based on Mobile Components

In traditional static systems, the ID System is easier to locate and can itself be subject to a successful attack that could leave the entire network vulnerable. A thorny problem is hence to ensure above all the security and the robustness of the ID System itself. By enabling stealth operation and dynamic reconfiguration, MAs can be a good solution to this problem. A MA is a piece of code that can run on one host, perform a transparent migration to another host, and resume its running state. While visiting the network in an autonomous manner, MAs can interact with each other. In order to accomplish their task, MAs can also gather data and use services present on visited hosts. To date, only a few investigations have been undertaken to develop MA-based response schemes to intrusions [5] [6], even if MA technology seems to exhibit good properties to accomplish this task. As in [5], we advocate the recourse to MA technology for supporting the answer to intrusions rather than exclusively for detecting intrusions.

Uesful MA characteristics that could be retained for the answer are:

- the rapidity of execution due to the small quantity of code the MA represents when implemented efficiently. This rapidity is particularly desirable to answer to the attack as soon as possible.
- the ability to adjust their execution code depending on the characteristics of the machine they visit. This factor is also quite important since it enables MAs to adjust the defence parameters to better protect the system.
- the mobility, which is their main property. Because MAs can travel across the network they can filter relevant information from the different machines. They have the ability to correlate all this information and adapt their answer. It could be helpful e.g. if an attack comes from several sources or if an attack reaches several destinations. They can also take advantage of their migratory ability in order to limit potentially risky interactions with entities suspected for being offensive.

We can notice that these characteristics are similar to those we retained for the use of MAs to deal with intrusion detection, as already mentioned in [7].

The following section gives a classification of the different types of responses to an intrusion.

2.3 Different Degrees of Responses

Two main degrees can be distinguished in the response process, namely passive and active responses.

2.3.1 Passive Responses

A response can happen before an attack really begins because there are already some indications that the system is becoming vulnerable. It is an anticipated way of setting the system on the defensive without disturbing too much its operation. For instance, a user can still access his account, but some file accesses are limited because the system is checking the integrity of these files. Responses can also be implemented after the attack has occurred. One can for example attempt to repair the damage due to the attack, gather related informations from heterogeneous sources, or try to avoid the repetition of the same attack in the future. These patterns are easier to implement, since the time as well as the speed of the answer are not essential; the attack

has either not yet occurred, or is already finished (with all the following consequences, of course).

2.3.2 Active Responses

This type of response encompasses all actions that can limit the damage due to a running attack or stop it. Many measures can be implemented, like automatically generating firewall rules to block an incoming data flow, closing open ports, updating internal routing tables, etc. This kind of "real-time" response is not hard to implement when dealing with a human attacker using interactive tools. However, modern attack tools perform automated actions that often take only some seconds to penetrate a system. Moreover, actual response mechanisms lose their effectiveness against multiple distributed attacks originating from many forged IP addresses. As it is suggested in [8], infrastructures that support development of automated response systems are critically needed. Again, MAs could bring some solutions. As mentioned by [5], they offer the ability to intervene on all network components, and not only on the machines involved in the security policy.

3. OUR APPROACH AND MODEL

This section focuses on the design goals we retained for the intrusion detection and response model. The model has been designed to allow:

- intrusion detection based on Intrusion Detection Agents (ID Agents), which map the functionalities of the natural immune system to distinguish between normal and abnormal events (respectively "self" and "non self" in the immune system) as explained in [9].

- intrusion response based on IR Agents, which map the collective behavior of an ants population by triggering throughout the network the release of a synthesized information specific to the collected events. This kind of collective paradigm is very interesting because it consists in having each ant execute a rather light task (MAs play the role of ants in the IR System) to induce collectively a more complex behavior; this will be explained in further details in section 3.2.

Our approach also is very powerful because the ID System as well as the IR System are completely distributed in the network, without any centralized control: both systems are essentially constituted by MAs which travel across the network, dynamically adjusting their routes according to collected events, without any simple way to trace them. Besides, our MAs are quite polyvalent because they can detect and/or respond to intrusion. This enhances the difficulty for an attacker to distinguish between ID Agents and IR Agents.

We are principally dealing with responding after the attack is detected. Our ID System looks for behavioral deviations in running applications. A strong deviation is a safe sign that an attack is occurring and will trigger the response mechanism. In addition, our ID System is able to check the vulnerability of applications. By adjusting the level of acceptable deviations, we can also choose to detect processes with a low deviation. For instance, the presence of too many processes running with a low deviation is an indication that the system is not stable and potentially attackable. In response, some anticipated actions could be executed to avoid future attacks.

Let us now concentrate on a target host (device, user account,...) that is suffering an attack. In our IR System, the response consists in intervening where the source of the alert is located by diffusing the alert in the neighborhood of the target. We also partially look for the origin of the attack insofar as the intruders are inside the network, because it is easier to isolate them. For intruders coming from outside, we take the point of view that it is not necessary to directly react against them as long as we are able to block their access point to the inside of the network.

3.1 Background Concerning our ID System Model

As we already mentioned, our IDR System combines two mechanisms inspired from natural systems, namely the immune system for the intrusion detection step and the social insects paradigm for the intrusion response one. Indeed, we were really struck by the fact that natural systems exhibit many interesting characteristics which could be transposed to distributed networks in general, and to face security problems in particular. Natural systems are complex systems, endowed with mechanisms allowing them to react efficiently to any perturbation coming from their environment, by adapting themselves to these changes.

3.1.1 The Immune System Paradigm

To defend the body against undesired organisms, the immune system distinguishes between molecules and cells of the body called "self" from foreign ones called "nonself". The body's immune defences normally coexist peacefully with cells that carry distinctive "self" markers. But when immune defenders encounter "nonselfs" they have to eliminate them quickly, to ensure some kind of body integrity.

3.1.2 The Social Insects Paradigm

In the same way, social insects organize themselves to ensure the survival of the colony by means of a reactive individual behavior and a cooperative collective one. Such behaviors can be observed from different activities: foraging, building nests, sorting larvas,...etc. Cooperation in these systems is mediated by an efficient communication mechanism relying on the inscription of task evolution in the environment. This paradigm, introduced for the first time by P. P. Grassé in [10], described the way social insects communities (ants, termites, bees, ...) interact through their environment. Schematically, each entity has a local view of its neighborhood but uses a chemical volatile substance (the pheromone) to mark its environment when achieving a collective task. The pheromone thus deposited is propagated in the environment and evaporates with time. The deposit of pheromone creates a gradient field in the environment which tends to attract other insects, and to enroll them in a self-catalytic behavior; it follows that the task is completed, and moreover that other insects are recruited long as the pheromone is present in the environment. When the task is finished, no more pheromone is deposited, leading to the disappearance of this information after a period of time through an evaporation mechanism. Each time an ant deposits a pheromone along the path, it reinforces the probability that other ants will choose the same path to reach

the food. The amount of deposited pheromone is called the *pheromonal gradient*, and every ant scanning its neighborhood will walk up the gradient. This indirect communication between the different members of the colony through the environment is called *stigmergy*. This paradigm has inspired many computer scientists across various research domains such as robotics [11], network routing [12] and optimization algorithms [13]. In all these cases the global and complex collective behavior emerging from interactions between simple entities is dominating. That is the reason why we think that we can map this mechanism to model a collective response to an intrusion.

3.2 Mapping these Paradigms into our Model

3.2.1 The Detection Step

Like the human body, computers systems have to protect themselves because they are often placed in an unsafe and uncontrolled environment such as the open Internet. In a first step, the immune system attempts to prevent or stop the entry of external organisms before they penetrate the body. This is the same role as played by firewalls in the computer world; firewalls attempt to limit access of undesired users and processes coming from outside the network they are protecting. In a second step, the immune system seeks the presence of undesired organisms in the body in order to destroy them.

The idea in our work was to map the "self"–"non self" detection in the immune system with intrusion detection. Our approach is targeted at corporate intranets, which corresponds to logical security domains. We subdivide an intranet into several smaller local domains constituted by a set of hosts or machines. We want to avoid a monolithic ID System on every host because of its cost; instead, we propose to dispatch MAs, dynamically visiting and randomly monitoring different local domains. To detect local attacks, these ID Agents, responsible for a local domain, have to be able to discriminate between normal and abnormal activity. In the immune system, this is done by distinguishing "self" from "nonself" entities. For the sake of simplicity, we choose to examine the correct execution of different programs and their deviation compared to a normal activity. For that, ID Agents dispatched throughout the network collect application-specific system calls and compute the deviation between these system calls and other safe system calls stored in a database, as done by Forrest and al. [9]. If the deviation is too high, an alert is launched, waiting for the response mechanism to come into play.

3.3 IR System Deployment and Tuning: Source Tracing

The network is a distributed environment, subject to perturbation and dynamic evolution. When an attack is detected on the network, it is not trivial to locate efficiently and rapidly its source. The behaviour of foraging ants seems to provide an interesting solution to this problem. In nature, they release in their proximity a chemical information (the already mentioned pheromone) to trace the way from their nest to the location of some food; we use this metaphor to allow MAs (artificial ants) to detect the location of an alert and to follow up to the source of the attack in order to answer to the attack.

3.3.1 Building the Electronic Pheromone

An agent surviving an attack must escape as soon as possible from the attack location, but should find a way to tell IR Agents where the source of the attack was located. In fact, identifying the source is quite fundamental to put together a response plan adapted both to the nature of the attack and to the kind of incident reported. For instance, if the source is a firewall and a service is misused, the response to the attack could be to change the access rights to the service protected by the firewall; if the source is a mail server and a user is flooding the server with a series of e-mails, the response could be to disable the user account in this server. As the quality of the response depends on the rapidity to trace the source of the alert, we want to take advantage of being MAs completely distributed in the network to rapidly intervene. An alert message is initiated at a node as soon as a local ID Agent present in the node detects an anomaly. For this locally detected attack, the ID Agent creates a so-called *pheromonal message* which is randomly launched across the network and will help other IR Agents in the system to trace the way back to the alert source. The ID Agents dispatched through the network are able to launch an alert and to build and disseminate an electronic pheromonal information synthesizing the attack scenario for other IR Agents. The IR Agents, completely distributed in the network, can track this pheromone and travel up the pheromonal gradient back to the source. For the moment, the different fields which compose the synthesized pheromone are gathered in a list as follows:

- the identifier of the ID Agent which detects the suspicious activity and builds the pheromone: the IdA;
- the suspicion index of the alert: the SI. The proposed response scheme depends on a behavior-based ID System scheme depicted in [7]. In [7], ID Agents dispatched throughout the network collect application-specific system calls and compute the deviation between these system calls and other safe system calls stored in a database. In other words, SI is the deviation between the "self" events stored in the database and the monitored events when the agent enters the node and detects an attack;
- the number of hops: the Hop. It corresponds to the distance in terms of the number of links through wihich the pheromonal information will be propagated from the initial node;
- the pheromonal gradient: the Gd. Like ants with the source of food, IR Agents have to find the better way to locate the source of an alert. For this purpose an electronic gradient field, Gd, is introduced in the pheromone. When the pheromone is diffused across the network, Gd diminishes hop by hop according to a strictly decreasing function. Gd is used in the opposite direction by other travelling IR Agents to travel up to the source of the attack;
- the date $t0$, which corresponds to the date when the attack is detected on the initial node and the pheromone is built;
- the date ti, which corresponds to the date when the pheromone is deposited on each intermediate node i during the pheromone propagation;

- In the rest of the paper, i is the number of the ith node reached by the pheromone during its diffusion, i varying from 0 to n. The initial node, where the pheromone is built, is 0 and the last node is n.

The different steps of the response are:

- the setting-off of an alert by an ID Agent;
- the building of the pheromone;
- the diffusion of the pheromone;
- the pheromone detection by an IR Agent;
- the travel of the IR Agent to the source of the alert;
- the response of the IR Agent.

3.3.2 *Evaporation of the Electronic Pheromone*

In an ants colony the effect of each pheromone is limited in time until the entire pheromone disappears. This phenomena is called the *pheromone evaporation* and it limits the number of ants reaching a particular source of food. The equivalent phenomena for the electronic pheromone is represented by two computed dimensions: (a) the general evaporation index and (b) the extrapolated evaporation date at each node i. These two dimensions are defined in the following sub-sections.

(a) Determining the evaporation index at the last node: Δ.

As long as the electronic pheromone is present along the path back to the source, IR Agents can travel up the pheromonal gradient to the first node. But the pheromone should not stay eternally in the network for two reasons:

- first, it needlessly overloads the network in the case where the response has already occurred, and the pheromone thus has become obsolete;
- second, even if no IR Agent detects the pheromone for a long time, and if the suspicion of an attack persists, it is preferrable to relaunch a pheromone from the same source. This way, it should augment the probability that other IR Agents located elsewhere in the network will meet the pheromonal path. In the worst case where the response is really too slow, we can imagine that the administrator has already solved the problem without waiting for the IR Agents to react. Here again, the pheromone has become useless.

Obviously, the evaporation process of the electronic pheromone has to begin in the last node attained by the pheromone. Then it will reach the other nodes of the path in the opposite way to the diffusion. In this scheme, an IR Agent visiting an intermediate node can always travel the pheromonal gradient up to the first node. We define the evaporation index Δ as the period of life, allocated to the electronic pheromone deposited in the last node n, before it shall disappear. Indeed, this deposit will first disappear at the last node n after a duration Δ. Then, it will successively disappear at each node i after a duration corresponding to the sum of three parameters:

- the duration of the pheromone diffusion between node i and the last node n: *Diff*;
- the evaporation index Δ;
- the duration of the disappearance of the pheromone between the last node n and node i: *Disp*; as we consider that the pheromone disappears along the reverse path at the same speed as it was diffused, Disp is equal to Diff.

Figure 1 represents the different evaporation durations at the last node n and at an intermediate node i.

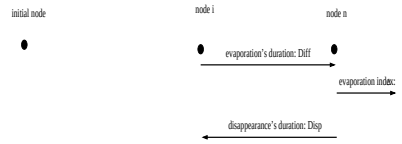

Figure 1: Duration of the Evaporation at the Different Nodes

We decide to evaluate Δ empirically, according to the computational time an IR Agent needs to execute its tasks. The IR Agent entering a node has to:

- access a list where it can read the pheromonal information, notably the pheromonal gradient;
- probe the nodes in its neighborhood to find the same pheromone, but with a higher gradient;
- move to the selected node.

To evaluate Δ empirically, we repeated a series of simulations with a simulation tool called *Starlogo*. Starlogo is a programmable modelling environment for exploring the workings of decentralized systems [2]. We modelled a network with 20 nodes. Each node knows only its neighbors and has at least 4 of them. Node *0* has the maximum number of neighbors, which is 10. We diffused the pheromone at a distance of 14 hops from the initial node and repeated the diffusion process until each node was reached at least three times. Then, we placed an IR Agent on a node as soon as the pheromone was deposited and we saved the IR Agent's computational time as showed in Figure 2. On average, the value of Δ was equal to 2.44 Starlogo time units.

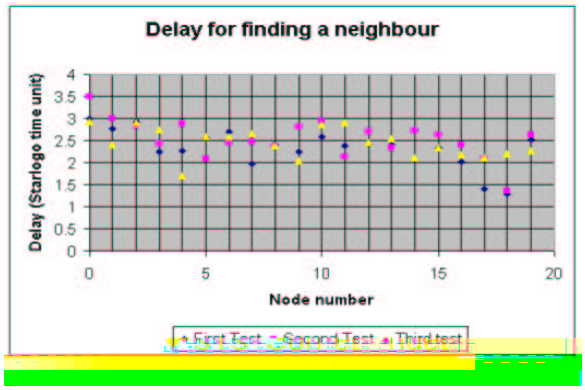

Figure 2: Evaluation of the Evaporation Index Δ

(b) Determining the extrapolated evaporation rate at node i: *Tevap(i)*.

An IR Agent can respond to an intrusion only if it reaches a node i before the pheromone's evaporation date at this node i. We attract your attention to the fact that now, we are speaking in terms of dates and no longer in terms of durations. We want to evaluate the evaporation date at node i in order to compare it with the arrival date of IR Agents at the same node. This will be the subject of more simulations in section 4. When the pheromone is deposited at node i and continues its travel to the last node n, we have to find a way to compute an extrapolated evaporation date at the node i because there are two missing parameters:

- node i does not have the means to know the duration of one hop (average or maximum duration);
- node i does not know the arrival date of the pheromone at the last node;

This extrapolated evaporation date at node i is *Tevap(i)*. This evaporation date should be smaller than the evaporation date at node *i-1*, and higher than the evaporation date at node *i+1*. This is quite logical if we consider that an IR Agent in node *i+1* should have time to travel up to node i (and obviously also to node *i-1*), before the pheromone at node i (respectively at node *i-1*), evaporates.

In order to deal with the missing parameters we proceeded as follows:

- hop duration: as each node i can only save the date *ti* when the pheromone reaches it, it is easy to compute the average duration of a hop from node *0* to node i because *t0* is carried by the pheromone. Then, the average hop duration seen from node i is equal to:

$$\frac{ti - t0}{i}$$

- arrival date at node n: node i has computed the average duration of a hop and can also compute the *n-i* remaining hops until the last node n. Then, viewed from node i, the pheromone will evaporate at the last node n at the date:

$$ti + \frac{n-i}{i} \times (ti - t0)$$

In the same way, the pheromone will evaporate at node n at the date:

$$ti + \frac{n-i}{i} \times (ti - t0) + \Delta$$

The pheromone will evaporate at node i after a duration equal to *Disp* (see sub-section (a)) from the evaporation date at node n. From node i, *Disp* is equal to:

$$\frac{n-i}{i} \times (ti - t0)$$

Finally, the pheromone will evaporate at node i at the date:

$$Tevap(i) = ti + 2 \times \frac{n-i}{i} \times (ti - t0) + \Delta$$

Figure 3 resumes these different steps.

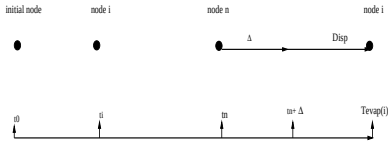

Figure 3: Extrapolated Evaporation Dates at Node i

3.3.3 *Limiting the Number of Responding IR Agents with the Inhibition Index at Node* i

After an IR Agent has found a pheromone and traced the route back to the source of an alert, it is not necessary that other IR Agents trace the same source, even if the pheromone has not totally evaporated. To avoid too many IR Agents converging to the same source, we inhibit the effect of the pheromone. For that, we choose to speed up the pheromone evaporation in each node already visited by an IR Agent, as explained in the following.

A first IR Agent entering node i should intervene between date *ti* and *Tevap(i)* as shown in Figure 3.3.3. We call *tA1(i)* the intervention date of the first IR Agent.

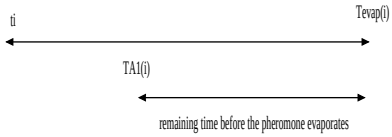

Figure 4: Intervention Date at Node i

So this IR Agent has an intervention duration equal to:

$$Tevap(i) - ti = 2 \times \frac{n-i}{i} \times (ti - t0) + \Delta$$

Thus, after date *tA1(i)*, the time remaining before the pheromone completely evaporates at node i is:

$$2 \times \frac{n-i}{i} \times (ti - t0) + \Delta - tA1(i)$$

Suppose that a second IR Agent reaches node i during this period. As a first IR Agent has already begun tracing the source, we want to avoid too many IR Agents converging to the same source for the same response. To this end, we found a way to inhibit the pheromonal effect by decreasing the remaining time for the next IR Agent by a coefficient that we call the *inhibition index*. Simply put, this mechanism functions as follows:

- the first IR Agent intervenes at date *tA1(i)*;
- the second IR Agent should intervene in a period of time corresponding to 90% (which is the inhibition index) of the remaining time;

Then the remaining time for the second IR Agent is equal to:

$$0.9 \times (2 \times \frac{n-i}{i} \times (ti - t0) + \Delta - tA1(i))$$

Thus, the second IR Agent has to reach node i at a date *tA2(i)* such that:

$$tA1(i) \leq tA2(i) \leq tA1(i)+0.9\times(2 \times \frac{n-i}{i} \times (ti - t0) + \Delta - tA1(i))$$

This inhibition process is repeated for every IR Agent detecting the same pheromone at the same node i, until the pheromone completely disappears.

4. SIMULATION AND RESULTS

As we already said, currently, the response model is being investigated with a simulation tool called Starlogo. Starlogo is a programmable modelling environment for exploring the workings of decentralized systems [2]. A number of preliminary results are reported in the following paragraphs.

4.1 Simulation Topology

The simulation topology of the chosen response scenario takes the following points into consideration:

- the 20 hosts are represented by single nodes in the topology; the nodes are numbered from 0 to 19;
- each host knows only its neighbor nodes;
- different nodes are subject to an attack;
- different ID Agents visiting the nodes can detect a suspicious activity and launch an alert;
- a pheromonal information is built by an ID Agent as soon as the suspicion index is too high. The ID Agent sends away this pheromonal information, choosing randomly one node in its neighborhood. This operation is repeated by each node receiving the pheromone until the last hop;
- different IR Agents visiting the nodes can detect the pheromone and trace the pheromonal gradient up to the alert source.

Table 2 represents the neighbors of each of the 20 nodes.

4.2 Simulation Context

4.2.1 *Pheromone Evaporation Date*

The following simulation parameters are used for the first set of evaluations:

- we choose node *0* to launch a suspicious activity; we iterate this suspicious activity five times;
- we diffuse the pheromone at a distance of 5 hops and we record the nodes reached by the pheromone;
- we collect for each iteration the date of the pheromone evaporation at the visited nodes;
- the average evaporation gradient is set to $\Delta = 2.44$ Starlogo time units, as computed in section 3.

Figure 5 shows the date of pheromone evaporation at each visited node. Each color represents one of the 5 paths borrowed by the pheromone for each iteration.

Node Number	Corresponding Neighbors
0	1-2-3-4-6-7-9-10-12-16
1	0-2-4-5-7-11-14-15-19
2	0-1-3-6-8-10-12-15-19
3	0-2-5-9-11-16-17
4	0-1-5-8-11-16-18
5	1-3-4-6-13-14-19
6	0-2-5-8-9-10-15
7	0-1-8-9-12-14
8	2-4-6-7-10-13-17
9	0-3-6-7-11-13-15
10	0-2-6-8-16-17-18-19
11	1-3-4-9-12-18
12	0-2-7-11-14-16
13	5-8-9-14-15-19
14	1-5-7-12-13-15-16
15	1-2-6-9-13-14-19
16	0-3-4-10-12-14
17	3-8-10-18
18	4-10-11-17
19	1-2-5-10-13-15

Table 2: Neighbor of each Node

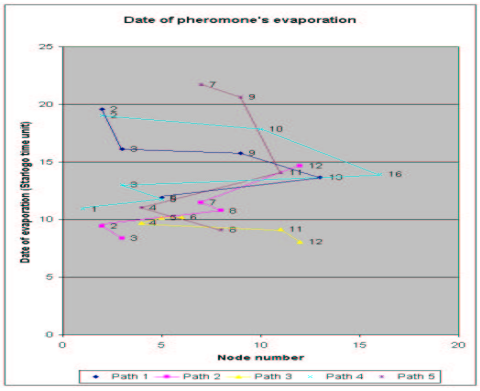

Figure 5: Pheromone Evaporation Dates for a 5 Hops Simulation

4.2.2 *Agents Response Dates*

For the fastest pheromone evaporation of the previous simulation (path number 3), we dispatched IR Agents in the network and we collected the date of arrival of IR Agents on all nodes of the network. The following simulation parameters are used for this second set of evaluations:

- we choose node *0* to launch a suspicious activity;
- for path 3, the evaluated evaporation duration is 10.18 Starlogo time units;
- at the end of the pheromone diffusion, we randomly launch IR Agents for the duration of the pheromone evaporation;
- there is a random delay between two successive IR Agent launches;
- on each visited node, we collect the date of each IR Agent arrival;

- we repeat the simulation 10 times;

Figure 6 shows the date of the IR Agents' arrivals during the fastest evaporation duration for one of the 10 simulations.

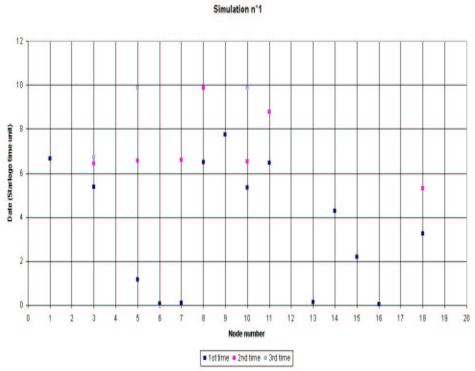

Figure 6: Agents Response Dates for the First Simulation

4.3 Simulation Results

The purpose of this section is to correlate the simulations represented in Figure 6 with the particular case of the fastest evaporation duration (path 3 in Figure 5). To this end, we compare the arrival dates of IR Agents on the different nodes with the evaporation dates of the pheromone along path 3. By locating the common nodes, we compute the number of IR Agents which responded in time. Table 3 summarizes the results obtained for the entire simulations of section 4.2.2. Each row of the table shows the frequency of the effective responses according to the number of IR Agents dispatched through the network.

We notice that in all simulations there are IR Agents which answered before the complete evaporation of the pheromone. Considering that we did not limit the number of acting IR Agents during the duration of the evaporation, an average of 27.5% of dispatched IR Agents responded in time. Besides, there are in average 5 effective responses. These first results show the efficiency of the concept of electronic pheromone to trace the source of an alert and to obtain a significant number of responses. This also demonstrates that the values computed in section 3 are viable, even if some of them should be adjusted in the future. We are already investigating how to tune our model by:

- adjusting the inhibition index in order to limit the number of IR Agent responses;
- adjusting Δ;
- limiting the number of randomly dispatched IR Agents in order to find their optimal number. That is, the minimal number of IR Agents needed to obtain just one answer.

5. FUTURE WORK AND CONCLUSION

We presented in this paper an approach inspired by natural systems for intrusion detection and response. Our approach uses the immune system metaphor for intrusion detection (ID Agents) and social insects stigmergic behaviour

Simulation	Answers	Agents	Frequency
1	6	24	0.250
2	6	20	0.300
3	3	13	0.231
4	4	17	0.235
5	7	23	0.304
6	9	17	0.529
7	4	14	0.286
8	4	26	0.154
9	5	14	0.357
10	2	19	0.105
Average frequency			0.275

Table 3: Response Frequency According to the Number of IR Agents

metaphor for intrusion response (IR Agents). The paper is focused on the study of the intrusion response scheme, furthering the approach proposed in [6], by tuning the parameters of electronic pheromone diffusion and evaporation, and introducing a new mechanism for limiting the number of responding IR Agents, using an inhibition index. We presented the first results of a simulation, which show that the approach is promising. The next steps consist in refining the approach in order to determine if some parameters can be omitted or merged and still result in a useful behavior, or if, on the contrary, additional parameters are needed. This work is beneficial for the general understanding of the model and also for the optimization of the implementation. This model is being implemented using the J-SEAL2 mobile agent framework [14]. As can be deduced from the previous sections, there are three kinds of agents involved: ID Agents, IR Agents and Pheromone agents. This specialization of roles is designed to make agents as light-weight as possible, in order to achieve good performance. An IR Agent is, on the other hand, intentionally very generic; the goal is to be able to locate the source of an alert using a single "universal" mechanism, and once this source is reached, to enable a threat-specific response by downloading and activating a dedicated class file designated by a URL contained in the pheromone.

6. ACKNOWLEDGMENTS

Thanks to Giovanna Di Marzo for many helpful discussions and to David Landecy for his contribution to the simulations.

7. ADDITIONAL AUTHORS

Additional author: Jarle Hulaas (University of Geneva email: `Jarle.Hulaas@cui.unige.ch`).

8. REFERENCES

[1] CERT Coordination Centre. Cert coordination centre. Technical report, January 2000.

[2] Starlogo. http://el.www.media.mit.edu/groups/el/projects/starlogo/.

[3] D. Ragsdale, C.A. Carver, J. Humphries, and U. Pooch. Adaptation techniques for intrusion detection and intrusion response system. *Proceedings of the IEEE International Conference on Systems,*

Man, and Cybernetics at Nashville, Tennessee, pages 2344–2349, October 8-11 2000.

[4] C. A. Carver, Jr., and U. Pooch. An intrusion response taxonomy and its role in automatic intrusion response. *Proceedings of the IEEE Systems, Cybernetics Information Assurance and Security at Workshop West Point, New York*, June 6-7 2000.

[5] W. Jansen, P. Mell, T. Karygiannis, and D. Marks. Applying mobile agents to intrusion detection and response. Technical report, National Intitut of Sandard and Technology, Interim Report 6416, September 1999.

[6] S. Fenet and S. Hassas. A distributed intrusion detection and response system based on mobile autonomous agents using social insects communication paradigm. Proceedings of the 1st International Workshop on Security of Mobile Multiagent Systems (SEMAS), 2001.

[7] N. Foukia, D. Billard, and Pr. Juergen Harms. Computer system immunity using mobile agents. HPOVUA Workshop, Berlin, June 2001.

[8] D. Schnackenberg, K. Djahandari, and D. Sterne. Infrastructure for intrusion detection and response. Boeing Phantom Works, NAI Labs, Network Associates.

[9] S. Forrest, S. Hofmeyr, and A. Somayaji. Computer immunology. *Communications of the ACM*, 1997.

[10] P.P Grassé. La reconstruction du nid et les interactions inter-individuelles chez les bellicoitermes natalenis et cubitermes, la thorie de la stigmergie - essai d'interprtation des termites constructeurs. *Insectes Sociaux, no. 6*, pages 41–81, 1959.

[11] C. Ronald Kube and E. Bonabeau. Cooperative transport by ants and robots. *Robotics and Autonomous Systems*, 1998.

[12] M. Dorigo and G. Di Caro. Ants colonies for adaptive routing in packet-switched communication networks. *Lecture Notes in Computer Science*, page 673, 1998.

[13] E.B. Mallon and N.R. Franks. Ants estimate area using buffon's needle. *Proceedings of the Royal Society, London*, April 2000.

[14] J-Seal2. http://www.coco.co.at/development/.

Developing Secure Agent Systems Using Delegation Based Trust Management *

Lalana Kagal
University of Maryland
Baltimore County
1000 Hilltop Circle
Baltimore, MD 21250
lkagal1@cs.umbc.edu

Tim Finin
University of Maryland
Baltimore County
1000 Hilltop Circle
Baltimore, MD 21250
finin@cs.umbc.edu

Anupam Joshi
University of Maryland
Baltimore County
1000 Hilltop Circle
Baltimore, MD 21250
joshi@cs.umbc.edu

ABSTRACT

We present an approach to some security problems in multi-agent systems based on distributed trust and the delegation of permissions, and credibility. We assume an open environment in which agents must interact with other agents with which they are not familiar. In particular, an agent will receive requests and assertions from other agents and must decide how to act on the requests and assess the credibility of the assertions. In a closed environment, agents have well known and familiar transaction partners whose rights and credibility are known. The problem thus reduces to authentication – the reliable identification of agents' true identity. In an open environment, however, agents must transact business even when knowing the true identities is un-informative. Decisions about who to believe and who to serve must be based on an agent's properties. These properties are established by proving them from an agent's credentials, delegation assertions, and the appropriate security policy. We begin by describing our approach and the concepts on which it is built. Then we present a design that provides security functions (authorization and credibility assessment) in a typical agent framework (FIPA) and describe initial work in its realization using the semantic web language DAML+OIL.

1. INTRODUCTION

Though there has been some research in trust based security for multi-agent systems, generally multi-agent systems have always relied on traditional security schemes like access control lists, role based access control and public key infrastructure. These physical methods use system-based controls to verify the identity of an agent or process, explicitly enabling or restricting the ability to use, change, or view a computer resource. However these methods generally require some sort of central repository or control to provide authentication and need to store access control information for individual agents or groups of agents. We believe that these schemes will not scale adequately or provide the increased flexibility required for emerging dynamic multi-agent systems that consists of an extremely large number of agents that are spread over a large geographic area [11] like the agentcities project[1]. Hence we argue that it no longer makes sense to divide authorization into authentication and access control [16, 14].

We propose a security framework for multi-agent systems which is based on distributed trust management. Distributed trust management involves proving that an agent has the ability to access some service/resource solely by verifying that its credentials comply with the security policy of the requested service [16, 2]. These credentials include properties of the agent, for example, membership in certain organizations, age or host of the agent, recommendations and delegations by other agents. The process of verifying the credentials is itself under the security policy of the verifying agent. Aspects of trust management include creating security policies, associating credentials with certain abilities and reasoning over these policies and credentials to decide the rights of an agent. Our trust management system includes a trust ontology for specifying entities or principals, policies, credentials, a mechanism for verifying credentials and a mechanism for checking if the credentials conform to the policy. The policy includes a set of rules that associate a required set of credentials with a certain ability or right; implying that only agents with the specified credentials can possess the ability.

Agents communicate their beliefs with each other for trust management. Beliefs are exchanged in terms of delegations, credentials, abilities of other agents and trust values. An agent will reason about beliefs (its own and of other agents) and policies while making authorization decisions.

This framework, based on FIPA specifications [6], addresses many of the security threats generally associated with Multi-Agent Systems (MAS) [20]. The challenges usually associated with MAS are corrupted naming (Agent Management System) and matchmaking (Directory Facilitator) services, insecure communication, insecure delegations, lack of accountability, access control for foreign agents, and lack of central control [20].

2. RELATED WORK

There has been a lot of interesting work in security for multi-agent systems, and in this section we describe some research projects that are most relevant to ours.

Wong and Sycara describe the design of a security infrastructure for multi-agent systems [20]. Their work is based on RETSINA, a resuable multi-agent infrastructure. The authors describe several threats associated with multi-agent systems with respect to the RETSINA framework; corrupted agent naming servers or matchmakers, insecure communication channels, insecure delegations, and lack of accountability. To prevent the threat of corrupted ANSs or matchmakers, the authors believe it is necessary to use only trusted ANSs and matchmakers that behave as they should, by only servicing valid requests, inserting/removing entries from their database in a way that is consistent with the request and giving responses that are consistent with their databases. As a way of counteracting lack of accountability, all agents should be given proofs of identity that

*This work was supported by NSF Awards IIS 9875433 and CCR 0070802, and the Defense Advanced Research Projects Agency under contract F30602-00-2-0 591 AO K528.

[1] http://www.agentcities.org/

cannot be forged and deployers of agents should be made responsible for the actions of their agents. Communication channels should be made secure and agents should be made to prove that they are delegatees of whom they claim to be. Certificates are used to link agents to actions and deployers to agents for accountability. The authors describe mechanisms for agent key certification and revocation, in which the deployer interacts with the ACA. Then they discuss protocols for registration, unregistration and lookup. To handle insecure communication channels, the authors plan to add SSL (secure socket layer) underneath their agent communication layer.

In their paper 'Distributed Trust in Open Multi-Agent Systems', the authors build on earlier work by Herzberg et al [8] to define an infrastructure for distributed trust in multi-agent systems [15]. Following Herzberg's assumptions, the authors think that identity is not required for trust management, and that there is no need for a centralized certificate mechanism or trusted third parties. This work is based on the use of certificates. Most role based access control mechanisms map users' identities to role. However this is not the approach used in this work; an agent uses its policy to map another agent to a role, based on the latter's certificates [8]. Any agent can be a certificate issuer, and may not be globally trusted. An issuer is trusted when it can provide sufficient certificates from other issuers to satisfy the requester's policy. An agent could have several certificates certifying its capabilities and its performance. These certificates will be from other agents that have used the agent's services. However these certifying agents may not be globally trusted. If an agent X needs to find a particular service, it sends a request to the MatchMaker in a system like RETSINA [20]. The MatchMaker will return a list of matching agents and their certificates. The requesting agent will reason about these certificates to decide which agents can be trusted. The policy will define rules for deciding trust levels based on the certificates. To solve the problem of authorizing accessing agents, every agent has as part of its architecture an access control mechanism. This component helps the agent decide which services should be accessible to a certain agent. The access control component uses certificates to map an accessing agent to a role, and then uses role based access control to decide its access rights.

In his paper, Hu explains how to build up an agent oriented PKI and demonstrates some delegation mechanisms for it [9]. In this agent oriented PKI, there are two types of certificates; identity certificates for humans and their agent, and authorization certificates for humans and agents. Authorization certificates are used to represent authorizations by entities. These include the public key for the granting entity, the public key of the entity receiving the authorization, the actual authorization (access right), re-delegation bit, and the validation period. However, the re-delegation bit always set to 1, because the author does not have any fail-proof method of preventing re-delegation. Though there is a difference between trust between humans and agents and between agents, the author models them in the same way. Hu also describes 3 types of delegations; chain-ruled, threshold, and conditional. In chain-ruled the access rights are delegated in a cascading manner. Threshold delegation allows an entity to delegate to multiple subjects. These subjects must co-operate with each other to perform the delegation. When the subject has to satisfy certain conditions in oder to use the delegation, it is called conditional delegation. As authorizations can be re-delegated, they form delegation networks. The verification process checks that every entity in the delegation network has the authority to re-delegate, that all the authorizations are within the validity period, and that none of the required certificates have been revoked. However, this study does not include mechanisms for handling revocation of certificates. The verification can either be done by a Trusted Third Party or the original issuer agent. Usually the service guardian authorizes other agents to use the service, who in turn authorize other agents. Generally the original issuer agent is the verifying authority as well. Rules for verifying an authority are specified as part of the delegation policies within the original issuer agent's rule base. If a Trusted Third Party is responsible for verification of authority validity, then it is also responsible for all the service access control. The author has included several performatives for human/agent identity certificate management and human/agent authorization certificate management. Hu also describes how these performatives are encoded in XML for agent communication.

Poslad et al. describe the security and trust notions currently part of the FIPA specifications and point out some of its strengths and weaknesses [17]. The FIPA security specifications were started in 1998, but are still not complete and have actually been made obsolete by FIPA. The authors believe that security is domain dependent and that it is not possible to have a general security architecture which is suitable for all applications. The authors describe the trust models existing in FIPA. All agents that want to use services or provide services in a platform must register with the platform's Agent Management System (AMS). The AMS is trusted and maintains the identity of all registered agents. However as authentication is not mandated, spoofing is possible. AMS is responsible for the life cycle for all agents in the platform and agents must report all significant changes to the AMS and allow the AMS to control their life cycles. However an agent need not obey orders from the AMS, causing the AMS to take some other course of action like using an external API or de-registering the agent. Agents also register their capabilities/services with the Directory Facilitator (DF). There are no specifications about this registration, so a malicious agent could cause a lot of damage by registering non-existent services, registering wrong service descriptions etc. FIPA does not define how accessing agents can specify their preferences. There exists a trust relationship between the Agent Communication Channel (ACC) and registered agents. The ACC is trusted to transmit the messages in a timely fashion and to maintain the integrity of the messages. The FIPA security model [7] defines mechanisms for keeping messages private, mechanisms to check the integrity of messages and authentication messages. This model extends the functionality of AMS and DF and introduces an entity called Agent Platform Security Manager (APSM), which is responsible for maintaining security in the platform. The AMS uses public key infrastructure mechanisms for authenticating agents wishing to register with it. This raises issues related to PKI [3]. The agents define additional security parameters as part of their service descriptions which they register with the DF. The current specifications also include some suggestions for secure Agent Communication Language (ACL) communications, mainly the envelop construct. Certain keywords like authentication, non-repudiation etc can be used to express a level of security. When an agent requests a service, it is the responsibility of the message transport layer to encapsulate the messages based on these levels. The semantics of these keywords are provided by the platform. The authors propose certain requirements for adding security to FIPA systems, including authentication of agents by middle agents (AMS and DF) when writing to directories accessed via middle agents, use of private channel to send messages, and authentication of middle agents by agents for bi-directional trust.

3. DESIGN

This model provides security based on distributed trust management for open, dynamic agent platforms, with methods for intra-platform and inter-platform security.

Agents are authorized to access a certain service if they have the required credentials. Our work is similar to role based access control in that a user's access rights are computed from its properties. However, we use additional ontologies that include not just role hierarchies but any properties and constraints expressed in a semantic language including elements of both description logics and declarative rules. For example, there could a rule specifying that if an agent in a meeting room is using the projector, it is probably a presenter and should be allowed to use the computer too. In this way, rights can be assigned dynamically without creating a new role. Similarly, rights can be revoked from a user without changing his/her role, making this approach more flexible and maintainable than role based access control.

We extend the functionality of the Agent Management System (AMS) and the Directory Facilitator (DF) to manage security for the platform, as not all agents should be able to register on a particular platform or use a certain DF. Similarly, agents are also given some access control ability. The AMS, DF and agents follow certain security policies to decide the access rights of requesting agents.

Our system addresses the challenges associated with MAS, namely, corrupted AMS and DF, insecure communication, insecure delegations, lack of accountability, access control for foreign agents, and lack of central control. The model manages corrupted naming and matchmaking services by using a PKI handshaking protocol between the agent and the AMS to verify validity of both parties. All messages are encrypted according to Public Key Infrastructure. However we do not use these certificates for authenticating agents but for exchanging messages securely. Our delegation mechanism is able to thwart any invalid or insecure delegations. Only agents with the right to delegate can actually make valid delegations that change the access rights of other agents. All agents are held accountable for their actions because they have to sign all service queries and requests with their own private key. As there is a unique private key public key pair, once an agent signs a request, the agent can be held accountable. Our infrastructure allows foreign or unknown agents access into the system using trust management. When an unknown agent tries to register with a platform, the platform checks the agents credentials, and decides its rights with that platform based on the security policy. Multiagent systems are inherently decentralized and it is not possible to have a central database of access rights or policies. This is not a problem in our system as no central information is required. The policy is enforced individually at the entity processing the request. The policy is enforced at two levels; at the platform level, where access to the AMS and DF is controlled and at the agent level, where an agent can specify who can access its services.

Agents are able to delegate their rights in a controlled and secure fashion. For example, if agent A delegates some service to agent B, and agent B tries to delegate this service to agent C, then the second delegation will fail as agent A did not give agent B the ability to redelegate.

The agents use a semantic language like DAML+OIL [4] as an ontology language. DAML+OIL is an ontology language for marking up resources, and is basically being developed for the realization of the Semantic Web[2]. The agents express security information including credentials, delegations, and policies in DAML+OIL making it easier for other agents to interpret them correctly.

3.1 Security Classification

We classify security into two levels depending on where it is enforced: platform or agent. In platform security, the AMS and DF have additional security features. The AMS can decide whether or not to allow an agent to register, search or use its other functions. Similarly, the DF can also decide whether to allow an agent to register, modify or search for agents based on certain access control information. An agent, while registering with a platform, can send some security information to the AMS specifying its security category; *private*, *secure* or *open*. A *private* agent's Agent Identifier (AID) is not displayed to any other agent by the AMS, a *secure* agent has to send some access control information so that the AMS can filter requests to the agent and and an *open* agent is visible to all agents. Similarly, while registering its services with a DF, the agent can choose a category for each service. For example, an agent A can register as an open agent with the AMS and register two services with the DF, a GPS service which is open and a navigator service which is secure. Agent A also specifies that only agent B, with certain credentials, can access the navigator service. In agent security, the agent uses a policy to decide how to further validate service requests.

3.2 Platform Security

In platform security the AMS and DF use distributed trust management principles to authorize requests to their services.

3.2.1 Security Module for an AMS

When an agent wants to register with the AMS, it signs it's request and sends it to the AMS, along with its digital identity certificate. The AMS verifies the certificate based on the rules. The rules could be of the form, an entity X of the organization Y with a certificate from trusted Certificate Authority CA(Y) is valid. Or there could be a rule saying, for all certificates from organization Z, calculate certification path and verify with the CA. If the certificate is valid, the AMS checks the signature. The AMS uses its policies to decide what access rights the agent has on the platform.

If the agent does not have the right to register with the AMS, its request is denied. If the agent does have the right to register with the AMS, the AMS starts the *handshaking protocol* that is common in Public Key Systems. It sends the agent a small message, a nonce, encrypted with the agents public key, and attaches the platforms certificate, to the address specified by the requesting agent. This is not only done so that both parties can verify each other, but also in order to verify the agents location, prevent spoofing and securely exchange information. The agent can now go ahead and verify the platforms certificate. It then replies to the AMS with the same nonce encrypted with the platforms public key. On receiving this, the AMS creates a *trust certificate* containing the platform related rights of the agent, the associated public key, time validity and other relevant information and sends it back. This certificate is valid only for a short time, after which the agent has to start the registration process again. This period is directly based on the *level of trust* associated with the agent or in fact the agent's *reputation* in the platform. Using a trust certificate enables the AMS and DF to skip the rechecking of the agents' credentials everytime the agent tries to use the services of the platform.

After creating the trust certificate, the AMS will inform the agent about all the agents that are either in the open category or the secure category for which the agent fulfills the required conditions for access. During the period of validity of the trust certificate, the agent can make requests to access the AMSs services. These requests have to be signed. The AMS does not need to check all the credentials of the agent, but only verifies that the agent has the right to the requested service.

3.2.2 Security Module for a DF

[2]W3C's Ontology Wrapper Language (OWL) is based on DAML+OIL

After obtaining a trust certificate from the AMS of a platform, the agent can access various services of the AMS and the DFs. Using the trust certificate, an agent can register its services with the DF, if the platforms policy allows that particular agent to use the DF. This service registration message is signed with the agents private key, and acts as a digital signature. This forces agents to be accountable for their actions. The DF verifies the trust certificate and checks that the trust certificate is valid and belongs to the agent. It retrieves the agents public key from the trust certificate, and checks the signature of the registration message. If the certificate states that the agent has the right to register, the DF proceeds with the registration. An agent can register its different services under different categories. This service description is also in DAML+OIL, making the searching more semantic and more flexible. To query the DF, the agent sends a signed query message to the DF. The DF verifies the message and the checks the category of the service that fulfills the search query, the conditions attached if a secure service, and the access rights of the requester before sending back any results. These results are encrypted with the agents public key, which is associated with the trust certificate.

Agents can 'delegate' authorization ability to the DF if they share domain ontology. If an agent cannot use the DF for making authorization decisions on its behalf, then the agent has to contain a trust management engine and interpret its own policies. This makes the presence of the engine in an agent optional, allowing agents to run on smaller, lightweight, devices. The AMS/DF has a list of conditions that an agent must satisfy in order to contact a particular agent or use a particular service. However the AMS and DF need to understand the service agent's[3] policy or have access to its knowledge base. It is upto the service agent to make sure that these conditions are accurate and conform to its policy. In some cases, the AMS or DF cannot understand the associated conditions. Then, based on the policy of the platform, the AMS and DF can decide to reject all requests for the agent or service or accept all requests and forward them to the appropriate service agent for interpretation.

3.3 Agent Security

The authorization decisions carried out by individual agents for access to their services comprises agent security.

3.3.1 Security Module for Agent

Every service agent has two modes of operation as an owner of a service and as a requester of a service.

Owner of a service

Security on the agents side can be handled in multiple ways. An agent can decide to register its services as *open* or *private* on the DF, so that the agent itself is completely responsible for access control. The second way, is for the agent to categorize its services as *secure* and specify the access control conditions in the DF. If the agent trusts the DF completely, it can rely on the DF to handle access control and the agent need not have a security module at all. If the agent does not trust the DF, it can implement its own security module for stricter access control. In this case, after the requests are filtered by the DF, they can be re-verified by the service agent.

Requester of a service

After an agent receives a matching list of services as a result of its DF query, it tries to execute one of them. The agent sends a request to the service agent and attaches its identity certificate and trust certificate. This message is encrypted using the agents private key. The receiving agent carries out similar reasoning as the AMS, by going through its certificate verification rules to verify the identity certificate and trust certificate. If both the certificates are valid, it verifies the signature. It then uses its security policy to decide if the agent meets its requirements for accessing that particular service. If all the checks are valid, then the receiving agent sends the result back encrypted with the senders public key. The agent does not go through the handshaking procedure because the sender has a valid trust certificate from the platform. Even after the platform checks by the AMS and DF, a service agent may decide not to honor a certain request, because there may be certain additional constraints it requires that the requesting agent fails to meet.

[3]The agent controlling a service is known as service agent

4. INTER PLATFORM SECURITY

If an agent is already registered with a platform and wants to access the AMS or DF on another platform, the agent should send along with its identity certificate, its current trust certificate, which contains information about its access rights. The remote platform decides the agents rights in the normal fashion based on its own security policy, and may take into consideration the platform that the agent is currently registered on.

DF's of different platforms can be accessed if they register with each other through principles of *federations of DF* [6]. If an agent is searching for a particular service, and its DF cannot find any matching service, the DF will forward the request with the trust certificate to the other DFs registered with it. These DFs will process the agents request as normal and return the results.

5. VERIFICATION OF CREDENTIALS

Credentials are properties of agents that are described in a semantic language and signed by other agents. Delegations are special credentials and are discussed in detail in Section 6. In order to accept credentials of other agents, an agent must be able to verify these credentials. Verification can be carried out in the following ways

- Simple Verification : In this scheme, a service agent expects all the credentials necessarily at the time of request. In order to use its services, a requesting agent must send all required credentials along with the request for service. The service agent will check its knowledge base, and question other agents about their beliefs in order to verify the credentials. Suppose agent A has an alarm service which requires that requesters be AAAI members. The security policy of agent A also states that the agent XYZ should be trusted to verify AAAI certificates. An agent B sends A a request to use the service along with its certificate from the AAAI CA. This certificate states that the bearer of this certificate is a member of AAAI. Agent A asks agent XYZ to verify the certificate. If the certificate is valid then agent B is authorized to use the alarm service. If agent B did not send the required certificate or sent an invalid certificate, its request would be denied.

- Negotiation : Certain service agents may provide a more interactive requesting mechanism. If the requested agent does not provide the correct credentials to access the service, the service agent asks the requester for specific additional credentials. For example, a service agent A only allows employees of XYZ Pvt. Ltd. to access its services, and accepts delegations from these employees. Agent B approaches agent A with a credential from AAAI. Agent A decides that the credential is not good enough and asks the agent B to prove that it is an employee of XYZ or if B has a delegation from an employee. Agent B possesses a delegation from Bob who is an employee of XYZ and sends this delegation to A. A ver-

ifies the delegation and the chain of delegations and decides to authorize agent B's request.

- Third Party : Some service agents do not have the resources to verify credentials and so request trusted third parties to handle the verification on their behalf. Suppose a trusted agent, C, did have the resources and the inclination to help agent A, agent A would send B's credentials to C to be verified and would trust C's response. C could either use simple verification or negotiation to verify these credentials.

6. DELEGATIONS

An agent has the ability to make any delegation, but whether it is honored depends on various factors, including the security policy, the agent's rights, and the rights of the agents ahead it in the delegation chain. Agents are not prevented from making delegations, but the delegations by unauthorized agents are considered invalid. Only agents with the ability to delegate can make valid delegations. Valid delegations change access rights of other agents. The right to delegate is defined implicitly and explicitly. Implicitly, an agent can delegate rights to any service it offers. Explicitly, an agent that has been given the right to delegate by an authorized agent can perform valid delegations, as long as the delegation fulfills the constraints of the previous delegation. This forms a chain of constraints; the agent at the end of the chain must satisfy all the constraints associated with the delegations in the chain. Our delegation mechanism, written in logic, verifies that the requesting agent satisfies all the constraints of the delegations before it in the chain.

Our framework allows certain authorized agents to delegate access rights, with restrictions attached, to other agents. A delegation usually has constraints attached, such as one that limits the access to a certain period, or to whom the right can be re-delegated. A delegation consists of various information; delegator, right, constraints on delegatee, constraints on execution, constraints on re-delegation and time period. By using constraints on delegatee, the delegator can specify whom to delegate to. For example, a delegation could be conferred on all agents with certificates from a certain CA and registered with a certain platform. By restricting which of the delegatee can actually use the right, the delegator can prevent wrongful execution of the right. Constraints on re-delegation allow the delegator to decide whether the right can be re-delegated and to whom it can be re-delegated. We have developed rules that capture this information and enforce security by checking these constraints at the right time. We have separated the constraints on execution from the constraints on delegatee, to make delegation more flexible and its management more complete.

6.1 Delegation Management

Though delegation is very important for the propagation of trust, managing delegations in a distributed and dynamic environment is rather difficult. Consider the following example, an agent (delegator), who is delegated a certain right, delegates it to another agent (delegatee) and goes down immediately. The delegatee asks to use the certain resource and presents its delegation certificate. However this request cannot be validated because the delegator's ability to delegate cannot be checked.

We suggest three schemes for managing delegations

Delegation Chain The previous example can be solved by making the delegator attach its own delegation certificate to the newly created delegation before sending it to the delegatee. This means that every agent will have to store a chain of delegation certificates leading to its own delegation, in order to validate its delegation. This is not feasible because each chain could be very long and there could be several delegations for every agent. To reduce the number of certificates in a chain, certificate reduction could be used [1], but the original delegator may not be accessible.

Centralized Delegation To avoid handling and processing chains of delegations, all delegations can be addressed to the service agent or the agent platform responsible for the service agent. However this scheme has two problems; it is rather centralized and the delegator may not be able to access either the service agent or the agent platform at the time of the delegation.

Delegations on the Web The last scheme is to continue using delegation chains, but instead of storing the chains within the agent, the chains could be stored on web pages. In order to prove it has a certain ability, an agent could point to a certain delegation on its delegation page. This delegation in turn would refer to a delegation on the page of the agent who made the delegator. By traversing this delegations, the agent platform and/or service agent would be able to verify the delegation and decide whether or not to authorize the request.

7. TRUST PERFORMATIVES AND INTERACTION PROTOCOLS

FIPA is based on speech acts, predicate logic and public ontologies. Speech acts are ways of communicating or expressing oneself [18]. A speech act only succeeds if it is understood by the recipient as intended. However FIPA does not include the speech acts required for trust management. As part of this security initiative, several speech acts, that are common to distributed trust domains, will be modeled. In this framework, agents will use certain speech acts to explain their intent; delegating, requesting etc. For example, "I delegate to you the ability to access my files for one hour", or "I request you to delegate to me the use of your workstation". These statements contain a lot of information that needs to be captured. An ontology, grounded in DAML+OIL will be used to describe these speech acts. This ontology will enable the audience to correctly interpret the speech act and understand its purpose. FIPA Communicative Acts describe a set of "utterances" used in multi-agent systems, and FIPA Interaction Protocol specifies the order of messages exchanged. Though most of the communication between the agents can be modeled with existing FIPA performative, we believe certain additional performatives are required for agent security and trust.

The performatives that will be added are Request Permission, Delegate, Request Verification, and Credential Required.

- Request Permission
  The action of asking another agent for permission to access a certain service.
- Delegate
  The action of delegating to another agent or group of agents the ability to perform a certain action on a certain service.
- Credential Required
  The action of asking the recipient to provide additional credentials. The content is the credential required and this performative is the response to a request where the recipient did not provide the correct credentials.
- Request Verification
  The action of asking the recipient to verify credentials supplied by an agent requesting access to the sender's service.

Using existing FIPA communicative acts and the performatives described above, we describe the interaction protocols for trust management in our system.

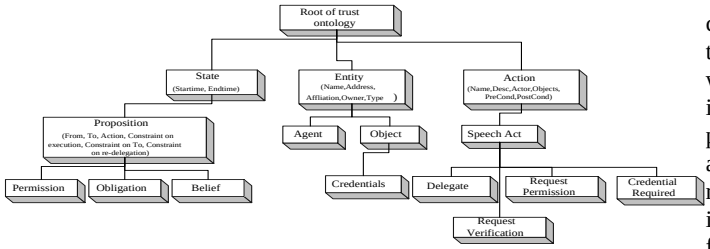

Figure 1: Our trust ontology as a class hierarchy

- Request Interaction Protocol
  This interaction protocol allows the initiator to request the use of another agent's service. The initiator sends the recipient a request message. Similar to FIPA, some responses are not-understood, refuse, agree, failure, inform-done, and inform-ref [5]. However depending on the kind of verification being performed by the recipient, the responses could also include Credential Required. The sender would now have to resend its request with the new credentials in order to gain access to the service.

- Request Permission Interaction Protocol
  An agent uses this protocol to request another agent to delegate certain abilities to it. The initiator starts the protocol by sending the recipient a Request Permission message. The responses from the recipient include not-understood, refuse, agree, failure, and Delegate.

- Request Verification Interaction Protocol
  The initiator uses this protocol when it is unable to verify some credentials and requires the recipient to verify the credentials on its behalf. The initiator starts this interaction protocol by sending a Request Verification message. The valid responses to this message are not-understood, refuse, agree, failure, inform-done, and inform-ref.

8. ONTOLOGIES

Our infrastructure uses ontologies expressed in DAML+OIL to represent security information and policies in a multi-agent system.

We have designed an ontology for trust and security information in this system, which is illustrated in Figure One. The root of the ontology is divided into State, Entity and Action. State contains all information pertaining to the current state. It currently has one subclass, Proposition, which is further sub-classified into Permission, Obligation, and Belief. Propositions are clauses that have a truth value in the system. An Entity could either be an Agent or an Object. An object can be extended to define domain specific resources like credentials, files, computers, printers, etc. An Action is associated with a set of Objects or resources. Speech acts like Requests and Delegations are extensions of Actions.

The ontology specific to agent systems extends the main trust ontology with information related to FIPA platforms; register an agent, deregister an agent, search, create, agent service, etc. as actions and certificates, platform address, network address, network protocol used etc. as objects. Figure Two shows part of the Agent ontology.

9. POLICY

The security policies are based on the Agent ontology. Each platform and agent follows a security policy. A security policy may contain rules for verifying certificates and credentials, access control, and delegation. Rules for verifying certificates could specify which certificate authorities are trusted, and the procedure involved in verifying different kinds of certificates, based on the CA, principal, agent etc. Rules for access control will state the credentials an agent must have for a certain access right. The policy also contains rules that describe the way delegations and revocations propagate in the system, how re-delegations are handled, how prohibitions affect access control and delegations and how revocations should be managed. For example, if a delegation is revoked, should all the agents that the delegatee delegated to, lose the access right as well or can they keep it and whether a prohibition is given priority over a delegation while deciding access rights.

Our *default policy* defines certain rules about the propagation of delegations so that all constraints in the delegation chain are applied before an agent can gain access to a service. If a certain link in the delegation chain fails or the right is revoked, the rest of the chain after this failed link loses the access right as well. This default policy also includes rules for the mechanisms of credential verification and belief management.

10. PREVIOUS WORK

We have previously developed two security systems based on distributed trust management - an agent-based supply chain management application [12] and an agent-mediated pervasive computing environment [13, 19]. During their implementation we have refined our trust management concepts and developed several programs in logic for handling the propagation of delegation, and validating requests.

10.1 Security for Supply Chain Management Systems

We successfully implemented a trust based framework for the Extended Enterprise COalition for Integrated Collaborative Manufacturing Systems (EECOMS) project, which is aimed at providing a set of technologies for integrated supply chain and business to business electronic commerce [10]. A supply chain management system consists of groups of buyers and sellers that need to open up their internal systems to each other in a secure way. In other words, a supply chain management system consists of a network of heterogeneous agents that interact to perform certain actions that may or may not need authorization. The main problem is guaranteeing the authenticity of requests between these agents, whether within a company or between one or more companies.

Our system sets up authorization and delegation rules, so that the information in the SCM may be accessed only by authorized agents. Special intelligent agents called *security agents* are required for authentication and authorization within a particular domain, and are trusted within the company and by the companys buyers and sellers. They also represent the company in some sense. The security agents of a company enforce the company policy. This policy describes certain rules for rights, delegation and for reasoning about them. The policy is not changed frequently and usually involves human intervention. Agents within a company possess an identity certificate that is signed by a trusted Certificate Authority. Agents within a company can be authenticated by the security agents through their ID certificates.

In order to allow the buyer's employees to access certain information within its company, the security agent of the seller gives the security agent of the buyer the permission to access that information, and the ability to delegate this right. To propagate this trust within its own company, the seller's security agent delegates this right to some of its employees based on the policy. Depending on

```
<?xml version="1.0" encoding="UTF-8" ?>

<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:daml="http://www.daml.org/2001/03/daml+oil#"
    xmlns:dtrust = "http://daml.umbc.edu/ontologies/trust-ont#"
>
<daml:Ontology>
      <daml:Class rdf:ID="Date">
          <daml:equivalentTo>http://daml.umbc.edu/ontologies/
calendar#Date</daml:equivalentTo>
      </daml:Class>
      <daml:Class rdf:ID="String">
          <daml:equivalentTo>http://www.daml.org/2001/03/
daml+oil#Literal</daml:equivalentTo>
      </daml:Class>
</daml:Ontology>


<!-- SubClass of Objects; Certificate  -->
<rdfs:Class rdf:ID="Certificate">
      <rdfs:subClassOf rdf:resource="dtrust:Object"/>
          <rdfs:label>Certificate</rdfs:label>
          <rdfs:comment>
            This subclass contains information about Certificates
          </rdfs:comment>
          <daml:Restriction>
            <daml:onProperty rdf:resource="dtrust:Affiliation"/>
            <daml:toClass rdf:resource="#Organizations"/>
          </daml:Restriction>
</rdfs:Class>

<!-- Properties of Certificates -->
<rdf:Property rdf:ID="CA">
     <rdfs:domain rdf:resource="dtrust:Agent"/>
</rdf:Property>
<rdf:Property rdf:ID="Principal">
    <rdfs:domain rdf:resource="dtrust:Agent"/>
</rdf:Property>
<!-- more properties  ..... -->

<!-- SubClass of Certificates; ID, Trust, Delegation  -->
<rdfs:Class rdf:ID="IDCertificate">
      <rdfs:subClassOf rdf:resource="#Cerificate"/>
      <rdfs:label>IDCertificate</rdfs:label>
      <rdfs:comment>
          This subclass contains information about ID Certificates
      </rdfs:comment>
</rdfs:Class>
<rdfs:Class rdf:ID="TrustCertificate">
      <rdfs:subClassOf rdf:resource="#Cerificate"/>
      <rdfs:label>TrustCertificate</rdfs:label>
      <rdfs:comment>
        This subclass contains information about Trust
        Certificates
      </rdfs:comment>
</rdfs:Class>
<rdfs:Class rdf:ID="DelegationCertificate">
      <rdfs:subClassOf rdf:resource="#Cerificate"/>
      <rdfs:label>DelegationCertificate</rdfs:label>
      <rdfs:comment>
          This subclass contains information about Delegation
          Certificates
      </rdfs:comment>
</rdfs:Class>
```

```
<!-- Properties for Trust Certificate -->
<rdf:Property rdf:ID="Roles">
     <rdfs:domain rdf:resource="dtrust:Object"/>
</rdf:Property>
<rdf:Property rdf:ID="PublicKey">
     <rdfs:domain rdf:resource="dtrust:Object"/>
</rdf:Property>
<rdf:Property rdf:ID="StartDateTime">
     <rdfs:domain rdf:resource="#Date"/>
</rdf:Property>
<rdf:Property rdf:ID="EndDateTime">
     <rdfs:domain rdf:resource="#Date"/>
</rdf:Property>
<!-- more properties ...-->

<!-- Subclass of Object -->
<rdfs:Class rdf:ID="Organization">
     <rdfs:subClassOf rdf:resource="dtrust:Object"/>
     <rdfs:label>Organization</rdfs:label>
      <rdfs:comment>
        This subclass contains information about organizations
      </rdfs:comment>
</rdfs:Class>


<!-- Subclass of Actions; RegisterWithAMS -->
<rdfs:Class rdf:ID="RegisterWithAMS">
        <rdfs:subClassOf rdf:resource="dtrust:Action"/>
        <rdfs:label>RegisterWithAMS</rdfs:label>
     <daml:Restriction>
        <daml:onProperty rdf:resource="dtrust:Actor"/>
        <daml:toClass rdf:resource="dtrust:Agent"/>
     </daml:Restriction>
</rdfs:Class>

<!-- Properties of RegisterWithAMS -->
<rdf:Property rdf:ID="IDCertificate">
        <rdfs:range rdf:resource="#IDCertificate"/>
</rdf:Property>
<rdf:Property rdf:ID="RegisterMessage">
        <rdfs:range rdf:resource="#String"/>
</rdf:Property>
<!-- more properties ... -->

<!-- Subclass of Actions; QueryDF -->
<rdfs:Class rdf:ID="QueryDF">
        <rdfs:subClassOf rdf:resource="dtrust:Action"/>
        <rdfs:label>QueryDF</rdfs:label>
     <daml:Restriction>
        <daml:onProperty rdf:resource="dtrust:Actor"/>
        <daml:toClass rdf:resource="dtrust:Agent"/>
     </daml:Restriction>
</rdfs:Class>

<!-- Properties of QueryDF -->
<rdf:Property rdf:ID="TrustCertificate">
        <rdfs:range rdf:resource="#TrustCertificate"/>
</rdf:Property>
<rdf:Property rdf:ID="QueryString">
        <rdfs:range rdf:resource="#String"/>
</rdf:Property>
<!-- more properties ... -->


</rdf:RDF>
```

Figure 2: This image shows a portion of the Agent System Ontology. Registration of an agent, deregistration of an agent, querying a DF, etc. are all subclasses of the Action class in our Trust Ontology. Similarly, certificates, addresses, organizations etc. are subclasses of the Object class in our Trust Ontology.

the previous delegations, the employees can further delegate this right to other employees, forming a chain of delegation from the buyer's security agent to the seller's security agent to the seller's employees. If at any point a delegation fails or is revoked the access cannot go through. The same holds if the situation is reversed and the supplier gives the buyer access to some of its resources. Delegation chains should always trace back to a security agent to be valid. Security agents are responsible for all accesses originating from its company and act as gateways. All access to information outside the company must go through a security agent. This agent will authenticate the requester, check the delegation chain and verify that the requester has the right to access the requested information. The security agent creates an authorization certificate for the requesting agent, that the requesting agent can use for access.

This framework led us to view trust management as a very effective method for resolving several issues related to security in distributed systems.

10.2 Security for Pervasive Systems

We have designed and implemented Vigil, a security framework, which provides security and access control in pervasive systems [13]. Vigil has been optimized to work in *SmartSpaces*, which is a specific instance of pervasive environments. A SmartSpace environment provides services and resources, that users can access using some short range wireless communications such as Bluetooth, IEEE 802.11, or Infrared, via any hand-held device, within a Vigil can also be used in wired systems, but the focal point of our research is the security in dynamic, mobile systems. Vigil is designed so that clients can move, attach, detach, and re-attach at any point within the framework.

Our infrastructure is designed to minimize the load on portable devices and provide a media independent infrastructure and communication protocol for the provision of services. Vigil, in addition to solving the issue of controlling access to services in a *SmartSpace*, also accommodates users that are foreign entities, that is entities that are not known to the system in advance. In many conventional systems, access rights are static; agents are not able to request permission to access a Service to which they are not preauthorized. To overcome these issues, we have incorporated the *Vigil Security Agent*. This Security Agent allows agents to ask for access permission and other agents to actually delegate rights that they have. This extends the security policy in a secure manner, as only agents that have the permission to delegate, can actually delegate.

The Vigil system is divided into *SmartSpaces*, and each SmartSpace uses one or more security agents to maintain security. The Security Agent is responsible for maintaining distributed trust in the Vigil system. It enforces the security policy of the organization or SmartSpace. It interprets the policy to provide controlled access

to Services and uses distributed trust as a more flexible and easily extensible policy based mechanism. There is generally a global policy associated with the organization and a local policy associated with a SmartSpace. All security agents in the organization will enforce the global policy and will additionally enforce a local policy, which is specific to the Space. A policy includes rules for role assignment, rules for access control, and rules for delegation and revocation.

The Security Agent uses a knowledge base and sophisticated reasoning techniques to handle security and distributed trust. On initialization, it reads the policy and stores it in a Prolog knowledge base. All requests are translated into Prolog, and the knowledge base is queried. The policy contains *permissions* which are access rights associated with roles, and *prohibitions* which are interpreted as negative access rights. The policy also contains rules for role assignments, access control and delegation. A user has the ability to access a service if the user has not been prohibited from accessing the service by an authorized entity and if it either has the role based access right or if some authorized entity has delegated this right to it. An entity can only delegate an access right that it has the ability to delegate.

When a user needs to access a service that it does not have the right to access, it requests another user, who has the right, or the service itself, for the permission to access the Service. If the entity requested does have the permission to delegate the access to the Service, the entity sends a delegate message, signed by its own private key, along with its certificate, to the Security Agent and the requester. The Security Agent checks the roles of the delegator and the delegatee and ensures that the delegator has the right to delegate, and that the delegation follows the security policy. It then adds the permission for the Client to access the Service, but sets a very short period of validity for the permission. Once this period is over, The Security Agent has to reprocess the delegation. This is very useful incase of revoked certificates, delegations or rights. If any one entity in the delegation chain loses the permission, then it is propagated down the chain very quickly, till everyone after the entity loses the ability. Everytime a Service Broker asks about the delegated rights of the client, the Security Agent sends back only valid permissions.

11. SUMMARY

In this paper we present the design for a security framework for multi-agent systems based on trust management, the delegation of permissions and credibility. We believe that other interesting concepts like reputations and obligations can also be built in once the basic framework is developed. This approach is particularly useful in open environment in which agents must interact with other agents with which they are not familiar. Research in security for multi-agent systems often tends to focus on a limited subset of the security challenges of MAS. We believe our model addresses several prominent security issues associated with these agent environments and provides a comprehensive trust based solution.

12. REFERENCES

[1] Tuomas Aura. Distributed Access-Rights Managements with Delegations Certificates. *Secure Internet Programming*, pages 211–235, 1999.

[2] M. Blaze, J. Feigenbaum, J. Ioannidis, and A. Keromytis. The Role of Trust Management in Distributed Systems. *Secure Internet Programming, LNCS vol. 1603, Springer, Berlin, 1999, pages 185-210*, 1999.

[3] Carl Ellison and Bruce Schneier. Ten Risks of PKI: What You're Not Being Told About Public Key Infrastructure. *Computer Security Journal*, 16, 2000.

[4] Ian Horrocks et al. DAML+OIL Language Specifications. http://www.daml.org/ 2000/12/daml+oil-index, 2001.

[5] Foundation for Intelligent Physical Agents. Interaction Protocol Specifications. http://www.fipa.org/repository/ips.html.

[6] Foundation for Intelligent Physical Agents. FIPA Specification. http://www.fipa.org/spec/, 2001.

[7] Foundation for Physical Intelligent Agents. FIPA 98 Specifications Part 10, Version 1.0, Agent Security Management, 1998.

[8] A. Herzberg, Y. Mass, J.Mihaeli, D.Naor, and Y. Ravid. Access Control meets Public Key Infrastructure : Or Assigning Roles to Strangers. In *Proceedings of 2000 IEEE Symposium on Security and Privacy, Oakland, May 2000*, 2000.

[9] Yuh-Jong Hu. Some thoughts on Agent Trust and Delegation. In *Proceedings of Autonomous Agents 2001*, 2001.

[10] Ingersoll Rand (Woodcliff Lake, NJ) and QAD (Carpenteria, CA) and Berclain Group (Schaumburg, IL) and IBM Corporation (Somers, NY). CIIMPLEX Consortium, Consortium for Integrated Intelligent Manufacturing PLanning and EXecution. http://www.ciimplex.org, 2000.

[11] Lalana Kagal, Tim Finin, and Anupam Joshi. Trust based security for pervasive computing enviroments. In *IEEE Communications, December 2001*, 2001.

[12] Lalana Kagal, Tim Finin, and Yun Peng. A Framework for Distributed Trust Management. In *Proceedings of IJCAI-01 Workshop on Autonomy, Delegation and Control*, 2001.

[13] Lalana Kagal, Jeffrey Undercoffer, Filip Perich, Anupam Joshi, and Tim Finin. A Security Architecture Based on Trust Management for Pervasive Computing Systems. In *Proceedings of Grace Hopper Celebration of Women in Computing 2002*, 2001.

[14] Ninghui Li, Benjamin N. Grosof, and Joan Feigenbaum. A Practically Implementable and Tractable Delegation Logic. In *Proceedings of IEEE Symp. on Security and Privacy, held Oakland, CA, USA, May 2000*, 2000.

[15] Yosi Mass and Onn Shehory. Distributed Trust in Open Multi Agent Systems. In *Workshop on Deception, Fraud and Trust in Agent Societies, Autonomous Agents 2000*, 2000.

[16] M.Blaze, J.Feigenbaum, and J.Lacy. Decentralized Trust Management. In *Proceedings of IEEE Conference on Privacy and Security*, 1996.

[17] Stefan Poslad and Monique Calisti. Towards Improved Trust and Security in FIPA Agent Platforms. In *Autonomous Agents 2000 Workshop on Deception, Fraud and Trust in Agent Societies, Spain, 2000*, 2000.

[18] J. R. Searle. Speech Acts : An essay in the Philosophy of Language. Cambridge University Press, 1969.

[19] Jefferey Undercoffer, Andrej Cedilnik, Filip Perich, Lalana Kagal, and Anupam Joshi. A Secure Infrastructure for Service Discovery and Management in Pervasive Computing. *ACM MONET : The Journal of Special Issues on Mobility of Systems, Users, Data and Computing*, 2002.

[20] H.C. Wong and K. Sycara. Adding Security and Trust to Multi-Agent Systems. In *Proceedings of Autonomous Agents '99 Workshop on Deception, Fraud, and Trust in Agent Societies, May, 1999, pp. 149 - 161*, 1999.

Adapted Role-based Access Control for MARISM-A using SPKI Certificates

G. Navarro
Dept. of Computer Science
Universitat Autònoma de Barcelona
Edifici Q - 08193 Bellaterra, Spain
gnavarro@ccd.uab.es

S. Robles
Dept. of Computer Science
Universitat Autònoma de Barcelona
Edifici Q - 08193 Bellaterra, Spain
Sergi.Robles@uab.es

J. Borrell
Dept. of Computer Science
Universitat Autònoma de Barcelona
Edifici Q - 08193 Bellaterra, Spain
Joan.Borrell@uab.es

ABSTRACT

We present an access control method for mobile agent systems. It is based in role-based access control and trust management and provides a flexible and scalable method to control the access to resources. It uses roles and allows the delegation of authorizations to mobile agents. The method uses SPKI to implement the role system and the delegation of authorizations. It is part of the MARISM-A project, a secure mobile agent platform for *Sea of Data* (SoD) applications. We also show its functionality with an example application based in the IST project INTERPRET. It is a medical imaging SoD application, and we provide a suitable solution to control the access to the data.

Keywords

Mobile Agents Security, Role-based Access Control, SPKI.

1. INTRODUCTION

Mobile agent systems are gaining popularity in the last years, allowing the development of new services and applications. Some applications, which are difficult to implement with more traditional programing paradigms, can now be easily implemented with mobile agent systems. One of these applications are known as *Sea of Data* (SoD), applications that need to process huge quantities of distributed data.

With mobile agent systems, we do not need to send the data across a network and centralize all the data processing. Instead, the code is executed where the data is located. The initial launching platform does not need to be always on-line to access the remote resources, so the user may be disconnected during the execution of the application. It is also possible to parallelize the execution of processes allowing a high degree of scalability.

One of the most important challenges of mobile agent systems is the security. An important security service that needs to be achieved, specially in SoD applications, is the resource access control. We need a lightweight, flexible and scalable method to control the access to data and resources in general. Traditional methods are normally based in the authentication of global identities (X.509 certificates). They allow to explicitly limit access to a given resource, through attribute certificates or ACLs. So they also require a certification authority and a centralized control.

An alternative to implement the access control are the authorization infrastructures. These infrastructures are based on trust management and allow to assign authorizations (permissions or credentials) to entities and delegate trust from one entity to another. One of these infrastructures is the *Simple Public Key Infrastructure* (SPKI) [6], which seem to be the most accepted. We think that SPKI provides a good base to implement the access control method. There are existing security frameworks providing SPKI functionalities [10], and it is probably the most standard solution to implement trust management mechanisms such as the delegation of authorizations. There is also some propositions to use authorization infrastructures to implement access control methods [2], [11].

We present a resource access control method for mobile agent systems. It is based in roles and trust management. It allows to control the access to resources based on role membership, as in systems such as *Role-base Access Control* (RBAC) [14], [18] , which facilitates the management of the access control. An important feature of our model, not provided by general role-based access control, is the possibility of delegating trust to manage and control the access. This way, it does not need a certification authority or other trusted third parties. It makes the system scalable, and allows the distribution of some of the main tasks for controlling and managing the access.

The particularities of mobile agent systems introduce some restrictions and limitations, not found in more classical systems (distributed or not). Specially when considering the security involved in a mobile agent. Our model allows to authorize a mobile agent to access a given resource and control its access with quite flexibility. The mobile agent does not need to carry any kind of information with regard to the resource access. This avoids the inconveniences of storing sensitive information in the mobile agent.

The model is going to be implemented in the MARISM-A (*An Architecture for Mobile Agents with Recursive Itineraries and Secure Migration*) project [3], a secure mobile agent platform for SoD applications. To clarify and explain our proposal, we will explain an example application based in the IST project INTERPRET (*International Network for Pattern Recognition of Tumors Using Magnetic Resonance*)

[1].

In Section 2 of the paper we introduce the environment of our proposition. Section 3 gives a brief overview of SPKI. We present our model in Section 4 and the example application in Section 5. Section 6 explains the main functionality of the proposed model and finally, Section 7 contains our conclusions.

2. MARIMS-A EXTENSION

As said before, the proposed access control model is an extension of the MARISM-A platform [17]. MARISM-A is a secure mobile agent platform implemented in Java. It is implemented on top of the FIPA-OS system [7], which follows the standards proposed by FIPA [8].

The basic element of the MARISM-A platform is the agency, the environment for the execution of agents. An agency consists of a directory service, an agent manager and a message transport service. An agent system has several agencies distributed on a network. Each agency in controlled by an entity (its owner).

Agents in MARISM-A can be mobile or static, depending on the need of the agent to visit other agencies to fulfill its task. There are several types of mobile agents according to the characteristics of its architecture: basic or recursive structure, plain or encrypted, itinerary representation method, etc. Agents can communicate each other through the agency communication service.

All mobile agent architectures in MARISM-A share some basic aspects, such as the differentiation of internal parts and migration mechanisms. A mobile agent consists of code, data, state, and an explicit itinerary. Code is the set of instruction describing the execution of the agent. Data is an information storage area that can be used by the agent at any moment for reading and writing and goes with it all the time. Results of executions are stored in this area, normally using some convenient protection mechanisms. State is reserved to store the agent information related with its state. Explicit itinerary is a structure containing all agencies that are going to be visited by the agent on its life cycle [13]. Itineraries consist of several basic structures: sequences, sets and alternatives. These structures can be combined to build complex itineraries. In a sequence, the agent will migrate to each agency one after the other. In a set, a group of agencies will be visited by the agent in no special order. On the other hand, only one agency of those listed in an alternative will be visited by an agent, depending on some conditions.

MARISM-A considers a minimal security infrastructure to protect the communications between agencies. All the agencies are registered in a CA, and we use SSL to provide both confidentiality and authentication for agency communications.

It is important to assume that agencies untrust each other. Therefore, they might try to modify results carried by the agent, or to gain knowledge about its itinerary, to favor themselves to the detriment of the rest. It is also reasonable to assume that agencies are not malicious and they do not seek to adversely affect the owner of the agent (the client), or the agent itself.

From now on, we will use the following notation:

- $E_i(m)$: encryption of m using a symmetric cipher with i's secret key.
- $P_i(m)$: encryption of m using an asymmetric cipher with i's public key.
- $S_i(m)$: signature of m using i's private key.
- $hash(m)$: hash function of m.
- $hash_i(m)$: keyed hash function of m using i's secret key.

Subsections 2.1 and 2.2 introduce the architecture of the static agents and mobile agents with explicit itinerary as an extension to MARISM-A mobile agents.

2.1 Static Agents

A MARISM-A static agent has the following form:

$$\text{Agent} = \text{ControlCode, State, Code, Data}$$

Because agent control code is in the agent itself, it is indifferent for the platform to deal with mobile or static agents. There are not many words to say about security in static agents. Communication and interface with other agents are provided by secure services of the agency. Data protection is assured by the agency too, and there is no itinerary to protect here.

2.2 Mobile Agents with explicit itinerary

Agent code is split into several pieces in this architecture. There is a main code that will be executed in all agencies (Common Code), and as many code fragments as agencies are in the itinerary, each one to be executed in a particular agency (Local Code). This feature makes MARISM-A very useful in some types of application where execution is context dependent. We consider the following mobile agent architecture:

$$\begin{aligned} \text{Agent}_i = \;& \text{PubKey}_o, \text{ControlCode, StateData,} \\ & \text{CommonCode, GlobalData, Itinerary} \\ \text{Itinerary} = \;& (\text{LocalCode}_1, \text{LocalData}_1, \text{Agencies}_1), \ldots, \\ & (\text{LocalCode}_n, \text{LocalData}_n, \text{Agencies}_n) \end{aligned}$$

Agencies_i is the agency (or agencies, depending on the type of itinerary) the agent is going to visit (migrate) next. The agent that is sent to the next hop of the itinerary (Agent_{i+1}) has the same structure. CommonCode is executed by all agencies when the agent immigrates and before the specific LocalCode. Programming is simplified by using this common code to include the non agency dependent code only once. The control code in the agent deals with the functions of agent management, in this case extracting the relevant parts of the agent. PubKey_o is a public key provided by the owner.

It might be interesting to protect integrity and secrecy of data that has been written in some agency. In an e-commerce application, for instance, where agencies represent shops and agents act on behalf of buyers, it might be necessary to protect offers from rival shops. The method to provide the secrecy and integrity required for this data in this agent architecture is based on a hash chain. Some of the data area is reserved to store results from executions (Results Data). Results are not stored plain, but they are firstly encrypted using agent's owner cryptographic information. Only the owner of the agent will be able to read these results. Once the result has been written a hash of the Result and previous hashed information is calculated, signed and written also. This hash has information about the identity of next agency in the itinerary, so that no agency can neither modify the result area nor remove some result. Each agency verifies during immigration that all hashes in the Results Data are correct. The format of the Results Data is:

$$\begin{aligned}\text{Results Data} = \; & P_o(nil, Id_1), S_o(hash(P_o(nil, Id_1))), \\ & P_o(R_1, Id_2), S_1(hash(P_o(R_1, Id_2))), \\ & P_o(R_2, Id_3), S_2(hash(P_o(R_2, Id_3))), \ldots, \\ & P_o(R_n, Id_o), S_n(hash(P_o(R_n, Id_o)))\end{aligned}$$

where o is the owner of the mobile agent; R_i is the result of agency i and Id_i is the identifier of the agency i.

We also need a way to ensure the agent's integrity. The owner, before sending the agent, computes a keyed hash of the Control Code, the Common Code and the Itinerary of the agent ($hash_{K_o}(ControlCode, CommonCode, Itinerary)$). Then, when the agent finishes its execution, the owner can verify the agent's keyed hash.

To protect the itinerary we use the following encryption schema:

$$\begin{aligned}\text{Agent}_i = \; & \text{PubKey}_o, \text{ControlCode, StateData,} \\ & \text{CommonCode, GlobalData, Itinerary,} \\ & hash_{K_o}(\text{ControlCode, CommonCode, Itinerary}) \\ \text{LocalStructures} = \; & E_1(\text{LocalCode}_1, \text{LocalData}_1, \\ & \quad \text{Agencies}_1, tripmark), \ldots, \\ & E_n(\text{LocalCode}_n, \text{LocalData}_n, \\ & \quad \text{Agencies}_n, tripmark)\end{aligned}$$

where *tripmark* is usually a timestamp or nonce, which identifies the agent journey and prevents replay attacks. As we will see, the encryption is performed by the agency itself before the whole agent is constructed. So the symmetric key is only used by the agency and it does not need to be distributed. Note that the keyed hash in the agent is only useful to the owner, thus it does not need to be included in the mobile agent. We show it in the agent definition just for clarity reasons.

A variant of this agent is the mixed one, where the list of information for agencies is scrambled. This makes it not possible to know which is the part of the agent that will be executed on which agency.

3. SPKI

The base to our proposal is SPKI (more formally named SPKI/SDSI). It is an infrastructure which provides an authorization system based in the delegation of authorizations and a local name model. It provides mainly two kind of certificates, authorization and name certificates. Any individual, software agent or active entity in general is called a *principal*. It is a *key-oriented* system, each principal is represented and may be globally identified by its public key. We can say that in SPKI a principal is its public key. Since it does not need a certification authority, each principal can generate and manage its keys. A key is a generic cryptographic key pair (public and private). Currently the SPKI specification supports RSA and DSA keys. The representation format used by SPKI is S-expressions [16].

An authorization certificate has the following fields:

- Issuer: principal granting the authorization.
- Subject: principal receiving the authorization.
- Authorization tag: specific authorization granted by the certificate.
- Delegation bit: if it is active, the subject may forward delegate the authorization received.
- Validity specification: specifies the validity of the certificate through a time range and on-line tests.

It is signed by the issuer. The on-line tests from the validity specification field, provide the possibility of checking, at verification time, the validity or revocation of the certificate.

In a normal situation there will be a principal controlling a resource, which delegates an authorization. The authorization may be further delegated to other principals. If a principal wants to access the resource, it needs to provide an *authorization proof*. Te proof is a certificate chain, which binds the principal controlling the resource to the one requesting the access. To find this certificate chain there is a deterministic algorithm, *Certificate Chain Discovery Algorithm*[5], which finds the authorization proof in polynomial time.

In SPKI a principal may have a local name space and define local names to refer to other principals. To define a name, a principal issues a name certificate. It has an issuer, subject, validity specification, (just as an authorization certificate) and a name. The *issuer* defines the *name* to be equivalent to the *subject*. For example a principal with public key K may define the name *Alice* to be equivalent to the the principal with public key K_A. Now K can refer to the principal K_A by the name *Alice* instead of the public key. Such a name certificate can be denoted as:

$$K \; Alice \longrightarrow K_A$$

meaning that K defines the name *Alice* in its local name space to be equivalent to K_A. If a principal wants to refer to a name defined in another name space, it just has to add

the local name space owner's public key to the name as a prefix. When we say K_A *Alice*, we mean the name *Alice* defined in K_A's local name space.

SPKI also provides the ability of defining compound names. Names that refer to other names which may also reference other names and so on. For example, the principal K_B can define the following name in its local name space:

$$K_B \ employee \longrightarrow K \ Alice$$

It defines the name *employee* to be equivalent to the name *Alice* defined in K's local name space. Note that it is referring to K_A without knowing it.

This is a key concept in our proposal since we will consider a role as a SPKI local name.

4. OUR ACCESS CONTROL MODEL

One of the first problems we found when planning the authorization model, is if the mobile agents should have a SPKI key and be considered as principals. A mobile agent cannot trivially store a private key, so it cannot perform cryptographic operations such as digital signatures. There are some propositions to store sensitive information (private keys) in mobile agents [15]. But the problem arises when the mobile agent uses the private key to compute a cryptographic operation. The agency where the agent is in execution will be able to see the private key. As a result we consider that a mobile agent should not have a private key.

Our solution is to delegate authorizations directly to the agent. This way the mobile agent does not need to carry any kind of authorization information, making the agent more simple and lightweight. This issue will be discussed in Section 6.2.

The main components of the access control method can be seen as independent modules. Each module is implemented as a static agent, has a SPKI key, and it is considered as a SPKI principal. The modules are:

Authorization Manager (AM) it manages the delegation of authorizations, issuing SPKI authorization certificates. It follows a *local authorization policy.*

Role Manager (RM) it manages the roles (mainly the role membership) by issuing name certificates. It follows a *local role policy.*

Certificate Repository Manager (CRM) it manages a certificate repository. Provides services such as certificate chain discovery.

Resource Manager (DM) it is an authorization manager, which controls a resource (data), it has to verify resource access requests. Normally its authorization policy will be quite restrictive, delegating to an authorization manager the responsibility of performing complex authorization tasks.

Figure 1 shows a simple schema of the model with two DMs, an AM, a RM and a CRM. The RM defines the roles and determines its membership. The DMs delegate the authorizations related to the resources to the AM, and the AM delegates authorizations to the roles. Each static agent stores the issued SPKI certificates in the certificate repository through the CRM (denoted by broken lines).

4.1 Authorization Manager (AM)

The main functionality of the AM is to provide authorization certificates under request. To obtain an authorization certificate, a principal sends a request to the AM with the specific authorization, it wants to obtain. Then the AM decides whether to issue the certificate or not, and under what conditions it has to be issued. To do that, it follows its local authorization policy. Since the policy is local to the AM agent, it does not need to follow any specification and its format is implementation dependent.

We propose an authorization policy, described as a SPKI ACL, where each rule can be expressed as an ACL entry in S-expression format. A SPKI ACL entry is an authorization certificate without the issuer and it does not need to be signed because it is local to the AM and stored in secure memory. It has the following fields:

- Subject: the principal receiving the authorization. It may be a role or another AM.
- Authorization tag: determines the specific authorization that the subject can obtain. SPKI allows quite flexibility to specify the authorization tag with S-expressions.
- Delegation bit: determines whether the subject may receive the right to delegate the authorization or not.
- Validity specification: allows to limit the authorization to a time range, and include some on-line tests to verify the validity or revocation of the authorization certificate.

To be more specifics, the AM will receive authorization delegation requests from a RM or another AMs. It has to delegate authorizations to roles or to other AM which will finally authorize roles.

4.2 Role Manager (RM)

The RM is responsible for assigning and managing roles, and determines the role membership. The use of roles facilitates the access control management and the specification of policies. The main idea is to exploit the advantages of Role Based Access Control (RBAC) [14] and trust management. The RM assigns a role by issuing a SPKI name certificates following its local role policy. It can also assign a role to a role defined by another RM, thus allowing the delegation of role membership management. Section 6.1 details how roles are assigned and used.

Each RM has a local role policy which determines what roles does it manage. It also includes rules to determine if a given principal requesting a role membership has to be granted or

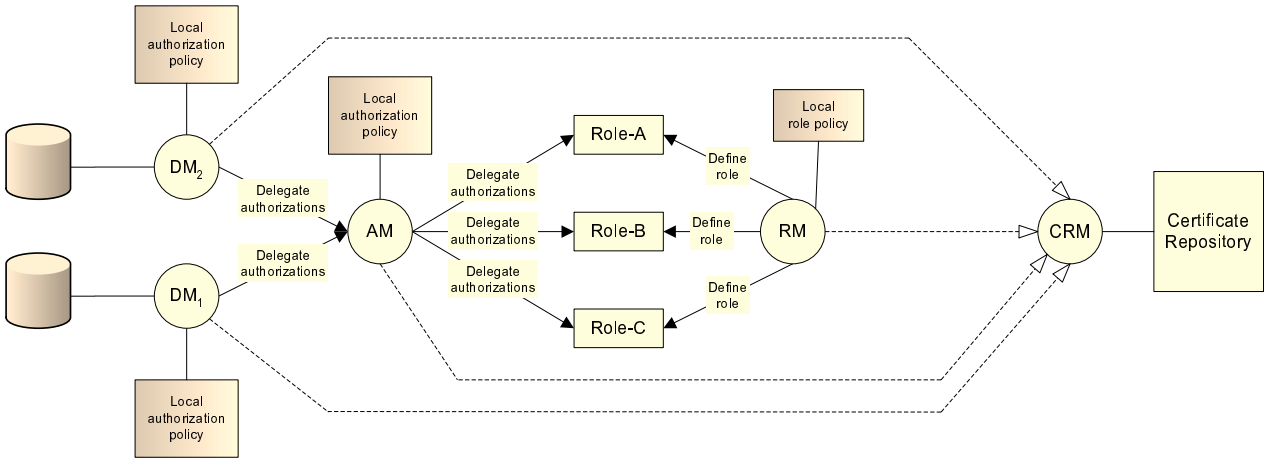

Figure 1: Authorization modules

not. If we choose to describe the role policy as a SPKI ACL, it is quite similar to an authorization policy. Now the subject of the SPKI ACL entry is a principal or another role and the authorization tag determines the role that the subject can have. This local policy also provides rules for the RM to define role hierarchies an constraints over the user assignment to roles if needed.

4.3 Certificate Repository Manager (CRM)

A CRM implements a certificate repository. For example, one agency may have one CRM to collect all the certificates issued by agents inside the agency. The CRM provides the repository and all the services needed to query, store or retrieve the certificates in the repository. It also provides a certificate chain discovery service. A principal can make a query to the CRM to find a specific certificate chain. This way we solve the problems derived from certificate distribution and leave the task to perform chain discoveries to the CRM and not to the other principals. It decreases the communication traffic, certificates do not need to travel from one principal to another and reduces the task that generic principals need to perform.

4.4 Resource Manager (DM)

The main task of a DM is to control the access to a resource (data). It holds the master SPKI key to access the resource, delegates authorizations to AMs, and verifies that an agent requesting access to the resource has the proper authorization. Another important feature of a DM is to issue *Certificate Result Certificates* (CRC) to agent hashes, see 6.2.

As it has to delegate authorizations issuing authorization certificates it also acts like a AM and follows a local authorization policy. But this policy is much more restricted. A DM only has to issue authorization certificates to AMs and a special certificate to mobile agents (see 6.2), which are quite straightforward operations.

5. EXAMPLE APPLICATION

This example is derived from the project IST-1999-10310 INTERPRET [1]. The example is going to be developed using the MARISM-A platform. Consider a medical SoD application for radiology images. There are several hospitals, research centers and companies with a radiology department which produces some kind of sensitive, and possibly expensive, radiology images such a magnetic resonances or high resolution radiologies. Each center organizes the data in a database accessed by at least one agency with DMs. The application may provide the ability for clients to process the distributed data in a variety of ways, for example testing a classification algorithm. The owner of the data may also provide classification services, such as a trained classification algorithm, which a client may use to classify a reduced set of data provided by herself.

The reason for using a mobile agent approach in this application, is due to the high quantity of distributed data, which is difficult to centralize. Also because medical data normally contains some sensitive information, which the hospitals are normally not allowed to give it to someone else. That is, a mobile agent processing the data, may get back to the client with the obtained results, but not with the data.

We will consider each participating entity as a principal. A principal may be a static agent or an individual (normally the owner of a mobile agent) with its own SPKI key. We consider three kinds of principals, data producers, data consumers and process consumers:

- *data producer*: updates the database, adding new images or replacing existing ones.
- *process consumer*: provides a reduced set of data, and wants to use some processing service provided by the agency (normally a complex trained algorithm such as a classification one).
- *data consumer*: it provides a code to be executed with the data provided by the agency.

A simple definition of roles for the example application may be:

- *physician*: authorized as data and process consumer for all the resources.
- *external_physician*: authorized as process consumer for a reduced set of data.
- *radiography_technologist*: authorized as a data provider.
- *external_researcher*: authorized as data consumer for a restricted set of data.

These roles may be hierarchically extended, for example as *radiography_technologist*, there may be *radiographer*, which provides only radiographies and *mr_technologist*, which provides only magnetic resonances. Specially the *external_researcher* role, which may be seen as a client, may have several sub-roles to be able to specify several specific authorizations for different kinds of clients. The definition of role hierarchies is quite application dependent. Thus, we do not explicitly specify any role schema. In some specific environments the role definition will not require the use of hierarchies or constraints over the assignment of role membership.

6. ACCESS CONTROL MANAGEMENT

Given the example application we will show the functionality of the access control method. The main features are the role system and the delegation of authorizations to mobile agents. A principal may be authorized to access a resource as a role member. The AM may give several authorizations to a specific role. Then a principal belonging to that role, has all the authorizations of the role. We already said that we do not consider a mobile agent as a SPKI principal. Thus we need a way to authorize mobile agents and control its access to resources.

We also consider the distribution of the access control management by distributing some of the modules. We can distribute several modules, or just one, for example. This makes the model easily adaptable to specific applications. Since a module is implemented in a static agent, to distribute a module means to use several static agents, which may operate independently.

6.1 Roles

An important issue of the RM is that it is the main responsible to grant access permissions to principals. When a principal requests a role membership and succeeds, it automatically has all the authorizations of the role. The main task of the RM is to deal with role management. This involves three main tasks:

- Role definition and membership management.
- Role hierarchies definition.
- Apply constraint to the user assignment to roles.

The constraints determine mutually exclusive roles or subsets of roles. These constraints are what the proposed NIST standard calls *Separation of Duties*[14]. The hierarchies are defined by issuing name certificates. For example, consider that the role *radiography_technologist* inherits all the permissions of the role *mr_technologist*. If both roles are defined by RM_A, it will issue the following name certificate:

$$RM_A\ mr\_technologist \longrightarrow RM_A\ radiography\_technologist$$

Another important issue is that the role membership can be restricted through the validity specification of the name certificate, which grants the membership. That is, it can have a *not-after* and *not-before* time range and some on-line tests [6].

6.2 Authorizing mobile agents

A client, as a principal, may be member of a role or roles, say *external_client*. It may be authorized to access resource A with a mobile agent. Since mobile agents cannot have private keys, we can not delegate authorizations to the mobile agent or make it member of a role. Our approach is to delegate the authorization to a hash of the agent. The subject of a SPKI authorization certificate and any SPKI principal in general can be a public key or a hash of a public key. So a hash may be seen as a principal, subject of a certificate. This idea does not really follow the SPKI specifications. Since the subject is not the hash of a public key, it is not a principal. Thus we need to extend the SPKI specifications to introduce this idea.

As we said before the mobile agent is constructed from the itinerary, separately including the code to be executed in each agency. Let m_i be the local structure of the agent to be executed in the agency i. That is:

$$m_i = E_i(LocalCode_i, LocalData_i, Agencies_{i+1}, tripmark)$$

The client already has an authorization to access resource A, which is controlled by DM_A. Once the client has specified all the m_is it constructs the itinerary and proceeds to get the authorization for the agent. The main idea is to request a *Certificate Result Certificate*(CRC) to DM_A having the hash of m_i as the subject of the certificate. The CRC is an authorization certificate, which resumes a certificate chain, in this case the authorization proof for the client to access resource A. The process involves the following steps:

1. The client sends a CRC-request to DM_A. It includes the specific authorization it wants to obtain, the code m_i and the client's public key. This request is signed by the client.
2. The DM_A requests the CRM to verify if the client is authorized to access the resource. That is, verifies if there is an authorization proof which allows the client to access the resource.
3. If the authorization is correctly verified, the DM_A computes the hash of the code, and issues an authorization certificate which has DM_A as the issuer and the hash of the code as the subject. The specification tag and the validity specification is the intersection between the ones from the client's CRC-request and the ones returned in the authorization proof request.
4. Finally the DM_A encrypts the code m_i with a symmetric cipher. It uses a secret key only known by the

DM_A. The DM_A is the only one who is able to decrypt m_i.

Once the mobile agent is constructed it will be able to access the resource. The mobile agent will travel to the agency and request access to DM_A. The DM_A just has to compute the hash of the agent code (m_i) and check if there is an authorization certificate, which directly authorizes the hash to access. This authorization verification is straightforward, since it does not require the generation of a full authorization proof.

This approach allows to delegate authorizations to mobile agents. Note that the mobile agent does not need to include any kind of authorization information, it just has to provide the specific code so the DM_A can compute the hash.

One thing we have not explicitly talked about is how to control the proper behavior of the mobile agents. In our example, how do we know that a mobile agent is not *stealing* data?. First of all, the process of authorizing a mobile agent involves the computing of the hash of the piece of code of the agent, which is going to be executed in the agency. Therefore, we can easily log this code for auditing purposes. It is also feasible for an agency to include a local monitoring system looking for anomalies in the behavior of the agents.

6.3 Distribution of Role Management

Due to the local names provided by SPKI, the role management can be easily distributed. We can have several RMs managing its local roles and using compound names to reference one local role to another. For example, consider we have two RMs, RM_A and RM_B. Each one has its local roles definitions, RM_A may define:

$$RM_A\ radiography\_technologist \longrightarrow K_1$$
$$RM_A\ physician \longrightarrow K_2$$
$$RM_A\ physician \longrightarrow K_3$$
$$RM_A\ companyB\_client \longrightarrow RM_B\ ext.\_researcher$$

That is, it says that the principal K_1 is member of the *radiology_technologist* role; the principals K_2 and K_3 are members of the role *physician*. And that the name *external_researcher* (which is also a role) defined in the local name space of RM_B is member of the role *companyB_client*. Then RM_B may define:

$$RM_B\ external\_researcher \longrightarrow K_4$$
$$RM_B\ external\_researcher \longrightarrow K_5$$

So the principals K_4 and K_5 are members of the role *external_researcher* defined by RM_B. And they are also members of the role *companyB_client* defined by RM_A. Note that each RM defines independent roles, both RMs could define locally two roles with the same name, and they will be considered as different roles by the system. Is important to notice that all the roles, as SPKI names, are local to each RM. We can globally identify the role by adding the public key of the RM as a prefix of the role (just as a SPKI name). This independence of role definitions makes the system easily scalable and distributed. Note that in the example we can say that the role management is distributed over the two RMs since both of them take part in the hole role management. So independent RMs can interact in the same model without having to redefine roles.

This can be also seen as trust management, in some way RM_A trusts RM_B to manage the role RM_A *companyB_client*. From the RBAC point of view it is considered a role hierarchy definition. We can say that the role RM_B *external_researcher* inherits all the permissions (authorizations) of the role RM_A *companyB_client*.

6.4 Distribution of Authorization Management

The distribution of the authorization management is achieved by distributing the management over several AMs. This distribution is straightforward. Each AM manages authorizations following its local policy. It can only delegate the authorizations that it has received. To be more precise an AM or any principal may delegate a certificate granting an authorization it does not have. But any principal receiving the authorization will not be able to have the proper authorization proof, since the certificate chain will be broken.

There will be no conflict between several AMs. If there is an authorization proof for one principal to access a resource, the principal will be able to access no matter which AMs or principals have interfered.

6.5 Distribution of the Certificate Repository

The distribution of the certificate repository is a complex task. All the authorization proofs are obtained from the repository. In fact, it is the CRM which performs the certificate chain discovery. To distribute the repository will considerably increase the complexity. We need to use a distributed certificate chain discovery algorithm, which adds not only complexity to the implementation but also introduces the need for more communication and process resources.

There is some work done in relation to distributed certificate repositories and chain discovery, such as dRBAC [9] or [12]. These approaches could be used if an specific application really needs to distribute the certificate repository.

The application we are going to implement does not impose the distribution of the certificate as a must. In fact, it can easily be implemented with a centralized repository. And there is no need to add complexity to the system by distributing the repository.

7. CONCLUSIONS

We have proposed an access control model for mobile agent systems, specially suitable for SoD application. It provides a simple, flexible and scalable way of controlling the access to resources. It takes the advantages of RBAC and trust management ideas. The proposed model is part of the MARISM-A project. A secure mobile agent platform for SoD applications. We have also introduced an example application, a medical SoD imaging application based

in the IST project INTERPRET. Even though, there are some problems which are still unsolved, like malicious hosts acting against agents, which are still open problems [4].

We are working on the implementation of the proposed model. This process involves the study of subtle aspects, which still are open question. For example considering alternatives to implement the local policies. By using SPKI ACLs, the policy is based in SPKI keys. This may be reflected in limitations of the key management. We also want to consider issues such as anonymity, specially relevant in key-oriented systems.

8. ACKNOWLEDGMENTS

This work has been partially funded by the Spanish Government Commission CICYT, through its grant TIC2000-0232-P4-02, and Catalan Government Department DURSI, with grant 2001SGR 00219.

9. REFERENCES

[1] INTERPRET Project - IST-1999-10310. International Network for Patern Recognition of Tumors Using Magnetic Resonance. 1999 `http://carbon.uab.es/INTERPRET`.

[2] T. Aura. Distributed access-rights management with delegation certificates. In J. Vitek and C. Jensen, editors, *Secure Internet Programming: Security Issues for Distributed and Mobile Objects*, LNCS 1603, pages 211–235. Springer Verlag, 1999.

[3] CCD Research Group. MARISM-A, An Architecture for Mobile Agents with Recursive Itineraries and Secure Migration. `http://www.marism-a.org`.

[4] D. Chess. Security issues of mobile agents. In *Mobile Agents*, volume 1477 of *LNCS*, pages 1–12. Springer-Verlang, 1998.

[5] D. Clarke, J. Elien, C. Ellison, M. Fredette, A. Morcos, and R. Rivest. Certificate chain discovery in SPKI/SDSI. *Journal of Computer Security*, 9(9):285–322, 2001.

[6] C. Ellison, B. Frantz, B. Lampson, R. Rivest, B. Thomas, and T. Ylonen. RFC 2693: SPKI certificate theory. The Internet Society, September 1999.

[7] Emorphia. FIPA-OS. `http://fipa-os.sourceforge.net`.

[8] Foundation for Intelligent Physical Agents. FIPA Specifications, 2000. `http://www.FIPA.org`.

[9] E. Freudenthal, T. Pesin, L. Port, E. Keenan, and V. Karamcheti. dRBAC: Distributed role-based access control for dynamic coalition environments. New York University, Technical Report TR2001-819.(to appear ICDCS 2002), 2001.

[10] Intel Architecture Labs. Intel Common Data Security Architecture. `http://developer.intel.com/ial/security/`.

[11] L. Kagal, T. Finn, and A. Joshi. Trust-Based Security in Pervasive Computing Environments. *IEEE Computer*, pages 154–157, Dec. 2001.

[12] N. Li, W. Winsborough, and J. Mitchell. Distributed credential chain discovery in trust management. Accepted for publication in Journal of Computer Security, Nov. 2001.

[13] J. Mir and J. Borrell. Protecting general flexible itineraries of mobile agents. In *Proceedings of ICISC 2001*, LNCS 2288. Springer Verlag, 2002.

[14] D. Rerraiolo, R. Sandhu, S. Gavrila, D. Kuhn, and R. Chandramouli. Proposed NIST standard for role-based access control. In *ACM Transactions on Information and System Security*, volume 4, pages 224–274, August 2001.

[15] J. Riordan and B. Schneier. Environmental key generation towards clueless agents. In *Mobile Agents and Security*, pages 15–24, 1998.

[16] R. Rivest. S-expressions. Internet-draft: The Internet Society, 1997.

[17] S. Robles, J. Mir, and J. Borrell. Marism-a: An architecture for mobile agents with recursive itinerary and secure migration. In *2nd. IW on Security of Mobile Multiagent Systems*, Bologna, Italy, 2002.

[18] R. Sandhu, E. Coyne, H. Feinstein, and C. Youman. Role-Based Access Control Models. *IEEE Computer*, pages 38–47, February 1996.

A Security Framework for a Mobile Agent System

Guido van 't Noordende
Computer Systems Group
Faculty of Sciences
Vrije Universiteit Amsterdam
The Netherlands
guido@cs.vu.nl

Frances M.T. Brazier*
Interactive Intelligent
Distributed Systems
Faculty of Sciences
Vrije Universiteit Amsterdam
The Netherlands
frances@cs.vu.nl

Andrew S. Tanenbaum
Computer Systems Group
Faculty of Sciences
Vrije Universiteit Amsterdam
The Netherlands
ast@cs.vu.nl

ABSTRACT

The Mansion paradigm provides a logical model for designing distributed multi-agent applications. Mansion is designed to be a scalable, secure and extensible system for supporting multi-agent applications. This paper presents the security architecture of Mansion.

An Agent Container (AC) allows for secure transport and flexible storage of heterogenous agents and data. The AC uses lists of trusted hosts, fixed rules about how persistent and transient segments are handled, and possibly policies that describe the allowed changes to the AC at trusted destinations. A secure handoff protocol is presented as part of the agent transfer protocol, that allows for on-the-fly detection of malicious alterations to an AC.

Mansion provides protection of agents, hosts and information in the system. Avoidance of security risks, and (audit) mechanisms to detect malicious actions of entities in the system are important mechanisms used to protect the system.

Keywords

Mobile Agents, Multi-Agent ystems, Middleware, ecurity, Distributed ystems, Agent Transfer Protocol, Audit Trail

1. INTRODUCTION

Mansion is a system aimed at supporting heterogenous, large-scale distributed mobile agent applications. Mobile agents have a number of well-described advantages over traditional distributed systems [1]. The most significant of these is that an agent can move its computation to the resource or data which it needs, which alleviates problems due to latency or bandwidth limitations.

There are a number of solutions for Multi-Agent ystems (MA es), most of which provide little structure to application developers. Most existing MA es provide some form of security to agents and machines in the system, but typically those solutions are tied to a single programming language (e.g., Java) [2, 3, 4]. Mansion is a MA which provides a clear paradigm for designing multi-agent applications.

Important aspects for security in this system are: protection of agents, protection of hosts, protection of information and protection of the middleware and resources. This paper first introduces the logical and physical model underlying Mansion. Then we will explain how the identified security areas are addressed in Mansion, followed by a discussion and related work.

*This research has been funded by tichting NLnet

2. THE MANSION FRAMEWORK

The Mansion framework consists of a logical and a physical model. The logical model is used to structure an application and to provide a consistent view of this application to agent programmers. The physical model underlies the logical model, and consists of a network of (heterogenous) hosts on which the logical model is mapped. Mansion provides a (distributed) middleware which provides an interface (API) to agents which they can use to interact with the system and hides distribution and location aspects of the system.

2.1 Logical Model

An application in our framework is modeled as a closed world containing a set of hyperlinked rooms. Entities in a room can be agents, objects, or hyperlinks. Each agent is a (possibly multithreaded) process running on one host. No part of the internal process state of an agent can be accessed from the outside by other agents. Objects are strictly passive: they consist of data and code hidden by an interface. Hyperlinks determine how the rooms in a world are connected.

An agent is injected into a world (by its owner) through a world entry daemon (WED). Each world has one or more entry rooms, each of which contains a WED. The WED does some security and consistency checks on each injected agent, and places the agent in its entry room. Once in an entry room, an agent may follow a hyperlink to go to another room.

Once in a room, an agent can access a special object, called *Room Monitor Object (RMO)*. The RMO registers all content in the room, and provides an interface for agents to interact with the room. Descriptions of entities in a room (e.g., agents, objects and hyperlinks to other rooms) are specified in *Attribute Sets (ASes)* which can be obtained through the RMO interface. The RMO Interface is automatically loaded in an agent's address space when it enters a room.

Each entity has a (possibly empty) attribute set, placed in the RMO, using which the entity is described. Example attributes are the name of an agent or the coordinates of an object in a room. Attribute sets are represented as a set of (entity, attribute, value) triples. The attributes of an A are typically predefined in a world, but they may be extensible in some applications. An agent can alter an attribute set if it is the owner of the entity.

Each object in a room can also contain an attribute set internally, independent of the RMO, which specifies additional (private) attributes. An event-mechanism can be provided by objects (including the RMO), based on attribute set matching, comparable to template-tuple matching in Linda or Javaspaces [5].

Every world can also have an *attic*. The attic contains global services and is directly accessible to agents in any room. Through the attic, an agent can obtain world-scoped information, for example, the topology (hyperlink layout) of a world, directory services, or a bulletin board service (e.g., for publishing agent information). ervices in the attic are provided as attic objects or attic agents. Attic agents are the only agents allowed in the attic. Other agents cannot move to the attic.

Each world has a *basement,* which keeps track of the information needed to make the world function, such as the location of the agents. When an agent enters a world, it is assigned a Global Agent ID (GAID), which is registered in the basement. The basement is not visible to agents.

2.2 Examples

As an example of the Mansion paradigm, consider a shopping mall world. A shopping mall can be modeled as a number of separate stores, which each consist of a set of hyperlinked rooms, one or more of which are entry-points to the store. An entry room to the mall is provided by the world owner, which provides hyperlinks to the (entry) rooms of the shops in the mall. In each room, there may be objects that represent items (for example clipart or music) for sale, and shopkeeper agents which can be queried for information or be involved in commercial transactions. Agents that represent users can roam through the mall to find items to their liking. Agents can communicate with each other or their owner to speed up their search or notify each other of interesting bargains. An agent may take some form of digital cash with it to be able to buy items for its owner. Items that are bought by an agent can be transported to their owner by means of inter-agent communication or as part of the agent.

Other examples are: Multi-User Dungeons (MUDs), in which players have to find their way through a maze of rooms, in which they can find items and may meet many adversaries; a virtual learning environment, where users can move among classrooms; and a library world, where (groups of) rooms represent sections for different topics.

In short, the Mansion paradigm replaces the World Wide Web paradigm of a collection of hyperlinked documents that users can inspect with that of a collection of hyperlinked rooms in which agents can meet to do business.

2.3 Physical Model

A world may be spread over multiple machines. In particular, a room can be distributed over multiple machines by means of distribution of the RMO and other objects in the room. Globe is an object architecture [6] which can be used to distribute passive objects over multiple hosts (e.g., using active replication). This can provide reliability (availability) and locality of data in the system.

The basement is a central component in each world. To achieve scalability, the basement can be distributed over multiple hosts (servers). In particular, for worlds with a large number of agents which may migrate frequently, the agent location database which is part of the basement may be partitioned over multiple separate servers based on a hash over the agent's GAID, possibly combined with a primary backup scheme to provide reliability.

In Mansion, logical and physical migration are coupled into one atomic operation *follow_hyperlink*, which is invoked by an agent on the Mansion API. A hyperlink is a logical link, which uniquely identifies a target room. If necessary, part of following a hyperlink is that the middleware transports the agent to a physical location from which the room is available.

Mansion - by design - only supports weak migration. An agent's execution state is not retained during migration. After physical migration, each agent is restarted from its initial state. This decision is in part taken because of supporting heterogenous agents (agents written in multiple programming languages); only weak migration can be supported by all programming languages that we want to support, which includes binary agents whose stack and data cannot be transparently transported from machine to machine.

In order to support migration of an agent, Mansion provides a data structure for transporting the agent and its data. This data structure is managed by the middleware and called an *Agent Container (AC)*. The AC contains a number of (typed) segments, which are used to contain the agent's code, data, authentication information and other information needed by the agent on its itinerary. The AC can be used to transport data back to the agent's owner, or to move objects from one room to another, for example. An agent needs to serialize all information that it needs for its own execution, and store all data that it or its owner may need at a later time, in its AC.

Currently, each hyperlink is internally associated with a set of IP-addresses from which the target room is available. If the agent does not already reside on one of those machines, the agent has to be physically migrated to one of the machines associated with the hyperlink.

A *zone* is used to indicate the physical boundary for distributing a room. A room is only accessible from one of the hosts in the room's zone. An agent that wants to access a room has to migrate to a host in that room's zone.

The room owner trusts the zone in which he / she deploys his room (and the objects therein); typically, the zone owner is also the owner of the rooms in the zone.

As an example, a zone may consist of a set of trusted hosts located in a protected network segment within an organization (e.g., a staff-network segment at a university). In another case, a zone may consist of a number of hosts which are placed in 'server hotels' located all over the world, which are deployed by an organization to host rooms.

2.4 Zone Administration

A world deployer determines which and how many zones may be deployed in its world. For example, certain worlds may consist only of one zone owned by the world administrator, while other worlds may contain zones (and rooms) owned by a number of organizations.

Zones are protected by public-key cryptography. Each zone has a unique public/private key-pair, which is registered with the world. The public keys of registered zones in a world are made available through the basement (possibly signed by the world owner).

Which hosts are part of a particular zone is managed in a decentralized way. Each host in a zone has a *zone certificate,* using which it is capable of proving that it is part of a zone. A zone certificate consists of a host's public key, signed using the zone's private key. Using its zone certificate, a host can prove that it is part of a zone. A host's certificate may expire or be blacklisted: a zone manager may revoke host certificates, or issue them for a limited time only (i.e., as a lease).

Using zone certificates has administrative scalability advantages: zones can be managed decentralized. In addition to this, there are security advantages compared to using a shared private key for all zone members (i.e., it is possible to pinpoint the host on which a security violation took place within a zone)

3. SECURITY ASPECTS

The previous sections discussed the logical and physical model of Mansion. The following sections discuss the security aspects that need to be considered for applications using Mansion: protection of an agent against malicious hosts in the system, protection of hosts against agents, and protection of objects and information in the system.

3.1 Agent Protection

In Mansion, we do not assume general availability of mechanisms which protect an agent from the host on which it executes. Although solutions exist which protect an agent against the host on which it resides (at least for a limited period of time [7]), it is not yet clear whether these solutions can be applied in a multiple-language, heterogenous system. For example, solutions based on language-dependent mechanisms or secure hardware cannot be expected to be immediately applicable in a heterogenous, large-scale system.

Zones are convenient to express and analyze distribution and security properties of the hosts on which rooms are deployed. The zones in a world and their properties (e.g., owner) can be listed in a central service, for example in the attic.

The owners of agents can use the central zone-infor mation service to gather information about the zones in a world, and determine which zones they trust enough for their agent to migrate to.

Certain worlds may require a certificate containing a set of predefined properties to be published by each zone-deployer. These properties can then be stored in a database which can be queried by users (agent-owners) of that world. For example, this can be used to find all zones owned by company *xyz,* or all zones that guarantee to protect your privacy.

In our model, an agent's owner can make sure that its agent does not migrate to a host in a zone which is not trusted. To do this, the agent is equipped with a list of trusted zones to which the agent may migrate. The host on which the agent executes then is assumed not to send an agent to a host in an untrusted zone. This is not fail-proof, but Mansion provides a mechanism to detect which host violated the agent owner's trust by sending the agent to an untrusted host, and for detecting this violation as soon as the agent is migrated to another host (see section 3.1.1).

As an aside, note that it does not make sense for an agent to specify its own list of allowed zones as an argument when it follows a hyperlink. This would offer no security, since no-one except the agent's current host can, in retrospect, verify what the list of allowed zones was, in case the agent's host sent the agent to an untrusted host after all.

If an agent needs to inspect data or a room that is located on an untrusted machine, it has to spawn off a 'helper agent,' which is equipped to take orders from its parent, or has minimal functionality and takes minimal (sensitive) information with it to the untrusted host.

3.1.1 Agent Container Security

An agent should be able to pick up data and / or objects on its way. Objects in Mansion can be transported from room to room in some worlds. Details of this mechanism are outside the scope of this paper. imply put, however, this entails storage of a reference to this object, or storage of serialized state of the object, in the AC.

Data storage is a more general requirement. ince we only support weak migration, an agent does not retain data stored in its address space if it migrates to another host. Therefore, it has to store any data it may need later, or which it thinks is useful for its owner, into its AC.

The AC is also used to store the agent's code and possibly initial data (dependent on the agent's programming language), static information about the agent, such as its Global AgentID, authentication information, owner, and so on.

Agent Container Design

A Mansion AC consists of a number of (typed) segments and a table of content (TOC). For each segment, an entry exists in the AC's TOC, which maps the segment's name to its internal segment-name and other metadata about the segment. The segments are typed to indicate the information (e.g., the agent's code or data saved by the agent) stored

in the segment. Furthermore, each TOC-entry contains a checksum (e.g., a secure hash) of the entry's content. Using the checksum, the segments integrity can be verified.

When a new segment is created, as part of the per-segment TOC-entry a bit may be set which indicates whether this segment is persistent (nonremovable), or whether it is transient (i.e., it may be removed some time later). For transient segments, it is possible to note in the segment's entry where (in what zone or on what host) the segment may be removed.

For integrity verification throughout the agent's itin erary, we use a mechanism in some ways similar to work by Karnik and Tripathi [8]. Each time an agent migrates to another host, the agent's middleware signs the agent's TOC (reflecting the AC's current content) with the private host key. By retaining old TOCs as part of the AC (before the new one is signed), a complete audit trail can be established of all the changes that are made to the AC during its itinerary. The world entrance daemon's (trusted) host signs the first TOC of the agent's AC (the initial signature is returned to the agent's owner).

It is important that the private host key of each host on the agent's itinerary is used to sign the TOC, since this makes it possible to pinpoint the exact location where a possible security breach took place; for convenience, the zone certificate of the signing host may be added to the AC before it is signed.

One problem with this solution, is that it is possible to roll back the state of the AC to an earlier state, by removing all changes that were made to the AC after visiting a particular host, and by reinstating the TOC to the one signed by that host.

Including forward references in the AC to the next host that has to sign the TOC (i.e., the host that the agent is going to be sent to), makes it harder to revert back to an arbitrary previous state of the AC, however, it is still possible to remove segments if cycles are present in the agent's itinerary.

As a simple solution to avoid rollback, each TOC can be sent to a trusted 'auditor process' which is part of the world. This way, the audit trail is stored at a (trusted) location external to the AC, where it can be collected by the agent's owner. It can also be timestamped upon arrival at the auditor to make it possible for the owner to track agents roughly in real time.

The audit trail mechanism makes it possible to verify whether segments were illegitimately removed on a particular host (typically, this can already be verified by the next host on the agent's itinerary).

Agent Transfer Protocol

A handoff protocol between hosts is used as part of the agent transfer protocol (ATP) which requires that each host verifies the content of the AC as it comes in. The TOC of the outgoing AC is already signed: in the protocol discussed above, each host on the agent's itinerary signs the TOC as the agent is migrated to the next host. An additional requirement on the ATP is that the target host verifies the integrity of the incoming agent's AC, and returns the TOC back to the source-host signed with its own key (see fig. 1), as proof that the AC arrived consistent with the AC's TOC as signed by the source host.

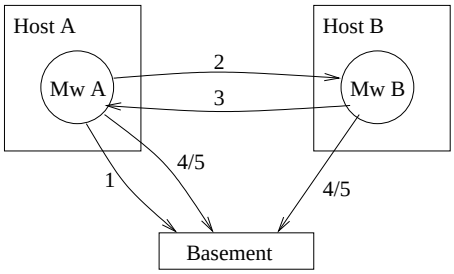

Figure 1: The agent handoff protocol. 1) Middleware A connects to the basement and initiates agent migration using the call *init_migr(GAID,[target host])*. Next it connects with Middleware B on host B, and tranfers the agent's AC to B (2). Middleware B verifies the AC's TOC as signed by middleware A, and possibly evaluates whether any changes (visible by means of the audit trail) made to the AC on host A were actually allowed (see sec. 3.1). If middleware B accepts the agent, it sends back the AC's TOC signed with its own key, and it commits the migration on the basement (3 + 4). Middleware A verifies middleware B's signature, and commits the migration on the basement (4). Both Middleware A and B can abort the migration transaction at any time during the protocol (5). Middleware A can store the signed TOC obtained from B, or it can send it to an auditor (sec. 3.1). The basement authenticates both middleware A and middleware B during the protocol – only the agent's current host may initiate an agent's migration, while the target host is indicated by the source host using the *init_migr* call.

In Mansion, physical migration is completed by registering the agent's new physical location in the basement. Before this happens, both the sending host and the receiving host have to agree to agent migration, and both have to indicate this agreement to the basement. Essentially, agent migration is an (atomic) transaction involving both the sender and the receiver (host) of the agent. The sending host will only commit a migration transaction if it has obtained the signed TOC from the target-host; the target host will only commit the migration (and return this TOC signed with its own secret key) if it has verified that the content of the AC corresponds to the AC's signed TOC. Note that the basement does not need to know anything about the handoff protocol, signed TOCs or any other security measure taken as part of the ATP.

Verifying incoming TOCs is important so that the target host cannot claim at a later time that the source-host omitted certain segments; conversely, the source-host cannot omit segments and claim that the target host removed them. Enforcing that each host on the agent's itinerary verifies the AC's integrity avoids the situation that a valid audit-trail is established, but that the segments are lost nevertheless

along the way, without being able to prove where those segments were lost exactly. This is an improvement over the system proposed in [8]. The 'incoming-TOC' which was signed by the target host can be stored by the source-host, or it can be forwarded to a (trusted) auditor process in the world.

An essential and not often realized advantage of audit trails, is that an audit trail views a host on which an agent executes as a black box. An audit trail shows which segments have been added and which segments have been removed on each host that the agent visited.

ince in Mansion the signed TOCs can be stored as readable (unencrypted) segments in the AC, this audit trail can be used to verify whether any changes were made to the AC that were not allowed on the previous host, or possibly even further back.

For example, for each transient segment, there may be an indication of the host or zone that may remove this segment in the segment's TOC-entry. If the segment is removed by a different host than the indicated one, this will be detected. If an agent is sent to a host that is not part of one of the zones of the trusted zones list that was provided as part of the agent, this can be observed from the audit trail too.

Audit-trail Based Security

As an example of a security measure that can be based on an audit trail, an agent's owner may equip an agent with a policy describing the changes that may take take place to an agent's AC at any host on its itinerary. This 'AC-change' policy can be used to avoid that a supposedly trusted host can do too much or any damage to an agent, for example by stealing (removing) segments from its AC.

An AC-change policy may specify how many (transient) segments (for example containing e-cash) may be removed from the agent's AC per zone. Verification of policies regarding (changes to) the AC may take place as part of the TOC-verification step at the next host on the agent's itinerary, Alternatively, verification may be done by an independent 'notary' process to which the agent's signed TOC is sent after each physical migration. uch a notary process can, for example, make the validity of a particular e-commerce (e.g., payment) transaction dependent on adherence to the agent's AC-change policy (see related work). Another example is a policy that specifies that segments may only be added to the AC.

The secure audit trail makes it possible to pinpoint illegitimate changes made to an AC to the host where these changes took place. This detection can take place as soon as the next host on the agent's physical itinerary; this host can not only refuse the agent if it finds a discrepancy between the TOC and the AC, but also if it detects an illegitimate removal of a transient segment (e.g., at the wrong host) or an 'AC-change' policy violation.

Providing an agent with a list of trusted zones (security domains) makes it possible to limit the chances of malicious attacks against an agent. Note that it is not necessary to predefine the full physical itinerary of the agent. This is important, since in Mansion the physical itinerary of an agent depends on the hyperlink layout of the world, in combination with the agent's interests. Which zones will be transfered as the agent roams a world will (in most cases) not be known before the agent sets out.

3.1.2 *Secrecy of Data*

To provide secrecy of data in an AC, public key encryption can be used. An agent or the agent's middleware can encrypt data (segments) intended for its owner using the owner's public key, which can be provided as part of the agent's AC. It is also possible to encrypt data (segments) intended for usage in a particular zone or on a particular host, using that zone or host's public key.

To provide secrecy of data between two communicating agents (or other entities), link encryption can be used to hide the information sent over the wire between two hosts (see section 3.4).

3.2 Protection of hosts

Protecting hosts in Mansion has two important aspects. The first is that a host should start up (untrusted) agents in a sandbox which makes it possible for the execution environment to control an agent's actions and resource-usage, and to protect the host from a malicious agent. An example sandbox is the well-known Java Virtual Machine, which can be used to protect a host from Applets downloaded from the Internet. Another example is the padded cell approach used for interpreting afe-Tcl applets [4].

The other aspect is trust. This aspect is particularly important because of the support for heterogenous agents in Mansion. ome applications may support agent programming languages against which no easy or foolproof way exist to completely protect the machine. An example is of course a binary program. In this case, it is important that some verification company or the author of the code vouches for the agent's safety. In some cases, it may be that a host's owner knows the owner of an agent personally, and therefore trusts the agent. Authentication of the principal owning an agent and possibly other principals related to the agent are important to establish trust in an agent, which may determine whether a host will execute the agent or not.

3.2.1 *Agent Authentication*

Agent authentication in Mansion is based on the Agent Passport (AP, see also [9]) concept. An AP is composed of a set of signatures of the agent's code[1]. In particular, the agent's owner signs the Agent Passport; this signature declares that the agent is sent in the system on behalf of this owner, i.e., a middleware that receives the agent can find out what principal owns the agent. The AP is stored in a segment of the agent's AC.

Another principal that can sign the agent's code is the author of that code. This may allow receiving hosts to attach trust in an agent's code (e.g., because the agent was written by un Microsystems). In addition, a (trusted) code verification company may sign the agent's code (this may be

[1]This can be executable or interpretable code, or an URL from which the code can be obtained, for example.

required for languages that one cannot sandbox easily, such as binaries).

In how far an agent's author or a code verification principal should sign the code, depends on the application and the agent programming language in question (e.g., a Java applet or afe Tcl program may be sandboxed enough to avoid malicious agent behaviour, so it is more easily trusted). The AP contains as much information as necessary to convince a host that the agent packed in the AC is trustworthy.

3.2.2 Secure Agent Execution

In Mansion, each agent is started up as a separate process. The only interactions that an agent may make with the (outside) world is by using the Mansion API. The Mansion middleware can act as a reference-monitor with regard to all invocations made by an agent, and is it possible to enforce security policies (access control policies). The Mansion middleware (MMW) runs in a protected address space separate from the agent's address space. The middleware may even run on a different host.

A Mansion agent is started up by the middleware in a sandbox, which makes sure that a sandboxed agent can only interact with the Mansion middleware using an IPC channel (e.g., a socket connection) to the middleware. Invocations by an agent on the Mansion API (provided through the agent's runtime system) are sent as marshaled invocations to the middleware.

The agent's execution environment (sandbox) has to take care that the agent does not bypass the middleware's control mechanisms. The way in which sandboxing is implemented differs per programming language and operating system, and is subject to research. As an example, it is possible to extend an O kernel (e.g., Linux or FreeB D) with a system call which makes sure that all system calls made by a child process of the process that invoked the system call (i.e., the MMW) are sent to its parent (in marshaled format), rather than being executed by the O . This mechanism can be used to sandbox binary agents.

Typically, the supported languages in a world are decided upon by a world designer based on the application's requirements (e.g., a scientific application may require high performance, and assume trustworthy agent owners. This may lead to support for binary programs. Other worlds may only support Java or afeTcl agents). Whether an individual agent is accepted on a host (and whether its code is run) is decided by the host's Mansion middleware. This decision can be based on the language in which an agent has been developed, its size or any other aspect.

3.3 Protecting Information (Authorization)

When an agent migrates to a host in a zone, it is authenticated using its agent passport. Then it immediately has to access the RMO of the room to which it migrated. ubsequently, the agent may have to access many more objects which are located in the room.

All objects (including RMO) are located in the room's zone. As soon as an agent attempts to access an object (using a *bind* request, which results in the object's interface being loaded into the agent's address space), the agent will have to be authorized with regard to the object.

In Mansion, we use the Globe access control mechanism using roles [10] for access control to objects. A role-certificate is a bitmap, signed by the object's owner, which indicates the methods that may be invoked on an object by the client to which the role-certificate was issued.

Normally in Globe, the role-certificate binds an authorized user's public key to a role-bitmap. In Mansion, the middleware obtains a role-certificate using which it accesses the objects that an agent is bound to on the agent's behalf.

To obtain a role-certificate, the middleware that authenticated the agent contacts a central database in the zone, called the authorization server. The authorization server trusts the agent's middleware to provide proper authentication information about the agent's principal (owner), which is obtained from the agent's AP, and looks up what role is associated with this owner. The owner-to-role mapping is stored in the authorization server by the object's owner. The authorization server returns a role-certificate to the middleware reflecting the agent owner's role.

The middleware stores the role-certificates for each object that an agent is bound to in an internal table. Each time an agent does a method invocation on an object, the middleware provides the role-certificate belonging to this agent to the object. Role-certificates are only valid within a zone, possibly only for a limited time (this is a decision made by the object's owner).

In some cases, there may be a 'guest' role-bitmap for principals for whom no specific access rights were set by an object's owner.

3.4 Link Encryption and Middleware Authentication

The basement stores each agent's current location in a world. For example, for each agent, the IP-address and port number to which a connection request can be made may be stored in the basement.

Besides this information, the basement can contain the zone certificate of the host on which the agent currently resides. Using this information, any middleware process in a world can verify whether it is communicating with the right host, for example when a request for communication is made. There are mechanisms to keep track of the agent's current location in a secure and verifyable way (using the handoff protocol, explained in sec. 3.1).

Agent authentication in Mansion happens transitively and contains a trust component. This applies only to inter-zone authentication of entities. Within a zone (i.e., intra-zone), all hosts and middlewares trust each other equally, so authentication of an agent taking place by one MMW in a zone is assumed to be correct and trusted by all other MMW processes in that zone (e.g., the authorization server).

An agent executes as a process on a host, and is under control of that host: all data in transport can be intercepted by

the agent's current host, in general. Also, an agent's execution can be tampered with by the host on which it executes. Therefore, one can never be sure that one is not talking with an impersonating process rather than the agent one intended to talk to. In short, one needs to trust the host on which an agent executes not to impersonate the agent; in general, one depends on the host on which the agent executes to authenticate an agent properly.

In Mansion, end-to-end (middleware-to-middleware) authentication will be used to set up authenticated, encrypted communication channels (e.g., using L) between middleware processes, and for communication with the basement.

4. RELATED WORK

In the literature a number of Multi-agent systems are described that support mobile agents [11, 12, 2]. ecurity of Mobile agents is addressed in most multi-agent systems [13, 4, 12], although often only briefly. Most MA es are Java-based and build on protection mechanisms offered by Java.

There are some approaches that address the problem of protecting agents against the host on which they execute. Code-obfuscation (cloaking) or time-limited blackbox [7] techniques can be used to obtain secrecy of data or computation inside an agent, at least for a limited period of time. Another approach is protecting agents based on cryptographically hiding polynomials or rational functions in an agent [14]. As discussed in section 3.1, we do not currently use such mechanisms as part of the Mansion security architecture, although Mansion could easily support agents that embed their own internal security mechanisms or detection mechanisms (e.g., cryptographic tracing [15]) in addition to the mechanisms provided by the Mansion middleware.

Ajanta [8] is a java-based mobile agent system. Ajanta is the first MA that provides a tamper-detection mechanism as part of its append-only data container. Ajanta's audit trail (based on signing added objects) is only visible to the agent's owner because this information is encrypted using the agent's public key; the audit trail cannot be inspected by other principals (hosts) on the agent's itinerary.

In contrast to Mansion, Ajanta's ATP requires only the sending host to update the location service. This makes it hard for a host to defend itself against certain security attacks that can be mounted by a malicious source-host. For example, a host can send an AC of which the segments are tampered with, even though the TOC still reflects the original, untampered state. It is hard for the target host to prove that it did not change the segments itself. The handoff protocol in Mansion makes it possible for the target host to refuse an agent based on verification of the incoming agent's AC.

Ajanta supports removable (transient) data in an unprotected container, but does not allow an agent or host to specify where (e.g., on which host) this data may be removed. Deviation from a (fixed) itinerary can only be detected in retrospect by the agent's owner.

An interesting technique that may be usable in Mansion, is the blinded-key signature proposal by Ferreira and Dahab [16]. This proposal is based on the idea of *blinding* secret keys. A blinded secret can be used to sign agreements, for example. A notary is a trusted third party which is responsible for verifying blinded-key signatures and enforcing agreements signed by agents using the blinded key. Enforcement of an agreement can only be done by a notary process which has access to the blinding factor using which the private key is blinded. uch enforcement of an agreement (e.g., a payment transaction) by a notary may be dependent on adherance to the AC-change policy set by the agent's owner, as explained in section 3.

D'Agents [13] is one of the few systems that supports heterogenous agents (currently Tcl, cheme and Java agents). D'Agents supports strong migration within trusted domains; transitive trust is used to authenticate the agent from host to host. Only if a host trusts the previous host on the agent's itinerary (and this one trusts the one before, etc.), does an agent remain 'owned' (authenticated). As soon as an agent migrates to a host that does not trust the previous host, the agent becomes anonymous (hence it has less access rights). After that, the agent can never become owned again, that is, the agent can no longer be authenticated transitively, so it remains anonymous.

In a large-scale system this approach's applicability can be questioned: agents whose execution state been changed will probably not be trusted by any machine after they have been changed on their itinerary.

Ara [9] distinguishes mutable and immutable parts of an agent's execution state, and has an agent passport concept. This is similar to our approach. Ara supports heterogenous agents using strong migration. Ara does not provide an AC concept. Instead, as many other systems, Ara relies on sending an agent from one host to another over a protected channel, but provides no mechanisms to protect an agent or its stored data from tampering by malicious hosts on the agent's itinerary.

5. DISCUSSION

Mansion provides a framework for designing MA es in a structured way. The logical model provides a clear framework for developing applications. This model is mapped on a set of hosts, using zones as an abstraction to group hosts that belong to a common (security) domain.

Mansion provides a middleware layer for multi-agent systems. This middleware provides the basic primitives for interaction with the world, such as inter-agent communication, binding to objects, and for logical (hyperlink) and physical migration. Mansion provides location and distribution transparency of logical entities in a world.

ome parts of Mansion's middleware functionality will be based on the Agent cape O middleware system [17], for which a prototype exists. Mansion middleware is currently being implemented as a second middleware layer on top of the Agent cape O . Agent cape O offers basic functionality such as inter-agent communication and migration primitives. The Globe middleware [6] is used to provide access to distributed objects. The security design of Mansion will be supported by Agent cape O .

In this paper, we presented a number of security mechanisms, which are designed to provide protection of agents, hosts and information in the system.

Agents can be sandboxed to protect the host. Agents are authenticated using their agent passport, which contains at least a signature of the agent's code by its owner, but possibly also of other principals like the author of the code. Authorization takes place using the authorization server in a zone, which assigns role-certificates for the objects in the room that an agent entered. Besides by using the Mansion API, an agent is not allowed to interact with the outside world. This way, it is possible to sandbox an agent in the Mansion environment, and enforce the logical rules and security policies set by the world.

An agent communicates directly with the Mansion middleware. The Mansion middleware manages such issues as inter-agent communication, binding to objects and migration to another room, possibly another host, mostly transparent from the agent. The middleware contains an Agent Container for each agent, which is the programming-language independent physical representation of an agent in Mansion. Agent protection is based on defining lists of trusted zones to which the agent may migrate, in addition to protection of the AC.

To our knowledge, there is currently no middleware design that allows for storing heterogenous agents and both persistent and transient segments in a secure and flexible way. Mansion's AC design is the first attempt to build a flexible container for transporting heterogenous agents as well as fixed and transient data from host to host in a secure way. The secure audit trail makes it possible to pinpoint illegitimate changes made to an AC to the host where these changes took place. Providing an agent with a list of trusted zones makes it possible to limit the chances of malicious attacks against an agent by those hosts, while not depending on fixed, completely predefined itineraries of trusted hosts.

Acknowledgements

We wish to acknowledge Benno Overeinder, Maarten van teen and Niek Wijngaards for their useful contributions to the model and this paper.

6. REFERENCES

[1] D. Chess; B. Grosof; C. Harrison; D. Levine; C. Parris; G. Tsudik. Itinerant Agents for Mobile Computing. *IEEE Personal Communications*, 4(5):34–49, October 1995.

[2] J. Baumann; F. Hohl; M. trasser; K. Rothermel. Mole - Concepts of a Mobile Agent ystem. *Technical Report, Universität Stuttgart*, August 1997.

[3] D.B. Lange and M. Oshima. *Programming and Deploying Java Mobile Agents with Aglets.* Addison-Wesley, 1998.

[4] J.K. Ousterhout; H.Y. Levy; B.B. Welch. The afe-Tcl ecurity Model. *Mobile Agents and Security*, 1998. LNC 1419, pringer-Verlag.

[5] E. Freeman; . Hupfer; K. Arnold. *JavaSpaces Principles, Patterns and Practice.* Addison-Wesley, 1999.

[6] M. van teen; P. Homburg; A. . Tanenbaum. Globe: A Wide-Area Distributed ystem. *IEEE Concurrency*, January-March 1999.

[7] Fritz Hohl. Time Limited Blackbox ecurity: Protecting Mobile Agents from Malicious Hosts. *Mobile Agents and Security*, pages 154–187, 1998. LNC 1419, pringer-Verlag.

[8] N. Karnik and A. Tripathi. ecurity in the Ajanta Mobile Agent ystem. *Software - Practice and Experience*, 2001.

[9] H. Peine. ecurity Concepts and Implementation for the Ara Mobile Agent ystem. *7th IEEE Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises*, June 1998.

[10] Bogdan C. Popescu; Maarten van teen; Andrew . Tanenbaum. A ecurity Architecture for Object-Based Distributed ystems. *Technical Report IR-492, Vrije Universiteit Amsterdam*, February 2002.

[11] H. . Nwana; D.T. Ndumu; L.C. Lee. ZEU , A Collaborative Agents Toolkit. *Proc. of the 3d Int'l Conference on Practical Applications of Agents and Multi-agent Technology, London, UK*, pages 377–392, March 1998.

[12] H.C. Wong and K. ycara. Adding ecurity and Trust to Multi-Agent ystems. *Proceedings of Autonomous Agents Workshop on Deception, Fraud and Trust in Agent Societies*, pages 149–161, May 1999.

[13] R. . Gray; D. Kotz; G. Cybenko; D. Rus. D'Agents: ecurity in a Multiple-language, Mobile-agent ystem. *Mobile Agents and Security*, pages 154–187, 1998. LNC 1419, pringer-Verlag.

[14] T. ander; C.F. Tschudin. Protecting Mobile Agents Against Malicious Hosts. *Mobile Agents and Security*, 1998. LNC 1419, pringer-Verlag.

[15] G. Vigna. Cryptographic Traces for Mobile Agents. *Mobile Agents and Security*, pages 137–153, 1998. LNC 1419, pringer-Verlag.

[16] Lucas. C. Ferreira and Ricardo Dahab. Blinded-key ignatures: ecuring Private Keys Embedded in Mobile Agents. *17th ACM Symposium on Applied Computing, Madrid, Spain*, pages 82–86, March 2002.

[17] N.J.E. Wijngaards; B.J. Overeinder; M. van teen; F.M.T. Brazier. upporting Internet- cale Multi-Agent ystems. *Data and Knowledge Engineering*, 2002.

Extending execution tracing for mobile code security

Hock Kim Tan *
Department of Electronics and Computer Science
University of Southampton
Southampton SO17 1BJ, UK
hkvt99r@ecs.soton.ac.uk

Luc Moreau
Department of Electronics and Computer Science
University of Southampton
Southampton SO17 1BJ, UK
L.Moreau@ecs.soton.ac.uk

Keywords
Mobile agent security protocols, mobile agent security framework, cryptographic tracing

ABSTRACT
The problem of protecting mobile code from both denial-of-service and state tampering attacks by malicious hosts are not well addressed in existing techniques for mobile code security. We propose a possible approach based on extending an existing mobile code security technique: cryptographic tracing. This is achieved through the introduction of a trusted third party, the verification server, which undertakes the verification of execution traces on behalf of the agent owner. The interaction between the verification servers and host platforms in the new protocol is outlined. Security properties of the protocol are verified by modelling the system in CSP and checking the resulting state transitions using the model checker FDR. Limitations of this approach to verification are then briefly discussed.

1. INTRODUCTION
Mobile code security can be broadly classified into two areas: host security and code security. Host security is concerned with protecting the host platform (i.e. the computational environment that supports the execution of the agent) from malicious agents. Such agents may attempt to gain unauthorized access to local resources on the host or else inflict damage on other agents or programs executing on the host. Code security deals with the exact reverse; it attempts to safeguard honest agents from potentially malicious host platforms. Attacks from these malicious hosts could take the form of extracting confidential information (such as cryptographic keys or credit card numbers) embedded within the agent. Many viable mechanisms have been developed to tackle the host security aspect, but code security still remains problematic. An overview of security issues in both these areas, along with a comparative discussion of the current techniques available to address them can be found in [10], [4].

In general, the most common types of attacks on mobile agents described in literature can be classified as involving either:

- manipulation / extraction / truncation of information accumulated in the agent from its previous hops, particularly in a free-roaming scenario (i.e. where the itinerary of the agent is dynamically determined during migration). Techniques such as forward integrity [11] and chained signatures [5] can provide some protection again this type of attack by making it possible to detect the point in the route at which the attack occurred.

- alteration of the state or execution flow of the agent. Techniques such as execution tracing [24] or obfuscated code [9] are designed to either detect an attack and identify the perpetrator or render such attacks impractical by increasing the difficulty of interpreting the semantics of code execution correctly.

More recently the issue of resource control has become a topic of interest in host security research [25], [18], particularly in the Java environment which is extremely popular for developing mobile agent systems. This raises the interesting question from the viewpoint of code security: how do we ensure that an agent is provided sufficient resources by its host in order for it to complete execution successfully ? In its most basic form, a denial-of-service attack would involve a malicious host platform simply terminating all mobile agents that migrate to it. A more subtle form of attack could involve withholding resources (memory, CPU cycles) for a protracted period of time so that an agent executes for a longer period than it normally would. This may be problematic in certain situations; for example, if the agent owner is later charged for the amount of resources allegedly consumed by the agent on that host.

A survey of the code security literature reveals very few techniques that address this problem directly. The techniques we have mentioned so far appear to be vulnerable to this type of attack. Approaches that could address this problem include the use of replicated agents [17]or co-operating agents [19]. In [17], replicated agents are executed on different hosts and simple voting is used to determine the outcome of computational results. This approach is extended on in [19], where other strategies such as secret sharing, remote authorization or remote storage of commitments can be used as part of protocols involving two co-operating (but not necessarily identical) agents that communicate with each other while migrating in different host platform domains. Some

*This research is funded in part by QinetiQ and EPSRC Magnitude project (reference GR/N35816).

of the criticisms regarding these approaches are that they require replication of services on all host platforms and may fail if the number of malicious hosts outnumber the honest ones (for the case of replicated agents). Co-operating agents appear more feasible but require that a specific co-operating agent and associated protocol be created for each application scenario, thus making it difficult to use for generic mobile agents.

In this paper, we provide three contributions:

- Describe an approach to detecting some forms of denial-of-service attacks that involves extending the execution tracing protocol, an existing code security technique.
- Formally model the extended protocol using CSP and FDR and establish specific security properties.
- Outline some general problems related to the use of finite state models in modelling mobile code security protocols.

In the next section, we discuss the original protocol and how it is extended. The detailed protocol of message exchanges involved in this extension version is outlined in section 3. Formal modelling and verification of the protocol using CSP and the model checker FDR is presented in section 4. Section 5 discusses the limitations of this modelling approach as well as some general problems that may arise when attempting to formally model mobile agent security protocols. Finally, section 6 concludes with a short summary and direction for future work.

2. EXTENDING EXECUTION TRACING

In execution tracing (Fig. 1), a host platform executing an agent creates a trace of an agent's execution that contains precisely the lines of code that were executed by the mobile agent as well as all the external values that were read by the mobile agent. The trace is then stored by the host. This tracing activity is repeated for all hosts in the path of the agent. Upon its return, the agent owner may (if she/he suspects that the mobile agent was not correctly executed) request the complete trace of the agent's execution commencing from the first host platform (a). The agent owner will then simulate the execution of the mobile agent based on the information contained in the trace. This simulation will result in an intermediate state and identify the next host platform in the mobile agent's itinerary. The agent owner requests from this platform its trace (b) and proceeds in this manner for all hosts in the agent's itinerary (c). If at some point a discrepancy is found during the verification of the trace provided by a particular host platform, then a malicious host has been detected.

There have been some criticisms of this approach. The main drawbacks are the size and number of logs related to traces that need to be retained by the hosts, and the fact that the detection process is triggered only on suspicion that an agent has been manipulated. Other problems include the difficulty of tracing the execution results of multi-threaded agents.

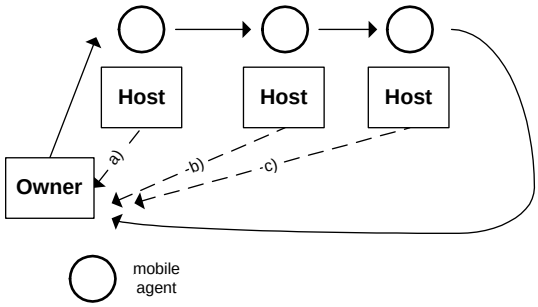

Figure 1: Execution tracing - original protocol

2.1 Introducing verification servers

In extending this protocol, we seek only to change the manner in which agents and traces are propagated in the system. The possible implementation of trace creation and verification using the approach outlined in the original protocol merits a complete analysis of its own and will not be discussed in this paper. Our approach is based on an earlier proposal [23] which involves the introduction of a trusted third party, the *verification server* that undertakes the process of verifying traces on behalf of the agent owner (Fig. 2). When an agent owner launches a mobile agent to a host platform (b), it creates a copy of the agent's code and state and forwards it onto a verification server (a) designated by the host platform. While the host executes the agent, it creates a trace of this execution simultaneously. Upon request of migration, the host then forwards this trace and the final agent state (c) to the designated verification server, which ensures that the execution sequence is valid. Once a verification server receives an agent copy, it will be aware of the identity of the platform executing the actual agent. It can thus implement a mechanism (for example, using time-outs) to ensure that a trace of the execution arrives from the required host within a reasonable time. This provides a way to safeguard against some forms of denial-of-service attacks.

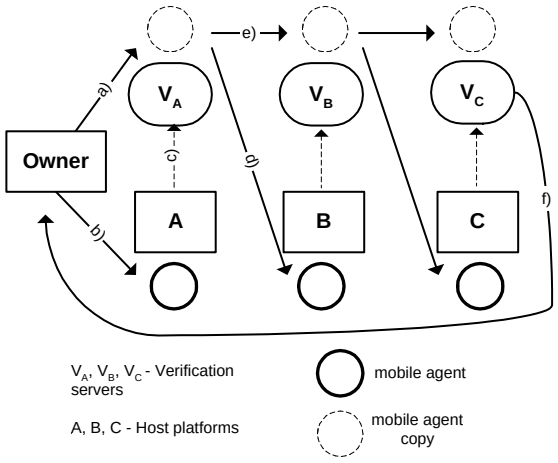

Figure 2: Execution tracing - extended protocol

When the validity of a trace is ascertained by a verification server, the agent is then forwarded from the verification server to its next destination host (d) and a copy (e) is sent to the corresponding verification server. Verification

and migration then proceed in this fashion until the agent completes its itinerary and returns to its owner (f). Host platforms do not need to retain traces once they are submitted to the verification server; verification servers only retain submitted traces which do not verify properly. These faulty traces can later be submitted as evidence to the agent owner or a suitable arbitrator for appropriate sanctions to be undertaken if so required. Only verification servers are allowed to migrate agents to host platforms; correspondingly, honest platforms will only accept agents from authenticated verification servers (more on this in section 4).

Prior to the commencement of a protocol run, host platforms will need to interact with verification servers to determine the servers that are willing (or capable) of verifying traces of agents executing in their environment. A host platform could thus have a choice of several verification servers to use in verifying any trace from its environment; conversely, a verification server could be responsible for verifying traces from several different hosts. Verification servers may delegate verification activities to other verification servers in the system if they are overloaded; this allows the formation of trust relationships between servers as detailed in [22].

2.2 Comparison with existing protocol

This extended protocol yields several advantages over the original one:

1. Trace verification is now performed by a verification server for each host platform that an agent migrates to in its itinerary. This permits the detection of malicious tampering as soon as it occurs at any platform on the agent's itinerary. In the original protocol, tracing only commences when an agent completes its tour and returns (by which time the damage inflicted by a malicious host could have been propagated to the remaining hosts in the itinerary) and even then, is only an optional activity triggered by a suspicious owner.

2. Traces have to be retained by a host in the original protocol (since the owner could request these for verification after a complete run of the agent), resulting in a high storage and maintenance overhead. This is no longer necessary in the extended version as verification is performed at every platform. In addition, verification servers can discard successfully verified traces and need only retain those with discrepancies as possible future evidence.

3. One of the primary motivations for using a mobile agent is to avoid communication problems attributable to low bandwidth or intermittent network links. In such an instance, the request of traces by an agent owner from potentially remote hosts in the original protocol could be problematic (for example, from behind a firewall). It would be easier instead for a host to select verification servers in its network vicinity that it can establish reliable communications with.

The extended protocol employs replication of agent code and state and is thus similar in motivation to the replicated agents approach. However, by imposing the trusted third party concept (i.e. the verification server is assumed to be a trustworthy entity that would not willingly collaborate with other hostile parties), we eliminate the need for replication of hosts as well as the possibility of failure that may arise when the number of malicious hosts outnumber honest ones in a voting scheme.

Our extended protocol shows greater similarity with the co-operating agents approach. The agent copy forwarded to verification servers for purposes of verifying the actual agent that migrates along a separate itinerary of host platforms can be regarded as a 'co-operating' agent that helps to detect tampering of the actual agent. However, the extended protocol offers the additional advantage of fault tolerance. A co-operating agent is not a replica of the actual agent to be protected, rather it is an agent that is specifically designed to support the actual agent in specific scenarios via constant communication so that it is immediately aware whenever the actual agent is compromised. In such an event however, it can only note or report the compromise but is incapable of continuing the compromised agent's agenda on its own. In the extended protocol however, if a verification server detects tampering in a trace, it can signal an exception to the agent copy executing in its environment so that suitable action can be taken.

3. PROTOCOL DESCRIPTION

In this section, we detail the protocol used in the extended version of execution tracing for one stage of a single protocol run (Fig. 3). A single protocol run is defined as the complete traversal of an unique agent instance along its itinerary, starting from where it departs from its owner (Fig. 2 b) to the point when it returns again (Fig. 2 e). This may include any possible loops in its path (i.e. when an agent returns to a previously visited host). We assume that a PKI is operating in the background, from which appropriate certificates and corresponding public keys can be obtained to perform encryption of data or verification of digital signatures. The pseudocode for the verification server and host platform is given in the Appendix. The format and sequence of messages exchanged in the protocol are shown in Fig. 4 and are explained as follows:

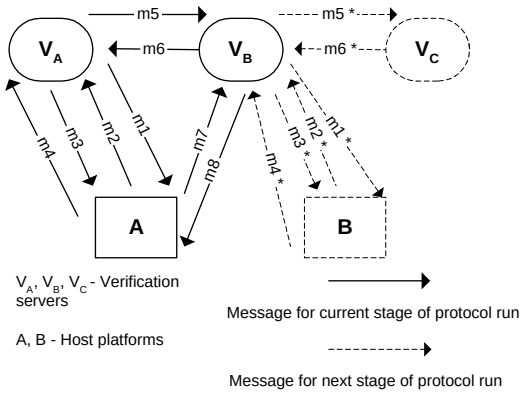

Figure 3: Message sequence in extended protocol

m_1: This message sent by V_A is in reaction to the mobile agent's request to migrate to A. It is essentially a request

m_1 : $\{\{V_A, A, n_{V_A}, Ic, \{c\}_{SPr(Owner)}\}_{SPr(V_A)}\}_{Pb(A)}$

m_2 : $\{\{A, V_A, t_{A_1}, n_{V_A}, Ic, Acc, V_B, \{A, V_B\}_{SPr(V_B)}\}_{SPr(A)}\}_{Pb(V_A)}$

m_3 : $\{\{V_A, A, t_{A_1}, n_{V_A}, Ic, S(c, V_A, T(c, V_A))\}_{SPr(V_A)}\}_{Pb(A)}$

m_4 : $\{\{A, V_A, t_{A_2}, n_{V_A}, Ic, S(c, V_A, T(c, V_A))\}_{SPr(A)}\}_{Pb(V_A)}$

m_5 : $\{\{V_A, V_B, \{c\}_{SPr(Owner)}\}_{SPr(V_A)}, m_4\}_{Pb(V_B)}$

m_6 : $\{V_B, V_A, n_{V_A}, Ic\}_{Pb(V_A)}$

m_7 : $\{\{A, V_B, Ic, n_{V_A}, T(c, A), S(c, A, T(c, A))\}_{SPr(A)}\}_{Pb(V_B)}$

m_8 : $\{V_B, A, n_{V_A}, Ic\}_{Pb(A)}$

Notation

- $\{X\}_{SPr(Y)}$ - indicates a message sequence X signed with the private key of entity Y
- $\{X\}_{Pb(Y)}$ - indicates a message sequence X encrypted with the public key of entity Y
- t_Y - indicates a timestamp created at entity Y
- n_Y - indicates a nonce created at entity Y
- Ic - refers to a unique agent identifier
- c - refers to static mobile agent code
- $T(c, Y)$ - refers to the trace of the agent code c after execution at entity Y
- $S(c, Y, T(c, X))$ - refers to the state of the agent code c after execution at entity Y, using the trace of the agent execution specified by $T(c, X)$

Figure 4: Protocol messages

to A to accept a mobile agent code instance Ic with associated code c. A nonce n_{V_A} is included here to keep track of a protocol run, and will be subsequently included in the following messages as a reference to that protocol run.

m_2: Upon receipt of m_1, A can decide whether or not to accept the mobile agent, based on consideration of the agent code. This may involve performing security checks on the code itself (i.e. host security) using techniques such as bytecode verification or proof carrying code. Acc in this message is therefore an indication of A's willingness (or otherwise) to accept the mobile agent. The last part of this message indicates the verification server (in this case V_B) that will verify the execution of any dispatched agent to A. This is accompanied by a certification signed by the verification server in question ($\{A, V_B\}_{SPr(V_B)}$). Message m_2 must be dispatched regardless of A's willingness to accept the mobile agent; this is necessary for V_A to distinguish between communication/server failure or agent rejection. A time-stamp t_{A_1} is included so that a record can be kept of m_2 in the event of a rejection.

m_3: An affirmative decision (with $Acc = Accept$) results in the state of the agent prior to migration being sent to A. In the event of rejection of platform A, the protocol run will terminate at this stage (with an appropriate exception flagged to the mobile agent), and recommence again at m_1 if so required by the mobile agent.

m_4: An acknowledgment message from A of the receipt of the agent. This message is vital to provide non-repudiation in the event that A attempts a denial of service attack once it has received the agent. The time-stamp t_{A_2} provides a reference value to implement a time-out mechanism in V_B to safeguard against denial of service.

m_5: A copy of the agent code and the entire contents of m_4 (which includes the agent's state) along with the appended signature created by $SPr(A)$ is then dispatched to V_B, the server that will be responsible for verifying correct execution on A. The identity of this verification server is obtained from the second portion of message m_2. Upon receipt of this message, a time-out mechanism will be in effect at V_B using t_{A_2} to ensure that m_6 arrives within a reasonable period of time, the failure of which is an indication of either a denial of service attack or a possible failure at A.

m_6: A simple acknowledgement of receipt of m_5 by V_B with inclusion of n_{V_A} and Ic to keep track of the current protocol run and agent instance.

m_7: Upon completion of agent execution, a trace is created at A ($T(c, A)$) and then submitted along with the new agent state $S(c, A, T(c, A))$ to the appropriate verification server.

m_8: A simple acknowledgement of receipt of m_7 by V_B with inclusion of n_{V_A} and Ic to keep track of the current protocol run and agent instance.

Upon receipt of m_7, V_B will commence replay of the agent c (identified by Ic) using the submitted trace $T(c, A)$. If the resulting state from this replay $S(c, V_B, T(c, A))$ is equivalent to the submitted state $S(c, A, T(c, A))$, then the next stage of the protocol run can be initiated, that is V_B can dispatch m_1* (the equivalent of m_1 in the next stage of the protocol) to B. The submitted trace can then be discarded. If equivalence is not obtained, an appropriate exception is raised to the mobile agent and the faulty trace and state is retained as evidence for further action by the home platform or an arbitrator (if so required).

4. FORMALLY MODELLING AND VERIFYING THE PROTOCOL

The primary security goal of execution tracing is to safeguard the state and execution flow of an agent, which is accomplished in the original protocol and in our extended version, by verifying agent states produced by replaying agents according to a given trace. If we assume that the verification and replay process is capable of detecting any malicious tampering, then the security goal essentially reduces to ensuring that traces and agents are dispatched correctly and securely to their designated destinations as outlined in the protocol. The original protocol uses various cryptographic primitives in order to achieve this goal (in a similar fashion to us) but does not attempt to formally verify the satisfaction of any security property. As it has been noted in literature that developing good security protocols is notoriously difficult [2], we believe that some form of modelling and verification is necessary in order to provide a basic assurance that certain specific security properties are achieveable. In the case of our extended protocol, we are primarily interested in two security properties that provide guarantee of correct and secure dispatch of agents and traces:

- Mutual authentication of verification servers and host platforms - It is important to make a distinction be-

tween these two entities as host platforms should only accept mobile agents that are dispatched from verification servers. This ensures that honest host platforms will never accept agents with potentially corrupted states directly from other platforms. The possibility of a hostile host platform spawning multiple copies of an agent and dispatching it randomly to other platforms in the system is also circumvented. In a similar context, a verification server has to ensure that it receives a copy of an agent from an authentic verification server to ensure that it is verifying the correct agent instance for a particular protocol run.

- Non-repudiation of commitment to executing agents - It is important to retain evidence of the fact that a host platform has committed to executing a particular mobile agent instance in a given protocol run to prevent a denial of service attack (i.e. terminating a mobile agent or delaying its execution for an inordinate period of time). This is primarily achieved by a digital signature appended to m_4 which is retained by V_A. The trace of agent execution as encapsulated in m_7 is also retained by V_B in the event it turns out to be faulty; this can be later be submitted to a third party arbitrator or the agent owner for sanctions to be undertaken towards the erring platform if the need arises.

In considering the security properties of the protocol, it should be mentioned that the basic underlying assumption is that the verification server is treated as a trusted third party. Thus we assume that verification servers will not engage in any action that will directly or indirectly lead to the corruption of an agent's state. We also make the usual assumption that the basic cryptographic primitives used are resistant to standard cryptanalysis and that private keys are not compromised. There are many approaches available for formally modelling and verifying properties of a security protocol. Some of the more commonly utilised ones include:

1. BAN logic of authentication [3], which reasons about the states and beliefs of agents involved in a protocol run and how these beliefs evolve with the reception of new information
2. Spi-calculus [1], which is an extension of π-calculus designed to deal with cryptographic primitives
3. Strand-spaces approach [6] uses the concept of a strand to represent the sequence of actions in which a particular protocol principal may participate and then reasons about how the strands interact or intertwine as participants interact by the exchange of messages
4. CSP-based approach [12] models the protocol interactions as a system described by CSP process algebra [8], for which violation of given specifications can be detected through the use of a finite-state model checker such as FDR [16]

We have chosen the last approach as it has been used successfully in discovering attacks upon cryptographic protocols ([12], [13], [15]). In addition, modelling protocol runs as interactions between entities using a process algebra like CSP appears to be a reasonably intuitive one. As a complement to this approach, tools such as Casper [14] have been developed that are capable of converting a high-level description of a security protocol to a CSP specification of the model that can be fed as input into the FDR model checker for subsequent verification. This greatly simplifies the process of CSP specification, which can be tedious and error-prone for complicated protocols. We employ Casper in describing our protocol in this section, further details on this protocol specification language can be found at [14].

4.1 Modelling the protocol in Casper

For the free variables section, we have declared the following variable types:

```
#Free variables

A : Agent
VA, VB : Server
nva, nt1, nt2 : Nonce
ca : AgentCode
asva, asa       : AgentState
ata, atvb : AgentTrace
PK : Agent -> PublicKey
SK : Agent -> SecretKey
SSK : Server -> ServerSecretKey
SPK : Server -> ServerPublicKey
hash : HashFunction
InverseKeys = (PK,SK), (SSK, SPK)
```

The agent code c, trace $T(c, Y)$ and state of the agent $S(c, Y, T(c, X))$ (see Fig. 4) are represented by the types `AgentCode`, `AgentState` and `AgentTrace` respectively and can assume different values independently of each other. This makes the protocol easier to model as it is difficult to define multi-variable functions correctly using Casper (i.e. the state of an executed agent would be a one way function of its code, initial state and trace). A side effect of this is that the chances for attack by a malicious host is increased since it can interleave different values of code, state and trace with impunity in a protocol run. Indirectly, this enhances the strength of the security property that we intend to establish. We distinguish between the public and private keys of host platforms and verification servers as we regard them as two distinct classes of entities in our protocol. A hash function is used to model the unique agent identifier, Ic, which we treat simply as a hash of the agent code (the actual value of Ic is explained at the conclusion of Section 6).

There are three processes representing V_A (SERVERINITIATOR), V_B (SERVERRESPONDER) and A (HOSTRESPONDER) respectively of Fig. 3. All three entities will have knowledge of their respective secret keys and will be able to access the public keys of the other entities.

```
#Processes

SERVERRESPONDER(VB) knows SSK(VB), SPK, PK
SERVERINITIATOR(VA, nva, ca, asva) knows SSK(VA), SPK, PK
HOSTRESPONDER(A, nt1, nt2, asa, ata) knows SK(A), SPK, PK
```

The protocol is modelled below, where lines 1 – 8 correspond to m_1 – m_8 of the protocol. We use nt1 and nt2 (of type

Nonce) to represent the time-stamps t_{A_1} and t_{A_2} issued by A, as we are only interested in their unique values in the protocol run and do not employ them to enforce a notion of freshness. The protocol run is preceded with step 0c., which establishes the result of an earlier interaction where A obtains a certification from a verification server V_B certifying V_B's capability of verifying agent traces from A. The % notation is used to indicate that this certification is not processed directly by A, rather stored in a temporary variable and then later relayed to V_A in message 2. The same comments apply as well to enc in message 4 and 5.

```
#Protocol description

0.    -> VA : A
0a.   -> A : VA, VB
0b.  -> VB : A
0c.  VB -> A : {{A, VB}{SSK(VB)} % storecert}{PK(A)}
1.  VA -> A : {{VA, A, nva, ca, hash(ca)}{SSK(VA)}}{PK(A)}
2.  A -> VA : {{A, VA, nva, nt1, hash(ca), VB,
               storecert % {A, VB}{SSK(VB)}}{SK(A)}}{SPK(VA)}
3.  VA -> A : {{VA, A, nt1, hash(ca), asva}{SSK(VA)}}{PK(A)}
4. A -> VA : {{A, VA, nt2, nva, hash(ca), asva}{SK(A)},
              {A, VA, nt2, nva, hash(ca), asva}
              {SK(A)} % enc}{SPK(VA)}
5. VA -> VB : {{VA, VB, ca}{SSK(VA)},
               enc % {A, VA, nt2, nva, hash(ca), asva}
               {SK(A)}}{SPK(VB)}
6. VB -> VA : {VB, VA, nva, nt2}{SPK(VA)}
7. A -> VB : {{A, VB, hash(ca), nva, ata, asa}{SK(A)}}{SPK(VB)}
8. VB -> A : {VB, A, nva, hash(ca)}{PK(A)}
```

We assume that the intruder is capable of creating its own agent trace, code and state. In addition, in line with the normal assumptions for an intruder in Casper, the intruder will also be capable of creating its own nonces and accessing the public keys and identities of all entities in the system.

```
#Intruder Information

Intruder = BadHost
IntruderKnowledge = {FirstServer, SecondServer, BadHost, Nb,
          Nbt1, Nbt2, Cb, PK, SPK, Asb, Atb, SK(BadHost)}
```

4.2 Specifying security properties

As mentioned earlier, the two important security properties to be established are mutual authentication and non-repudiation. We employ the concept of authentication as outlined in Casper . This is briefly expressed in the form of the statement **Agreement(A, B, [x])**, which states that A is authenticated to B on the basis of the fact that both A and B agree on the value of x. More formally [14], this means that if B (taking the role of responder) completes a protocol run, apparently with A, using the data value x, then the same entity A (taking the role of initiator) has previously been running the protocol, apparently with B, using the same value x. In addition, each such run of B corresponds to a unique run of A. x is typically some unique data item (such as nonce or time-stamp) known only to A or B. Mutual authentication will therefore require the additional statement **Agreement(B, A, [x])** to be verified as well.

For the case of V_A and A, we can claim that these two entities are properly authenticated to each other after the exchange of m_1 - m_4, if only these two entities agree on the values $Ic, t_{A_2}, n_{V_A}, S(c, V_A, T(c, V_A))$. Ic is necessary to provide reference to the unique agent instance, n_{V_A} provides reference to the current protocol run, t_{A_2} and $S(c, V_A, T(c, V_A))$ provides reference to A's response in m_4.

```
Agreement(VA, A, [nva, nt2, hash(ca), asva])
Agreement(A, VA, [nva, nt2, hash(ca), asva])
```

Similarly mutual authentication between V_A and V_B and between V_B and A can be expressed as

```
Agreement(VA,VB, [nva, nt2])
Agreement(VB,VA, [nva, nt2])
Agreement(VB, A, [nva, nt2])
Agreement(A, VB, [nva, nt2])
```

Non-repudiation can be simplified to the more general case of maintaining secrecy of specific data items whose non-repudiation is to be established. If we know that only entity A issues item x in a protocol exchange between itself and another entity B, and if we can establish that item x remains secret in such a protocol exchange, then we can conclude that A is indeed responsible for issuing x. In our protocol, we are not interested whether x can later be duplicated by another entity (such as B) in another protocol run, rather we are concerned with whether x is issued in a given protocol run. The property of non-repudiation then follows simply by applying a digital signature to x. As mentioned earlier, non-repudiation is necessary for messages:

1. m_2 - to provide evidence a host accepts or denies an agent for a particular protocol run indicated by n_{V_A} and an agent instance indicated by Ic;
2. m_4 - to provide evidence a host has received the state $S(c, V_A, T(c, V_A))$ necessary to begin execution of the agent instance Ic in the protocol run indicated by n_{V_A};
3. m_7 - to provide evidence the agent instance was Ic executed with a trace $T(c, A)$ to provide a state $S(c, A, T(c, A))$.

In Casper, the statement **Secret(A, x, [B])** is used to express the property that A believes x remains secret in an interaction between itself and an entity that appears to be B. If this entity is not B, then x will remain hidden to it. Thus, to show non-repudiation, we have the following security specifications [1]:

```
Secret(VA, nva, asva, ca, [A])
Secret(A, nt1, [VA])
Secret(A, ata, asa,  [VB])
Secret(VB, nt2, [A])
```

Both specifications for mutual authentication and secrecy were satisfied in the resulting CSP model that was checked using FDR. The checking process itself was lengthy (several hours) due to the complexity of the protocol and the number of independent data variables involved.

[1] These are provided in an abbreviated form; specifications for secrecy should be in the form Secret(VA, x, [A]) for each item x

5. LIMITATIONS OF MODELLING USING CASPER AND FDR

The use of Casper to model security protocols for mobile agent systems has been attempted previously by Hannotin et al. [7]. In their work, they attempt to verify the property of data integrity in a protocol proposed by Corradi et. al [5] which intends to safeguard data accumulated by a mobile agent (for example, price offers from various shop platforms) during its itinerary from invalid tampering. The protocol functions by making tampering of this data by a malicious host (for example:- modification or truncation of previous offers) detectable by either the home platform of the agent or the next honest host that the agent migrates to. Although this property is verified in the CSP model that they develop, the same protocol (as well as other protocols with a similar motivation of protecting accumulated data) was shown to be vulnerable to a certain type of attack described by Roth in [20], which has a general two-step approach:

1. Protocol data from an honest mobile agent is cut and pasted on to a 'dummy' mobile agent generated by a hostile host. This mobile agent is then launched to another honest host with which it interacts in a certain manner to acquire critical information about the current protocol run (which it would not normally be able to acquire without the use of the 'stolen' protocol data).

2. The 'dummy' agent migrates back to the hostile host with this information, which is then used by the host in some way to change the accumulated data of the honest agent. This change will subsequently be undetectable when the honest agent is migrated on to the next host or back to its home.

The strategy of this attack is not new and is similiar in motivation to an earlier well-known attack on the Needham-Schroeder public key protocol described by Lowe [13]. In his attack, a replayed message from a previous protocol run is used by an intruder to initiate a new protocol run in which an unsuspecting participant is then abused as an unwitting oracle to reveal confidential information. This information can then be used to compromise the integrity of a communication channel. Roth's technique is essentially the same with the primary difference being that a new mobile agent (instead of a replayed message) is used by a malicious host in a new protocol run to initiate the oracle attack. In order to nullify the attack, it is necessary to prevent one or both of these steps from occurring. Roth presents a method to prevent the first step by using authentication to uniquely associate the identity of an agent instance along with the protocol data transported by it. This would allow an honest host to discern whether an incoming agent is carrying protocol data that belongs to it or that was 'stolen' from another agent. The host could then refuse to accept or execute agents carrying 'stolen' data, thus preventing itself from being abused as an oracle. This security measure is in actual fact a form of host security, and we have here an interesting illustration of how the two different aspects of code and host security (which often appear to be orthogonal to each other) can be actually closely interlinked.

The main reason underlying the failure of Hannotin's Casper model to detect such attacks is the inability to model a mobile agent accurately using traditional cryptographic protocol analysis methods. In those methods, a fundamental assumption in analysis is that the format of messages exchanged between static entities and the sequence in which they occur within a single protocol run are predefined and remain fixed throughout the duration of the protocol run. Thus, attacks can only occur through judicious interleaving, reflection or replay of messages from different protocol runs. In Hannotin's approach (and our approach as well) the mobile agent is implicitly treated as a unique, static portion of a message. This permits a reasonably straightforward approach to modelling, but as we have just seen, it is not accurate as it does not reflect the ability of the mobile agent to potentially alter the sequence and content of messages during an ongoing protocol run. For example, in the second step of the attack, the data carried back by the 'dummy' agent to the hostile host has to be part of the specification of the protocol run (since this is the data that actually allows the hostile host to successfully carry out its attack). Obviously, the format and contents of this data cannot be predefined and will depend on the interaction of the agent with the honest host. Thus, in order for a model to be able to detect such attacks, two additional requirements are necessary:

1. The model must be able to encapsulate all possible behaviours of a mobile agent (as a function of its code, internal state and state of its execution environment) that have the ability to alter the format or sequence of messages exchanged within a single protocol run

2. The model must be able to take into account all these different possibilities of message contents and sequences when it is used to simulate a protocol run

With regards to the first requirement, the identification of the specific state or code of an agent that is capable of altering the format or sequence of messages is clearly not a trivial matter. Even if this could be accomplished, the additional possibilities for protocol runs with different message contents and sequences will greatly increase the number of possible interleaving of protocol runs, consequently creating a potential explosion in the state space to be explored. This may make it less suitable for use on a finite state space model checker such as FDR. In that case, checking the viability of the model may require techniques to reduce the state space explosion (such as those used in [21]) or modelling the protocol using a different approach (for example strand spaces or spi-calculus). As a matter of interest we note that the attacks described by Roth were discovered in an ad hoc, intuitive manner without resort to any formal methods of verification. It is thus possible that more subtle attacks may yet exist on the protocols in question (even after the remedy of authentication is applied), if these protocols can be expressed and analysed in more thorough manner using models that encapsulate the two requirements that we have briefly discussed.

Since we also treat the mobile agent as a static message in our approach, our model is equally susceptible to the same

vulnerabilities as Hannotin's. However, our modified protocol for execution tracing differs from the approach employed by Corradi as well the original execution tracing protocol in an important way: agent state (and code) is replicated. In our approach, the agent that is actually migrated on to the next host platform is the mobile agent copy on the verification server (the trusted platform), and not the actual agent on the current host. The consequence of this is that the current host will not be able to directly manipulate the state of the mobile agent, in effect nullifying step two of Roth's attack. The only way a hostile host can affect the state of the agent copy is through the trace it supplies; this however will also contain the signature of the host to act as a measure of non-repudiation. Supplying faulty traces as a form of attack is thus meaningless as liability can eventually be established for the resulting problems that arise (this, of course, is based on the assumption that the economic cost of being sanctioned for an attack is greater than the economic cost resulting from the attack). Therefore, our only concern is ensuring that traces, agent code and agent state are securely propagated in our system, and that traces are correctly associated with the corresponding agents. Mobile agent behaviour is subsequently of no concern to us any longer. We are therefore justified in using Casper to model our protocol as the nature of our protocol is now analogous to those modelled in standard cryptographic protocols. It is also our belief that completely secure code execution on untrusted platforms cannot be achieved without some form of code/state replication.

6. CONCLUSION

In this paper, we identified the need for code security techniques that address the concept of denial-of-service attacks in addition to the usual data integrity and state tampering attacks. A technique to detect some forms of such attacks is proposed which involves the extension of a well known code security technique, execution tracing. This essentially involves the introduction of a trusted third party, the verification server, that undertakes verification of traces on behalf of the agent owner. The advantages of this modified technique as compared to the original approach as well as other techniques that prevent denial-of-service attacks are outlined. The sequence of messages for the new protocol is described in detail, and is then modelled in CSP using the high-level security protocol description language, Casper. The model is then analysed in FDR to determine whether specific security specifications are valid. Finally, we discuss the limitations of modelling the protocol using Casper and finite state model checkers such as FDR and point out the difficulties involved in formal modelling of mobile agent security protocols in general.

Our current work focuses on developing a practical method to implement creation and verification of traces in a working mobile agent system. In addition, we are also looking at ways of reducing the cryptographic cost of the protocol without compromising on its security properties. A more formal method of expressing the use of time-outs to provide protection against denial-of-service attacks would also be useful. Once the protocol is sufficiently refined and trace verification properly developed, a mobile agent framework using the extended protocol can be created and evaluation conducted against existing code security techniques.

7. REFERENCES

[1] Martin Abadi and Andrew D. Gordon. A calculus for cryptographic protocols: The spi calculus. In *Fourth ACM Conference on Computer and Communications Security*, pages 36–47. ACM Press, 1997.

[2] Martin Abadi and Roger M. Needham. Prudent engineering practice for cryptographic protocols. *Software Engineering*, 22(1), 1996.

[3] Micheal Burrows, Martin Abadi, and Roger Needham. A logic of authentication. *Proceedings of the Royal Society*, 426(1871), 1989.

[4] D. M. Chess. Security issues in mobile code systems. In *Mobile Agents and Security*, number 1419 in LNCS. Springer-Verlag, 1998.

[5] A. Corradi, R. Montanari, and C. Stefanelli. Mobile agents integrity in e-commerce applications. In *Proceedings of the19th IEEE International Conference on Distributed Computing Systems Workshop, IEEE Computer Society Press, Austin*, May 1999.

[6] F. J. Thayer F'abrega, J. C. Herzog, and J. D. Guttman. Strand spaces : Why is a security protocol correct ? In *Proceedings of the 1998 IEEE Symposium on Security and Privacy*, pages 160–171, February 1998.

[7] Xavier Hannotin, Paolo Maggi, and Riccardo Sisto. Formal specification and verification of mobile agent data integrity properties : A case study. In *Mobile Agents : Proceedings of the 5th International Conference, Altanta, USA*, number 2240 in LNCS. Springer-Verlag, 2001.

[8] C. A. R. Hoare. *Communicating Sequential Processes.* Prentice-Hall International Series in Computer Science, 1985.

[9] Fritz Hohl. Time limited blackbox security: Protecting mobile agents from malicious hosts. In *Mobile Agents and Security*, number 1419 in LNCS. Springer-Verlag, 1998.

[10] Wayne Jansen. Countermeasures for mobile agent security. In *Computer Communications, Special Issue on Advances in Research and Application of Network Security*, November 2000.

[11] G. Karjoth and N. Asokan. Protecting the computation results of free-roaming agents. In *Mobile Agents and Security*, number 1419 in LNCS. Springer-Verlag, 1998.

[12] Gavin Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Proceedings of Tools and Algorithms for Construction and Analysis of Systems*, number 1055 in LNCS. Springer-Verlag, 1996.

[13] Gavin Lowe. Some new attacks on security protocols. In *9th IEEE Computer Security Foundations Workshop*, 1996.

[14] Gavin Lowe. Casper : A compiler for the analysis of security protocols. In *Proceedings of The 10th Computer Security Foundations Workshop*. IEEE Computer Society Press, 1997.

[15] Gavin Lowe and Bill Roscoe. Using CSP to detect errors in the TMN protocol. *IEEE Transactions on Software Engineering*, 23(10), 1997.

[16] F.S.E. Ltd. Failures-Divergence Refinement : FDR2 User Manual, Available at http://www.formal.demon.co.uk/fdr2manual/. Technical report, Formal Systems Europe, 1999.

[17] Yaron Minsky, Robbert van Renesse, Fred B. Schneider, and Scott D. Stoller. Cryptographic support for fault-tolerant distributed computing. In *Proceedings of the Seventh ACM SIGOPS European Workshop*, 1996.

[18] Luc Moreau and Christian Queinnec. Distributed computations driven by resource consumption. In *Proceedings of IEEE International Conference on Computer Languages (ICCL'98)*, 1998.

[19] Volker Roth. Mutual protection of co-operating agents. In *Secure Internet Programming: Security Issues for Mobile and Distributed Objects*, number 1603 in LNCS. Springer-Verlag, 1999.

[20] Volker Roth. On the robustness of some cryptographic protocols for mobile agent protection. In *Mobile Agents : Proceedings of the 5th International Conference, Altanta, USA*, number 2240 in LNCS. Springer-Verlag, 2001.

[21] D. Song, S. Berezin, and A. Perrig. Athena, a novel approach to efficient automatic security protocol analysis. *Journal of Computer Security*, 9(1), 2001.

[22] H. K. Tan and L. Moreau. Trust relationships in a mobile agent system. In *Proceedings of the 5th IEEE International Conference on Mobile Agents, Georgia, USA*, December 2001.

[23] H. K. Tan and L. Moreau. Certificates for mobile code security. In *Proceedings of the 17th ACM Symposium on Applied Computing*, March 2002.

[24] Giovanni Vigna. Cryptographic traces for mobile agents. In *Mobile Agents and Security*, number 1419 in LNCS. Springer-Verlag, 1998.

[25] Alex Villazon and Walter Binder. Portable resource reification in java-based mobile agent systems. In *Mobile Agents : Proceedings of the 5th International Conference, Altanta, USA*, number 2240 in LNCS. Springer-Verlag, 2001.

APPENDIX

Operation of host platform

Accept m_1
Verify signature $SPr(V_A)$ and identity of verification server (V_A) through a certificate
Verify signature on agent code ($\{c\}_{SPr(Owner)}$) to detect possible tampering
Perform security check on agent code ($\{c\}$)
If decision is made to commit to running agent code
  Select verification server V_B to be employed in verifying the code
  Submit in m_2 suitable certification $\{A, V_B\}_{SPr(V_B)}$
  If verification server responds with m_3
    Verify signature on $S(m, V_A, T(m, V_A))$ to safeguard against possible tampering
    Respond with acknowledgment m_4
    Instanstiate agent code with verified state
    Commence execution of mobile agent and create a trace of its execution sequence $T(m, A)$
    Upon completion of an agent run, sign trace and submit in m_7 to V_B
  else
    Terminate protocol run and await commencement from another verification server
else
  Indicate refusal in reply m_2
  Terminate protocol run and await commencement from another verification server

Operation of verification server V_B

Receive initial state of agent $S(c, V_A, T(c, V_A))$ in m_5
Implement time-out mechanism using t_{A_2} from m_4
If trace $T(c, A)$ in m_7 arrives in specified time period
  Verify identity of host submitting trace
  Replay agent execution from initial state and submitted trace to obtain final state $S(c, V_B, T(c, A))$
  If final state $S(c, V_B, T(c, A))$ is equivalent to submitted state $S(c, A, T(c, A))$
    Identify destination platform contained in $S(c, V_B, T(c, A))$
    Submit m_1* to destination
    Receive m_2*
    If Acc in m_2* is positive
      Verify identity of the verification server associated with submitted certification
      Submit final state of agent $S(c, V_B, T(c, A))$ to host platform in m_3*
      Receive acknowledgment of reception m_4* from the host platform
      Forward $S(c, V_B, T(c, A))$ and $\{c\}_{SPr_{Owner}}$ on to the next verification server in m_5*)
    else
      Signal exception to agent
      Record platform identity that refused to host the agent
  else
    Retain trace $T(c, A)$ as evidence
    Signal exception to agent
    Record occurrence of trace inequivalence in agent
else
  Signal exception to agent
  Record occurrence of time-out in agent

Securing Multi-Agent Platform Communication

Juan Jim Tan, Leonid Titkov, Constantinos Neophytou
Department of Electronic Engineering, Queen Mary, University of London,
Mile End Road, London E1 4NS
+44 (0) 20 7882 5542
Email: {juanjim.tan, leonid.titkov, c.neophytou}@elec.qmul.ac.uk

Abstract
Communication between multi-agent platforms has shown potential problems in aspects of security, for example confidentiality, authentication and integrity. To address some these issues, this document focuses on analysing and specifying inter-platform agent security in general. In this initial document, we investigate various core requirements and the S/MIME content-type for inter-platform messaging. This paper also presents a message confidentiality design using 'Enveloped Data' S/MIME content-type adhering PKCS 7 standards. In our design, we have also included CMS designs using ASN.1 notations to derive our message identifiers supporting platform-to-platform communication. We envisage the result of this specification to be applicable as most current HTTP messaging systems support MIME standards. Finally, further additional enhancements are discussed such as 'Enveloped and Signed Data' content-type and other key transport or key agreement methods; these will be covered in future publications.

Keywords
S/MIME (Secure/Multipurpose Internet Mail Extensions), MIME, ASN.1 (Abstract Syntax Notation), CMS (Cryptographic Message Syntax), PKCS (Public-Key Cryptography Standards), SSL, PEM, CBC, RMI, IIOP, HTTP (Hypertext Transfer Protocol), FIPA.

1. Introduction
The Foundation for Intelligent Physical Agents (FIPA) Agent Message Transport Service (MTS) Specification [1] can be improved by applying security over transport messages in support of the MTS [11]. Security issues within this communication protocol may pose a threat in developing large-scale e-business based solutions for example in Agentcities.RTD [2].

Agents mostly interact in two types of environments; inter platform and/or intra-platform communication [1]. In these environments agents may use various types of message transport protocols for instance HTTP, IIOP (Internet Inter-Orb Protocol) or RMI (Remote Method Invocation). In the MTS of an Agent Platform (AP), an agent has three options[1] when sending a message to another agent resident on a remote Agent Platform (AP) (see *Figure 1*):

1. Agent A sends the message to its local Agent Communication Channel (ACC) using a proprietary or standard interface. The ACC then takes care of sending the message to the correct remote ACC using a suitable Message Transport Protocol (MTP). The remote ACC that will eventually deliver the message.
2. Agent A sends the message directly to the ACC on the remote AP on which Agent B resides. This remote ACC then delivers the message to B. To use this method, Agent A must support access to one of the remote ACC's MTP interfaces.
3. Agent A sends the message directly to Agent B, by using a direct communication mechanism. The message transfer, addressing, buffering of messages and any error messages must be handled by the sending and receiving agents. This communication mode is not covered by FIPA.

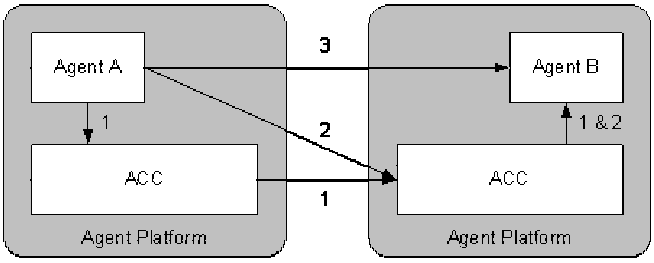

Figure 1: Three Methods of Communication between Agents on Different Agent Platforms

For now, the most commonly used communication method is based largely on the first of the three methods above. This paper will address the problem of securing data transmission between two different agent platforms over insecure channels. In doing so the following assumption has been made; the agent platform itself is secure. This means that intra-platform communication isn't a threat to the domain, as a result of this the focus of this paper concerns secure inter-platform communication.

2. Problem Analysis
This section will try to address the problem by identifying the security requirements and suitability of various secure communication mechanisms used with HTTP. The modelling choice for HTTP is encouraged by factors pertaining to the robustness and advantages of HTTP in general against other Message Transport Protocols such as RMI or IIOP in FIPA. HTTP also runs on ports that most often permitted by most firewalls and has been generally accepted as the de-facto transport protocol for data communication.

FIPA message specification employs MIME (RFC822) [3] standard. RFC822 specifies that Encryption Types for mail may be assigned. There are currently no RFC822 encryption types

[1] A fourth possibility (not illustrated) is that instead of completing the last two stages of the first path, the ACC on the first platform contacts Agent B directly – this depends upon the address that the ACC is delivering to.

assigned. Therefore usage of the Mail Privacy procedures is recommended [4] in RFC1421 [5]. Privacy Enhanced Mail (PEM) [6] or RFC1421 were in the past a popular cryptographic technique for authentication and privacy of messages. Unfortunately the success of PEM has suffered because of drawbacks or flaws that have become apparent over the years, such as the following:

- *Support Infrastructure*: the IETF standard that addresses secure e-mail proposes using a hierarchy of trusted bodies to reassure users of the validity of a particular e-mail message. At the top-level sits the Internet Policy Registration Authority (IPRA), which would be the governing certificate trusted by all. The IPRA would sign certificates for a second layer of trusted bodies called Policy Certification Authorities (PCAs). These in turn would authorize certificates for another layer of bodies called Certificate Authorities (CAs). In spite of this, there are several products on the market or the Internet that follow the PEM model, including ones from RSA Data Security, Trusted Information Systems, and Michigan State University (called RIPEM). However, because the certificate hierarchy suggested by PEM (the IPRA model) hasn't been established, some of these products use proprietary CA schemes. In other words, the lack of the infrastructure, public directories, has been the main obstacle in proliferation of PEM. For example, RIPEM, the flagship of PEM, does not implement certificates.
- *Surreptitious Forwarding*: PEM essentially provides only two variants of mail security; a message can simply be signed, or it can be signed and then encrypted. PEM has no notion of signing or authenticating ancillary attributes, and also doesn't support extra crypto layers. To prevent surreptitious forwarding, a PEM message's author would have to include the recipient's name directly in the message-body. Of course, it could be very difficult for the receiving PEM mail-client to find the recipient's name in the body, thus making it difficult to automatically prevent surreptitious forwarding.

Due to some of the PEM flaws mentioned above, newer improved standards have been developed such as S/MIME (Secure/Multipurpose Internet Mail Extensions) [7] and PKCS 7:CMS (Cryptographic Message Syntax) [8], which provide a consistent way to send and receive secure MIME data. Based on this standard, we have proposed S/MIME to be complemented with FIPA Message Specification to provide secure electronic messaging: authentication, message integrity and confidentiality.

2.1 Message Structure

The FIPA Agent Message Transport Protocol for HTTP Specification is based on MIME Specification for multipart message content type [13], this contains an envelope section based on RFC822 [12] and a content section based on FIPA ACL Message Structure Specification [14]. To provide security, using S/MIME over current HTTP Specification, we have re-defined the content multipart section of the HTTP message to a S/MIME multipart using PKCS 7 Specification. This re-definition doesn't change the overall specification, but merely contains the unencrypted ACL into a S/MIME definition that already resemble the current Multipart MIME Message Structure. The S/MIME section of the content utilises CMS to provide the necessary definition of security components and encryption of the ACL Message. This S/MIME part is also encoded in Base 64 encoding to ensure end to end transmission reliability of irregular character types. The figure below (Figure 2), expresses how simply an S/MIME feature can be integrated to the present FIPA HTTP Message Structure:

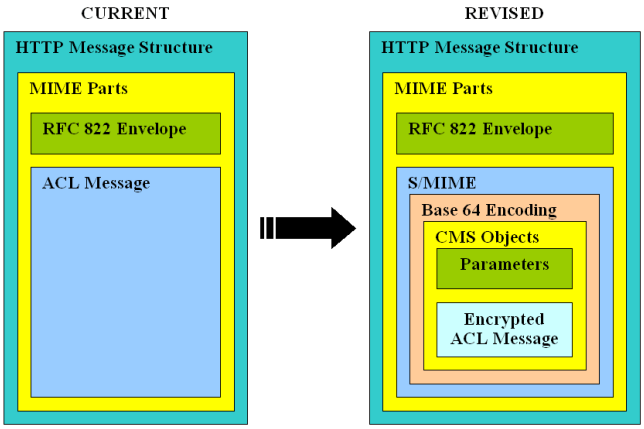

Figure 2: Changes to Current FIPA HTTP Message Structure to support S/MIME

In this context, S/MIME and PKCS 7 provides 4 core (there are more content types that are not listed) content types of security mechanisms:

- Signed Data
- Enveloped Data
- Clear-Signed Data
- Signed and Enveloped Data

In the effort to provide secure communication between inter-platforms, the use of S/MIME Enveloped data has been employed as a first step towards platform-to-platform security. The remainder of this paper will also provide an architecture of the system followed by use cases describing the high level design.

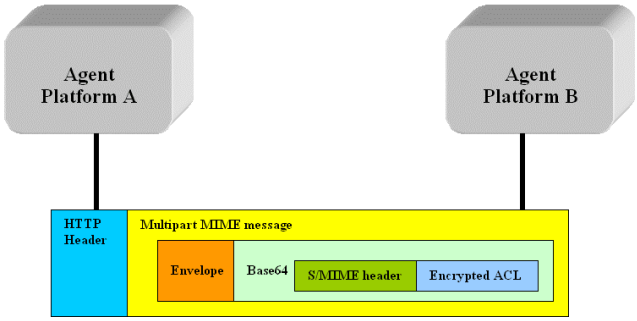

Figure 3: HTTP message structure

Based on the above architecture (Figure 3), some of the strong points have been noted in support of using S/MIME and PKCS 7 with FIPA Message Transport Specification, these are as follows:

- Clear separation between security related data (the signed attributes) and the application data (the content).
- CMS is widely used syntax for S/MIME enabled messaging that is platform-independent as well.

- Allows double signing of messages, especially catered for multi agent to multi agent messaging.
- Privacy in end-to-end communication which does not impact mail processing by intermediate relay hosts that do not incorporate privacy facility.

2.2 Use Case

This section discusses the steps in processing a message encrypted in S/MIME when it is transmitted from one ACC to another.

Step 1: The ACC of an agent platform receives a HTTP message from another agent platform's ACC. The message contains MIME bodies with a S/MIME part. The Message Processing use case strips the message into separate boundaries and relays the parts into the relevant parsers and message strippers.

Step 2: The encoded S/MIME message is decoded to retrieve the Java CMS objects. From the objects, the content key is decrypted using either a previous symmetric key or the private key of the recipient.

Step 3: Once the content decryption key is retrieved, the encrypted content can be decrypted to retrieve the ACL Message and its relevant information. Following this, the ACL Message is passed to the Message Transport System (MTS) to be routed to the final recipient.

3. S/MIME and CMS

This section defines how secure message exchange between two FIPA compliant agent platforms is constructed. This section also provides examples of CMS objects, and the choices and functions that are employed.

The process by which enveloped data is constructed involves the following steps [8]:

1. A content-encryption key for a particular content-encryption algorithm is generated at random.
2. For each recipient, the content-encryption key is encrypted with the recipient's public key.
3. For each recipient, the encrypted content-encryption key and other recipient-specific information is collected into a RecipientInfo value, defined in Section 3.1.2 and 3.1.3.
4. The content is encrypted with the content-encryption key. (Content encryption may require that the content be padded to a multiple of some block size)
5. The RecipientInfo values for all the recipients are collected together with the encrypted content into an EnvelopedData value, defined in Section 3.1.2.
6. It is assumed that the enveloped data contains some binary information. Therefore Base64 Content-Transfer encoding is used.

A recipient platform opens the envelope by first decoding and then decrypting the encrypted content-encryption keys with the recipient's private key and decrypting the encrypted content with the recovered content-encryption key. The recipient's private key is referenced by an issuer distinguished name and an issuer-specific serial number that uniquely identify the certificate for the corresponding public key. In a similar scenario, two communicating platforms may also use symmetric key exchange to public key when continuing previous communications using an earlier symmetric key. A big picture of HTTP Messages using MIME bodies coupled with S/MIME parts using PKCS-7 can be seen in the *Appendix* prior to the next part that is the CMS objects.

3.1 CMS using ASN.1 Notation for S/MIME Body

This section is divided into three parts. The first part describes the top-level type EnvelopedData, the second part describes the per-recipient information type RecipientInfo, and the third part describes the content-encryption and key-encryption type.

3.1.1 Top Level Type Enveloped Data

The top level EnvelopedData CMS Notation described below contains the identifier and enveloped data portions. The identifier defines the standards and specifications used for constructing an enveloped data S/MIME content type security mechanism. As for the other portion, it describes the version info, recipient info and encrypted content info that will be discussed in further detail in later sections of this document.

```
id-envelopedData OBJECT IDENTIFIER ::= {
        iso(1) member-body(2)
        us(840) rsadsi(113549) pkcs(1) pkcs7(7) 3
}

EnvelopedData ::= SEQUENCE {
        version 0,
        recipientInfo RecipientInfo,
        encryptedContentInfo EncryptedContentInfo
}
```

3.1.2 Per Recipient Information Type

This part describes the recipient information type; it consists of the choices available between using asymmetric or symmetric key exchange for content decryption. Both methods are valid depending on a given situation, for example, if two platforms were communicating, on their first attempt they would be using the RSA type objects. But if the platforms have communicated before they may choose the second option using symmetric keys for greater efficiency. There are 3 key management algorithms available in CMS [9], they are:

- *Key transport*: the content-encryption key is encrypted in the recipient's public key;
- *Key agreement*: the recipient's public key and the sender's private key are used to generate a pairwise symmetric key, then the content-encryption key is encrypted in the pairwise symmetric key; and
- *Symmetric key-encryption keys*: the content-encryption key is encrypted in a previously distributed symmetric key-encryption key.

In this document, we use two of the three available key management algorithms to secure communication between two platforms. The algorithms are expressed in Section 3.1.2.1 and 3.1.2.2 below.

3.1.2.1 Key Transport Algorithm

This section has described the Recipient Info that contains version, key identifiers, encryption algorithm used and the

encrypted key (will be described in Section 3.1.3) data. The key identifier refers to a unique conversation id and may also include more than one recipient. The encryption algorithm here refers to RSA encryption due to the nature of this algorithm that supports secure communication for the first time between two agent platforms that would latter facilitate symmetric key generation (can be used in conjunction with concepts in Section 3.1.2.2)

```
-- Used for first time connection, using RSA encryption of keys
-- Start RSA type object
RecipientInfo ::= SEQUENCE {
        version 2,
subjectKeyIdentifier
    agentB@agents.elec.qmul.ac.uk/12345678,
        KeyEncryptionAlgorithm rsaEncryption,
        encryptedKey EncryptedKey
}
rsaEncryption OBJECT IDENTIFIER ::= {
        iso(1) member-body(2)
        us(840) rsadsi(113549) pkcs(1) pkcs-1(1) 1
}
-- End of RSA type object
-- use either the RSA (above) or Triple-DES (below) type objects for key encryption
```

3.1.2.2 Symmetric Keys-Encryption-Keys Algorithm

This algorithm caters for secure communication using a previously agreed or newly generated symmetric key between two agent platforms. As in the previous section, one or more sets of Key-Encryption-Key (KEK) recipients' info with the desired Key Identifier (specified with a unique ID and a timestamp) can be defined, and lastly the key encryption algorithm is based on Triple-DES algorithm that is described in ANSI X9.52 [15]. The Triple-DES is composed from three sequential DES [16] operations: encrypt, decrypt, and encrypt. Also, the key encryption algorithm must be declared in Cipher Block Chaining (CBC) mode making the algorithm 'des-ede3-cbc' being declared in the syntax (CBC is mandatory in CMS).

```
-- Used for following connection (presumably Secure
  Tunnelling)
-- uses Triple-DES or RC2/40
-- Start of Triple-DES type object
RecipientInfo ::= CHOICE {
        Kekri KEKRecipientInfo
}
KEKRecipientInfo ::= SEQUENCE {
        version 4,
        kekid KEKIdentifier,
        keyEncryptionAlgorithm des-ede3-cbc,
        encryptedKey EncryptedKey
}
KEKIdentifier ::= SEQUENCE {
        keyIdentifier 12345678,
        date 20020315T083000000Z
}
-- End of Triple-DES type object
```

3.1.3 Content Encryption and Key Encryption Type

This part declares the encrypted key defined in the previous sections. The encrypted content info complements this declaration to provide information regarding the content type being used for instance 'Enveloped Data' in our example, 'des-ede3-cbc' encryption algorithm as per defined in Section 3.1.2.2, and the encrypted content which is a set of Octet String (64 bits). The encryption algorithm object identifier is defined in this part with associated information such as the initial CBC parameter. This parameter contains an Initialisation Vector (IV) that precedes the protected (encrypted) payload to make CBC happen [17]. Finally, the IV is declared as a set of Octet String, followed by the encrypted content payload.

```
EncryptedKey ::= OCTET STRING
EncryptedContentInfo ::= SEQUENCE {
        contentType envelopedData,
        contentEncryptionAlgorithm des-ede3-cbc,
        encryptedContent EncryptedContent
}
des-ede3-cbc OBJECT IDENTIFIER ::= {
        iso(1) member-body(2)
        us(840) rsadsi(113549) encryptionAlgorithm(3) 2
}
CBCParameter ::= IV
IV ::= OCTET STRING
EncryptedContent ::= OCTET STRING
```

4. Implementation

Before discussing the implementation, it is important to make clear that presently the system implementation only deals with securing inter-platform communication and not with intra-platform communication. The reason for this is that it is assumed that agents within a particular platform are trusted and therefore the platform itself is relatively secure. The other reason would be to avoid double encryption where a message is encrypted twice, once between intra-platform and another between inter-platform communications. Encryption is an expensive (processor power wise) procedure and over-using it is generally not a good idea.

The main advantage of the approach outlined here is that in securing inter-platform communication there is no need for changes in current ACL specification. As mentioned earlier, FIPA message specification employs MIME (RFC822), applying S/MIME protocol on top the current FIPA message structure will require just several additional tags (see section 3). If there is no encryption required these fields can be left blank.

At the agent level there is a need to introduce a new method, which allows an agent to request for secure communication. For example, assuming that there is a FIPAAgent.forward() method within a FIPA platform used to pass messages to the MTP. In order to specify a request for secure information transfer, a new FIPAAgent.secureForward() method must be introduced. Please note that this is only an option, and from a performance perspective it makes more sense not to overuse encryption. In order to do so FIPAAgent.forward() has been defined (i.e. unencrypted message transfer) as the default message sender method. The reason for this pertains to the paradigm of agent communication, where encryption is not always needed and is only needed when a user deems that security is necessary,

allowing him or her to explicitly specify the FIPAAgent.secureForward() method for secure message transfer.

There should also be an option to specify an encryption level and the algorithm used for secure communication (cipher suite), these options can be parsed as parameters to the FIPAAgent.secureForward() method. If no parameters have been specified the default cipher suite will be used. There also should be an option to specify a default cipher suite by invoking setCipher() method. Individual system developers can make this decision independently of their system requirements. If one of the platforms is not able to fulfil the cipher requirements, a CipherNotSupported message could be sent back to indicate that the receiver failed to decrypt the message supported by this particular cipher suite. In this scenario, the sender can either encrypt MIME contents of the message with one of the
algorithms or abort the communication all together. S
situation where a recipient may not support a recommended
cipher suite
cipher s
streng
weak
secu
proce
specifi
and 3, w
only if the
3. Consequently, Agent A sends a level 3-encrypted message to the other party and it appears that the recipient does not support this particular algorithm, and as a result it would reply with a CipherNotSupported message, also indicating its supported ciphers. Agent A can now check which ciphers between level 2 and 3 are supported and resend the encrypted message using one or the other cipher suites.

it makes sense to classify all available
levels according to the cryptography
level two, etc. starting from the
agent developer to specify the
information can be used in the
suites. For example, Agent A has
oundaries() method parameters 2
t agrees to establish communication
gent B uses ciphers between level 2 and

Our cur
communi
important
For exam
be tied to
problems su
IME, secure communicat
protocol (e.g. HTTP), how

[12] FIPA Agent Message Transport Protocol for HTTP Specification. Foundation for Intelligent Physical Agent, 2000. http://www.fipa.org/specs/fipa00084/

[13] MIME (Multipurpose Internet Mail Extensions) Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies http://www.faqs.org/rfcs/rfc1521.html

[14] FIPA ACL Message Structure Specification. Foundation for Intelligent Physical Agent, 2000. http://www.fipa.org/specs/fipa00061/

[15] American National Standards Institute. ANSI X9.52-1998, Triple Data Encryption Algorithm Modes of Operation. 1998.

[16] American National Standards Institute. ANSI X3.106, "American National Standard for Information Systems – Data Link Encryption". 1983.

[17] The ESP DES-CBC Cipher Algorithm With Explicit IV, 1998. http://www.ietf.org/rfc/rfc2405.txt

[18] Steve Kille, Which Protocols for Secure E-Mail and Secure E-Business?, 1999.

[19] Lars Eggert, Steve Hotz, PGP versus SSL for X-Bone Authentication and Security, 3rd Draft, 1999. http://www.isi.edu/~larse/work/pgp-vs-ssl.txt

8. Appendix

8.1 FIPA HTTP Messaging using S/MIME

The FIPA HTTP MTP is based on the transfer of data represented in MIME format containing both the envelope and ACL Message. The HTTP data transfer is a two-step process: the sender makes a HTTP request and the receiver sends a HTTP response [1]. The receiver parses the message envelope where it is handled according to instructions and information given in the message envelope.

The structure of the HTTP message is represented in Figure 3. As defined in Sections 2.1 [1] and 3, the message consists of a HTTP header and body, which is Multipart/Mixed MIME message (RFC2046) [10]. The encryption of such message is done using CMS objects in conjunction with MIME standards, making S/MIME [7].

The two subsections below describe an overview of the secure MIME messages transferred between two ACCs with a successful response.

8.1.1 ACC Sending HTTP Message

```
POST http://foo.com:80/acc HTTP/1.1
Cache-Control: no-cache
Host: foo.com:80
Mime-Version: 1.0
Content-Type: multipart-mixed ;
        boundary="251D738450A171593A1583EB"
Content-Length: 1518
Connection: close

--251D738450A171593A1583EB
Content-Type: application/xml

<?xml version="1.0"?>
<envelope>
 <params index="1">
  <to>
   <agent-identifier>
    <name>receiver@foo.com</name>
    <addresses>
     <url>http://foo.com/acc</url>
    </addresses>
   </agent-identifier>
  </to>
  <from>
   <agent-identifier>
    <name>sender@bar.com</name>
    <addresses>
     <url>http://bar.com/acc</url>
    </addresses>
   </agent-identifier>
  </from>
                <acl-representation>fipa.acl.rep.string.std</acl-
representation>
   <payload-encoding>US-ASCII</payload-encoding>
   <date>20000508T042651481</date>
   <encrypted>S-MIME</encrypted>
   <received >
    <received-by value="http://foo.com/acc" />
    <received-date value="20000508T042651481" />
    <received-id value="123456789" />
  </received>
 </params>
</envelope>

--251D738450A171593A1583EB
Content-Type: application/pkcs7-mime; smime-
type=enveloped-data;
        name=smime.p7m
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename=smime.p7m

ghyHhHUujhJhjH77n8HHGTrfvbnj756tbB9HG4VQpfyF467GhIGfHfYT6
4VQpfyF467GhIGfHfYT6jH77n8HHGghyHhHUujhJh756tbB9HGTrfvbnj
n8HHGTrfvhJhjH776tbB9HG4VQbnj7567GhIGfHfYT6ghyHhHUujpfyF4
7GhIGfHfYT64VQbnj756

 --251D738450A171593A1583EB--
```

8.1.2 The ACC responds with a successful notification

```
HTTP/1.1 200 OK
Content-Type: text/plain
Cache-Control: no-cache
Connection: close
```

Identifying collusions: Co-operating malicious hosts in mobile agent itineraries

E.C. Vijil
School of Information Technology
Indian Institute of Technology,
Bombay, India - 400076
vijil@it.iitb.ac.in

Sridhar Iyer
School of Information Technology
Indian Institute of Technology,
Bombay, India - 400076
sri@it.iitb.ac.in

ABSTRACT

A mobile agent is vulnerable to attacks by malicious hosts executing it. One of these attacks is the tampering of the data being carried by the agent. This is particularly relevant in comparison shopping scenarios where the mobile agent collects price quotes from various hosts. A malicious host in the itinerary could modify/delete the prices quoted by previous hosts, and/or insert spurious prices on behalf of latter hosts and modify the itinerary.

While there is some work on detecting tampering by individual hosts, there is not much work on identifying malicious hosts that may *collude* with each other to tamper with the mobile agent's data. In this paper[1] we present mechanisms that enable the detection of tampering by individual as well as colluding hosts, and also the identification of such hosts.

1. INTRODUCTION

Mobile agents are software programs that may move from one host to another to perform computations on behalf of their owner. The set of hosts visited by the agent (termed as the *itinerary*), may be *static*, i.e., pre-determined by the agent owner, or *dynamic*, i.e., decided by the agent as it moves in the network. The reader may refer to [2], for a survey of the mobile agent design paradigm, different agent frameworks and possible applications. One interesting application for mobile agents is comparison shopping in e-commerce [3]. Here, the agent owner launches the agent with a description of the goods he wishes to purchase and the agent visits several hosts in the e-market, collects their price quotes and returns to the owner.

At any host, the execution environment of the agent is controlled by the host. Hence mobile agents are vulnerable to attacks by malicious hosts [1, 4]. One of these attacks is the tampering of data being carried by the agent. For example, in the above comparison shopping scenario, a malicious host could modify/delete the prices quoted by other hosts, and/or modify the itinerary itself.

Several schemes have been proposed for detecting the modification of an agent's data by individual malicious hosts [3, 5]. However, these schemes do not address the problem of two or more malicious hosts *colluding* with each other to delete the data of other hosts. For example, suppose an agent visits hosts $H_1, \cdots, H_i, H_{i+1}, \cdots, H_j, H_{j+1}, \cdots, H_n$. Now if hosts H_i and H_{j+1} are both malicious and collude with each other, they may delete the data of $H_{i+1}, \cdots, H_j$ without being detected.

In [5], Karnik et. al, propose the notion of *AppendOnlyContainer* for detecting the tampering of an agent's data by individual malicious hosts. However the mechanism does not indicate the identity of the malicious hosts. Also, two hosts *colluding* to delete the data of intermediate hosts may escape detection in this mechanism.

In this paper, we incorporate and extend the notion of the *AppendOnlyContainer* to include not only the detection of tampering but also the identification of the malicious host. Subsequently, we introduce the notion of Expected Number of Deletions (END), that helps us to detect deletion of data by colluding malicious hosts in static as well as dynamic itineraries.

The paper is organized as follows: In Section 2 we discuss the *AppendOnlyContainer* and in Section 3 we describe our extension to this mechanism that enables identification of the malicious host. In Sections 4 and 5, we describe our schemes for detecting deletion of data by colluding malicious hosts. Section 6 concludes with a discussion of related work.

We use the following notation throughout this paper:

$E_A(X)$	Encryption of data X using public key of A.
$D_A(X)$	Decryption of data X using private key of A.
$hash(X)$	A one way hash on the data X.
$Sig_A(X)$	Signing of $hash(X)$ using private key of A.

2. AOC: APPEND ONLY CONTAINERS

In this section, we briefly discuss the *AppendOnlyContainer* (AOC) mechanism proposed in [5]. Given an agent that visits several hosts to collect data, the agent owner may use the AOC mechanism to detect any modification/deletion of data by individual malicious hosts. The basic idea in AOC is: For each visited host C, record

1. the data collected at the host, X,
2. the digital signature of the host on that data, $Sig_C(X)$, and

[1]This project was funded by the Ministry of Information Technology, India.

3. the identity of the signing host, C.

A *checksum* is used to detect modification/deletion of data from the AOC. The computation of the *checksum* and its verification is discussed below.

When an agent starts its itinerary, its AOC is empty. The checksum is initialized by encrypting a random nonce with the agent owner's public key:

$$checkSum = E_{owner}(N_a) \quad (1)$$

This nonce N_a is kept secret and is not carried by the agent.

When a host C wants to insert data X, C first signs X using its own private key, D_C. Then the data item X, its signature $Sig_C(X)$ and the identity of the host C, are inserted into the appropriate arrays in the AOC. The checksum is then updated as follows:

$$checkSum = E_{owner}(checkSum + Sig_C(X) + C) \quad (2)$$

where + denotes *concatenation* of the corresponding values.

When the agent returns to the owner, the integrity of the data is verified by decrypting the *checksum* and verifying the signature, in an iterative manner. Each step of the iteration is:

$$D_{owner}(checkSum) \rightarrow checkSum + Sig_C(X) + C \quad (3)$$

followed by verifying if:

$$E_C(Sig_C(X)) == hash(X) \quad (4)$$

If in the last iteration, the agent owner recovers the original random nonce N_a, it can be inferred that the AOC has not been tampered with.

If the verification procedure fails in any iteration, the agent owner can infer that the AOC has been tampered with. It also implies that the values extracted up to this iteration are valid, while other values whose signatures are still nested within the checksum cannot be relied upon.

In [6, 7], Roth describes an attack that will allow a malicious host to forge the AOC. The problem arises because a malicious host can abuse other hosts as oracles for signing and checksum computation. Assume that a malicious host M receives an agent A_O created by host O. Let the *checksum* carried by that agent be $checksum_O$. M creates another agent A_M and initializes its checksum with $checksum_O$. M can successfully add all the data added to A_M's AOC to A_O's AOC without risking detection.

A solution to this problem, also suggested in [7], is as follows: For each agent instance, a unique identifier is constructed by the owner as $id = hash\,(D_{owner}\,(code, timestamp))$, where *code* is the agent's code. The *timestamp* is used to distinguish between multiple instances of the same agent. This unique identifier is carried by the agent while it visits the hosts in its itinerary. Whenever a host C signs a data item X, it actually computes $D_C(id, X)$. This means that X is valid only in the context of the agent instance *id*. During verification, the host owner needs to check the context of the agent instance and verify that each data item was added in the context of the correct agent instance.

Since the above problem does not bear directly upon the focus of this paper, in the interest of simplicity, we continue to use the basic AOC mechanism for our discussion. Our proposal can be easily extended for an AOC mechanism augmented with the above modification.

Now, we observe that while the AOC mechanism ensures the detection of: (i) any modification/deletion by a host to the data collected from earlier hosts, and (ii) any modification by a host to its own data after it has been added to the AOC, it does not indicate the identity of the malicious host. Also, as explained subsequently (section 4.1), two hosts *colluding* to delete the data of intermediate hosts may escape detection in the AOC mechanism. In the next sections, we incorporate and extend the notion of AOC in order to provide solutions to these problems.

3. X-AOC: IDENTIFYING THE MALICIOUS HOST

In the AOC mechanism a host C upon adding data X into the AOC, re-computes the *checksum* using $Sig_C(X)$ and C. While this is sufficient for the agent owner to detect tampering of data by a malicious host, it does not indicate the identity of the malicious host.

We propose *X-AOC*, an extension to the AOC mechanism that enables the identification of the malicious host. The main idea is: A host C upon adding data X to a *container*, should append $Sig_C(container)$ and C, to the *container*. $Sig_C(container)$ is used to re-compute the *checksum*. In other words, a host, instead of merely signing the data item added by it, is required to sign the entire contents of the container at that point.

X-AOC assumes that all hosts in the itinerary add some data to the container. In case of multiple malicious hosts, X-AOC indicates the identity of the *last* malicious host that added data to the container. The detailed algorithm for X-AOC is shown in Figure 1.

The checksum is initialized as in equation 1. To insert an item X, any given host C first adds X to the *container* and signs the *container* using its private key, D_C. The checksum is then updated as follows:

$$checkSum = E_{owner}(checkSum + Sig_C(container) + C) \quad (5)$$

When the agent returns to the owner, the integrity of the data is verified by decrypting the checkSum and verifying the signature, in an iterative manner. Each step of the iteration is:

$$D_{owner}(checkSum) \rightarrow checkSum + Sig_C(container) + C \quad (6)$$

followed by verifying if:

$$E_C(Sig_C(container)) == hash(container) \quad (7)$$

If in the last iteration, the agent owner recovers the original random nonce N_a, it can be inferred that the *container* has not been tampered with.

If the verification procedure fails in any iteration, the agent

```
class X-AOC{
    Vector dataContainer;
    byte[] checkSum;
    X-AOC (PublicKey k, int nonce){
        dataContainer = new Vector();
        checkSum = encrypt (nonce); // with key k
    }
    public void checkIn(Object X) {
        dataContainer.addElement(X);
        sign = host.sign(datacontainer);
        checkSum = encrypt (checkSum + sign + hostId);
    }
    public boolean verify (PrivateKey k, int nonce){
        previoushost = null;
        loop{
            checksum = decrypt (checkSum);
            sign = extract.sign (checksum);
            currenthost = extract.hostId (checksum);
            checkSum = extract.checkSum (checksum);
            If (Verify(sign) == TRUE), then
                previoushost = currenthost
            Else
                throw security exception.
            // Either currenthost or previoushost is malicious.
        }until (checkSum == nonce);
    }
}
```

Figure 1: ***The X-AOC mechanism***

owner can infer that the *container* has been tampered *by either the current host or the previous host.* For example, suppose hosts $H_1, \cdots, H_i, H_j, \cdots, H_n$ are visited by the agent, in that order. Without loss of generality, let the verification fail while verifying $Sig_{H_i}(container)$. This implies that the tampering was done either by H_i after generating the checksum, or by H_j before generating the checksum. Hence, the malicious host can be identified to be one of either H_i or H_j.

Thus, the X-AOC mechanism not only detects modification/deletion of data by malicious hosts, but also indicates the identity of the malicious host. In the next section we discuss mechanisms to detect collusions among malicious hosts, in static as well as dynamic itineraries.

4. COLLUDING MALICIOUS HOSTS

In this section we address the problem of collusion among malicious hosts and propose some solutions to the same. We use the AOC mechanism to demonstrate collusions. However, our solution can be easily extended to any protocol that uses other variants of signature chaining to protect data items.

4.1 Attack

Suppose an agent visits hosts $H_1, \cdots, H_i, H_{i+1}, \cdots, H_j, H_{j+1}, \cdots, H_n$, in that order. Further, assume that hosts H_i and H_{j+1} are both malicious and collude with each other.

Host H_i on receiving the agent, does the following:

1. It adds its own data D_i and signature $Sig_{H_i}(D_i)$ to the AOC.
2. It re-computes the checksum as given in equation (2). We shall denote this checksum by $checkSum_i$.
3. It forwards the agent to H_{i+1} and sends $checkSum_i$ to H_{j+1}.

H_{j+1} on receiving the agent does the following:

1. It removes data items $D_{i+1}, \cdots, D_j$ from the AOC.
2. It adds its own data D_{j+1} and signature $Sig_{H_{j+1}}(D_{j+1})$ to the AOC.
3. It re-computes the checksum as given in equation (2). However, it uses $checkSum_i$ instead of $checkSum_j$.
4. It forwards the mobile agent to the next host in the itinerary.

In the conventional AOC mechanism, the agent owner would be unable to detect that items $D_i, \cdots, D_j$ have been removed from the AOC.

We now propose solutions for detecting collusions among malicious hosts, in static as well as dynamic itineraries.

4.2 Detecting collusions in static itineraries

In the case of a *static itinerary*, the agent owner apriori knows the identities of the hosts to be visited by the agent. The order in which these hosts are visited may also be static or may be dynamically decided by the agent.

The X-AOC mechanism can be easily extended for detecting collusions in a static itinerary, as follows:

1. Each visited host is required to compulsorily add some data to the *container*, and
2. The agent owner while verifying the integrity of the *container*, checks if there is data corresponding to each host in the itinerary.

If the data corresponding to a host H is missing, then it indicates that the data was deleted by colluding malicious hosts. If the order in which various hosts are visited is also known, then the missing data also gives an indication of the identities of the malicious hosts.

In the next section we present our scheme for detecting collusion in dynamic itineraries.

5. DETECTING COLLUSIONS: DYNAMIC ITINERARIES

In the case of a *dynamic itinerary*, the agent owner does not apriori know the identities of the hosts to be visited by the agent. The set of hosts to be visited (and the order in which they are to be visited) is dynamically decided by the agent.

A simple extension to the X-AOC mechanism for detecting collusions in a dynamic itinerary would be:

1. Any given host C, adds data X to the *container*, and updates the *checksum* as in equation 5.

2. C sends a *notification* M_{C^X}, to the agent owner, indicating that C was visited.

3. The agent owner while verifying the integrity of the *container*, as in equations 6, 7, also checks if there is data corresponding to each host that sent a *notification*.

For any *notification*, if the corresponding data is missing from the *container*, then it implies that the data was deleted by colluding malicious hosts.

However, the above solution is expensive in terms of the number of notifications required. We argue that all hosts need not participate in the collusion detection process. We use a notion called *Expected Number of Deletions* to reduce the number of notifications, without significantly reducing the ability of the owner to detect collusions.

5.1 Expected Number of Deletions

Let k and n be the owner's estimate of the number of *malicious* and *honest* hosts respectively. Now the malicious hosts may collude with each other to delete the data of one or more honest hosts. We assume that malicious hosts do not act against each other, and only the data of honest hosts may get deleted.

The Expected Number of Deletions (END) is the average number of honest hosts whose data may get deleted, assuming that all permutations of the itinerary are equally likely. If $P(m)$ is the probability that the data of exactly m honest hosts are deleted, then,

$$END = \sum_{m=0}^{n} mP(m) \tag{8}$$

We now have the following theorems.

THEOREM 1. $P(m) = (n+1-m)\frac{\binom{k-2+m}{k-2}}{\binom{k+n}{k}}$.

PROOF. Given k malicious hosts and n honest hosts, there are $k+n$ hosts in the itinerary and $\binom{k+n}{k}$ possible configurations of the itinerary. For a given configuration, let $x_1, \cdots, x_k$ be the positions of the malicious hosts. Without loss of generality, we assume that x_1 is the position of the first malicious host, x_k is the position of the last malicious host, and that there are m honest hosts between x_1 to x_k.

Let the positions in the itinerary be numbered in the range $[1, 2, \cdots, k+n]$. Now the highest value that x_k can take is $k+n$. Since there are m honest hosts and $k-2$ malicious hosts in between positions x_1 and x_k, the highest value that x_1 can take is $(k+n)-(k-2+m)-1$, i.e., $(n+1-m)$. Similarly, the lowest value that x_1 can take is 1 and the lowest value that x_k can take is $1+(k-2+m)+1$, i.e., $(k+m)$. Thus $1 \le x_1 \le (n+1-m)$ and $(k+m) \le x_k \le (k+n)$.

Now for a given m, the pair x_1, x_k can be chosen in $(n+1-m)$ ways. Also, there are $\binom{k-2+m}{k-2}$ configurations of the $(k-2)$ malicious hosts and m honest hosts in between x_1 and x_k.

Thus the probability that the data added by exactly m honest hosts will be deleted is given by
$P(m) = (n+1-m)\frac{\binom{k-2+m}{k-2}}{\binom{k+n}{k}}$. □

THEOREM 2. $END = \frac{n(k-1)}{k+1}$.

PROOF. We note that

$$\sum_{m=0}^{n} P(m) = 1$$

Hence from Theorem 1 we have

$$\sum_{m=0}^{n} (n+1-m)\binom{k-2+m}{k-2} = \binom{k+n}{k}. \tag{9}$$

Now

$$END = \sum_{m=0}^{n} mP(m)$$

Since, the $m = 0$ term does not contribute to END, we have

$$\begin{aligned}
END &= \frac{1}{\binom{k+n}{k}} \sum_{m=1}^{n} (n+1-m)m\binom{k-2+m}{k-2} \\
&= \frac{1}{\binom{k+n}{k}} \sum_{m=1}^{n} (n+1-m)m\frac{(k-2+m)!}{(m)!(k-2)!} \\
&= \frac{1}{\binom{k+n}{k}} \sum_{m=1}^{n} (n+1-m)\frac{(k-2+m)!}{(m-1)!(k-2)!} \\
&\quad \text{Let } m' = m-1 \\
&= \frac{1}{\binom{k+n}{k}} \sum_{m'=0}^{n-1} (n-m')\frac{(k+m'-1)!}{(m'!)(k-2)!} \\
&= \frac{1}{\binom{k+n}{k}} \sum_{m'=0}^{n-1} (n-m')\frac{(k-1)(k+m'-1)!}{(L!)(k-1)!} \\
&= \frac{k-1}{\binom{k+n}{k}} \sum_{m'=0}^{n-1} (n-m')\binom{k+m'-1}{k-1} \\
&\quad \text{Let } k' = k+1 \text{ and } n' = n-1 \\
&= \frac{k-1}{\binom{k+n}{k}} \sum_{m'=0}^{n'} (n'+1-m')\binom{k'+m'-2}{k'-2} \\
&\quad \text{Using equation 9} \\
&= \frac{k-1}{\binom{k+n}{k}}\binom{k'+n'}{k'} \\
&= \frac{k-1}{\binom{k+n}{k}}\binom{k+n}{k+1} \\
&= \frac{n(k-1)}{(k+1)}
\end{aligned}$$

□

Table 1: END: Experimentation results

Malicious Hosts (k)	Honest Hosts (n)	Notifications Sent (t)	Deletions Detected	Reduction in Notifications
5	20	3	96 %	77 %
5	20	5	99 %	63 %
5	50	3	95 %	91 %
5	50	5	99 %	85 %
15	50	3	96 %	93 %
15	50	5	99 %	90 %
20	100	5	100 %	95 %
30	100	5	100 %	94 %

It is interesting to note that END is directly proportional to n, the number of honest hosts and, its value approaches n as the number of malicious hosts, k, increases.

5.2 Reducing notifications

As mentioned earlier, detecting collusions by requiring each host to send notification to the agent owner is expensive in terms of the number of notifications. Let each host in the itinerary send notifications with a probability λ. Then, the average number of notifications sent by hosts whose data are deleted is given by $\lambda \cdot END$. We need just one notification from such a host to detect a collusion. However, to take into account the variance in the actual number of deletions from the estimate END, especially for small k, we require that at least 't' messages are sent, where $t \geq 1$. In other words, $\lambda \cdot END = t$ or $\lambda = \frac{t}{END}$. In practice, we found that a value of $t = 5$ works well.

While it may seem intuitive that a host needs to send a notification with a greater probability as k increases, this is not the case. This is because as k increases more honest hosts are likely to have positions in between the malicious hosts and hence an individual honest host now may reduce the probability for its sending a notification.

If k is large, then λ becomes close to $\frac{t}{n}$. If we have no idea about the number of malicious hosts, then we can be conservative and assume that $k = 2$. That is each host will send notifications with a probability slightly greater than $\frac{3t}{n}$.

We use END to reduce the number of notifications as follows:

1. The agent owner calculates END using Theorem 2.
2. The agent owner now calculates the value $\lambda = \frac{t}{END}$.
3. Each host on receiving the agent does the following:
   (a) It adds its data to the *container* and updates the *checksum* as given in equation 5.
   (b) It generates a random number, $r, (0.0 \leq r \leq 1.0)$.
   (c) If $r \leq \lambda$, it sends a *notification* to the owner.
   (d) It forwards the mobile agent to the next host in the itinerary.
4. The agent owner while verifying the integrity of the *container*, as in equations 6, 7, also checks if there is data corresponding to each host that sent a *notification.*
5. For any *notification*, if the corresponding data is missing from the *container*, then the agent owner infers that the data was deleted by colluding malicious hosts.

This probabilistic notification leads to a tremendous decrease in the number of notifications sent, without compromising much on the ability to detect collusions, since a single notification from any one of the affected hosts is sufficient to determine that a deletion due to collusion has occurred.

5.3 Experimentation

We performed experiments to determine the efficacy of our solution. In each run, the number of honest hosts and the number of malicious hosts were varied. An itinerary was chosen by a random permutation of hosts. The agent owner and the hosts follow the scheme in section 5.2. A collusion is detected by the agent owner if it receives a message from a host whose data would have been deleted by the colluding hosts. The actual number of deletions detected due to notifications were noted. The simulations show that our method can assure a high degree of confidence in detecting deletions while significantly reducing the number of notifications. The results are summarized in Table 1.

6. CONCLUSIONS

In a related work [3], Karjoth et. al., propose protocols to detect modification/deletion of data collected by free roaming agents. While some of these protocols are able to detect *modification* to the data by colluding hosts, they do not tackle the problem of hosts colluding with each other to *delete* the data of intermediate hosts in the itinerary. Some other problems regarding the robustness of these protocols are reported in [6].

We have extended the *AppendOnlyContainer* mechanism proposed in [5] to not only detect modification/deletion of data by individual malicious hosts, but also to identify the malicious host.

We have also proposed further mechanisms to detect malicious hosts that collude with each other to delete the data of other hosts, for static as well as dynamic itineraries. We have shown that the notion of Expected Number of Deletions helps us to significantly reduce the number of notifications that must be sent while offering a reasonable degree of confidence in detection of deletions.

7. ACKNOWLEDGMENTS

We would like to thank Volker Roth for his helpful suggestions and the email discussions that we had on the subject.

This project was funded by the Ministry of Information Technology, India, under the project "Mobile agents for collaborative distributed applications", 2001-2002. We would like to thank all the members of the project review committee for their encouragement and support in this project.

8. REFERENCES

[1] D. M. Chess. Security Issues in Mobile Code Systems. In G. Vigna, editor, *Mobile Agents and Security*, volume 1419 of *LNCS*, pages 1–14. Springer-Verlag, June 1998.

[2] A. Fuggetta, G. Picco, and G. Vigna. Understanding Code Mobility. *Transactions on Software Engineering*, 24(5):342–361, May 1998.

[3] G.Karjoth, N.Asokan, and G. Gulcu. Protecting the computation results of free-roaming agents. In *Proceedings of the second International workshop on Mobile agents (MA '98), LNCS 1477*, pages 195–207, Berlin Heidelberg, 1998. Springer Verlag.

[4] W. Jansen and T. Karygiannis. Nist special publication 800-19 - mobile agent security, 2000. National Institute of Standards Technology.

[5] N. Karnik and A. Tripathi. Security in the ajanta mobile agent system. Technical report, Department of Computer Science, University of Minnesota, Minneapolis, 1999.

[6] V.Roth. On the Robustness of some Cryptographic Protocols for Mobile Agent Protection. In *Proceedings of fifth International workshop on Mobile agents (MA 2001)*, Atlanta, USA, 2001. Springer Verlag.

[7] V.Roth. Programming Satan's agents. In *Ist International workshop on Secure Mobile Multi-Agent Systems*, Montreal, Canada, 2001.

Mobile Agent Security Facility for Safe Configuration of IP Networks

Kun Yang, Alex Galis
Department of Electronic and Electrical Engineering, University College London
Torrington Place, London WC1E 7JE
Tel.: +44 20 7679 5765
{k.yang | a.galis }@ee.ucl.ac.uk

Telma Mota
Portugal Telecomm Inovacao, SA.
Rua Eng. José Ferreira Pinto
Basto 3810-106 AVEIRO, Portugal
telma@ptinovacao.pt

Angelos Michalas
Department of Electrical and Computer Engineering, National Technical University of Athens,Greece
michalas@central.ntua.gr

ABSTRACT
Network management applications using mobile agents require secure techniques for the lifecycle of the mobile agent, from both mobile agent and host points of view. Based on the analysis of the security threats that may occur during the mobile agent based network management applications, this paper presents Mobile Agent Security Facility (MASF). MASF has several key features: 1) a secure mechanism for dispatching agents to given domains or network elements from secured agent repository; 2) provision of encrypted communication; 3) a safe mobile agents execution environment that enables mobile agents different resource access permissions according to the result of authentication and authorization; 4) logging services to record security relevant events. MASF architecture is further integrated and verified in a practical network management application, inter-domain IP VPN configuration.

General Terms
Management, Security

Keywords
Mobile Agent, Security, Network Management, IP Network

1. INTRODUCTION
The current network is characterized by its increasing distribution, its dynamic nature, and the complexity of its resources, due to the increasing requirement of different services. To manage such a network, it is getting more imperative to provide an overall network management mechanism that can quickly, intelligently and automatically configure the network elements and resources across the large-scaled networks. The integration of *mobile agent technology* (*MAT*) and policy-based network management (PBNM) provides a promising means to achieve this goal.

Policy-based Network Management (*PBNM*) technology offers a more flexible management solution for a specific application tailored to a customer under the environment of large-scaled networks [1]. Nevertheless, this flexibility doesn't come easily. The current PBNM architecture has problems in the sense that it can only address fairly limited issues and usually requires human intervention. Mobile agent, as an enabling technology, can resolve many of these problems. The mobile agent paradigm intends to bring an increased performance and flexibility to distributed systems by promoting "autonomous code migration" (mobile code moving between places) instead of traditional RPC (remote procedure call) [2]. With code migration, the actual code or script moves from place to place and executes locally, achieving lower latency, little need for remote interactions and highly flexible control. Mobile agents can easily represent one of the roles involved in the network management, such as service provider, connectivity provider, resource or end-user, and act on their behalf, based on established policies. Mobile agents are widely used in network management as they can effectively take over the burden of the complex interaction between different network players, such as negotiations or new service injection, so as to realize the automation of the network management.

Despite its many practical benefits, mobile agent technology results in significant new security threats from both malicious agents and hosts. For examples, as a mobile agent traverses multiple hosts that are trusted to different degrees, its state may be changed in a way that adversely impact its functionality. In the world of mobile agents, where a site can offer services not only to its local users but also to remote users, a number of serious security problems can develop. Many research work has been done about these problems [3], but most of which are undertaken theoretically and haven't been put into a real environment that is typical for mobile agent application and security problems, such as network management. This paper aims to give a concrete solution to the mobile agent security problems occurring during the use of mobile agents in IP network management.

The work described in this paper is part of MANTRIP (MANagement Testing & Reconfiguration of IP based networks using mobile software agents), a currently undergoing two-year collaborative EU IST project, whose main objective is to provide a set of novel IP network management applications using mobile agent technology.

The paper is organized as follow. Section 1, this section, described the background of the work presented in this paper. Section 2 briefly introduces the MANTRIP mobile agent platform and its security extension. Based on the analysis of potential security risks existing in the mobile agent based network management system, from both mobile agent and host points of view, and the necessary mechanisms that should be adopted, as introduced in Section 3, Section 4 develops a Mobile Agent Security Facility (MASF) that copes with the whole possible security threats possibly occurring in the lifecycle of mobile agent. MASF architecture is further integrated and verified in a practical network management application, inter-domain IP VPN configuration, which contributes to Section 5. Finally, Section 6 concludes the paper.

2. MANTRIP MOBILE AGENT PLATFORM AND ITS SECURITY EXTENSION

Since its inception in 1990s, mobile agent has attracted enormous attention from industry and institutes, which leads to a long list of mobile agent platforms developed [4], either for academic usage or commercial purposes. Among these, both Grasshopper [5] and Voyager [6] are selected as the mobile agent platforms used for network management application development in MANTRIP, based on the evaluation in MANTRIP Deliverable [7]. Both of these MA platforms are commercial products with extensive documentation and future development under way, therefore are more suitable for commercially oriented IP network management applications. Furthermore, both of them provide capabilities for wrappers development, resource management, logging services, move transparency, and certain degree of security services.

A common mobile agent API was developed in MANTRIP [7], which provides some MA services extensions and code portability between the two selected MA platforms (Voyager and Grasshopper). Introducing such an API in the MANTRIP System, each user will be able to program independently from the platform running on a visited machine in a uniform manner, as illustrated in Figure 1.

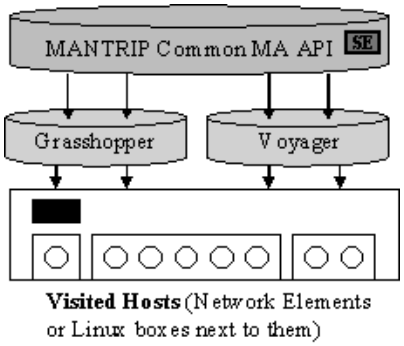

Figure 1: MANTRIP Common Mobile Agent API

The main terminologies used in the MANTRIP mobile agent platform adopt most of these from Grasshopper platform, which also conform to mobile agent industry standard given by OMG, namely the Object Management Group's Mobile Agent System Interoperability Facility (MASIF) [8]. Before describing the main part of this paper, some of these terminologies are briefly explained as follows: *Place*: provides a logical grouping of functionality inside an Agency; *Agency*: is the actual runtime environment for mobile and stationary agents; *Region*: facilitates the management of agencies and agents. Along with these elements there is also the concept of *agent*, which can be further categorized into mobile agent and stationary agent.

Both Grasshopper and Voyager provide certain degree of security services, which mostly come from the security mechanisms supplied by Java 2 (mainly JDK1.2). Based on these security services, a security extension is furnished in this paper. This security extension takes into account the potential security threats from both mobile agent and the host mobile agents intend to move to. Since all the actions of mobile agent have to be taken via its supporting environment, i.e., Agency and/or region, the host protection mechanisms have to be fulfilled via Agency and/or region. Figure 2 illustrates the position of MANTRIP MA platform and its security extension.

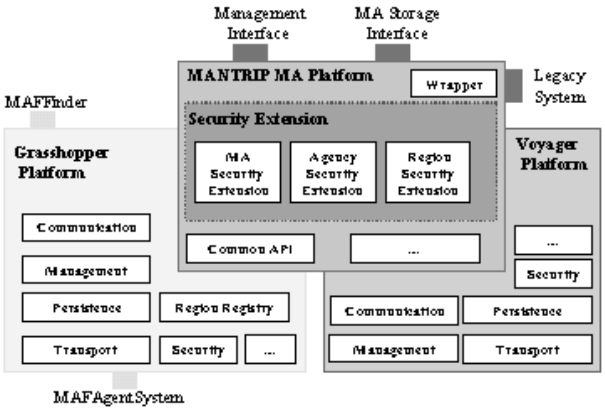

Figure 2: MANTRIP Mobile Agent Platform and its Security Extension

3. MOBILE AGENT BASED IP NETWORK MANAGEMENT AND ITS SECURITY

3.1 MANTRIP Network Management Architecture

The three basic network management applications that have been developed within MANTRIP project are the followings [7]:

- *Access Network Management Application*
- *QoS Configuration and Auditing Application*
- *Validation and Monitoring of Network Elements and Mobile Agents Application*

All three applications are carried out by mobile agents that are based on the MA common API. The architecture of QoS Configuration and Auditing Application is depicted in Figure 3. This architecture follows the overall MANTRIP Network Management System (NMS) architecture, although only the components and resources used for QoS and IP VPN are shown.

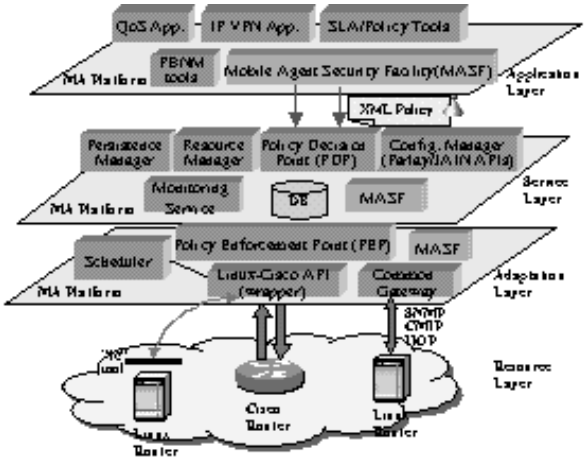

Figure 3: MANTRIP Network Management Architecture

The MANTRIP NMS has four layers as follows: *Application Layer*: includes the MANTRIP management user applications, e.g. QoS Configuration and Auditing Application; *Service Layer*: contains the MANTRIP management services (e.g. Parlay/JAIN API) that may be used by either the MANTRIP applications or some other third party applications; *Adaptation Layer*: is responsible for hiding the protocol details from the service layer; *Resource Layer*: contains the managed/controlled MANTRIP resources.

In MANTRIP QoS network management system, as shown in Figure 3, all the services in top three layers are implemented by agents, either stationary agents or mobile agents. Mobile agents, on behalf of service providers/network administrators or other services, can move themselves between different layers, and fulfill management tasks. The security threats that may occur during the whole lifecycle of mobile agent come from both malicious agents and the hosts that agents migrate to. The requirement for protection of mobile agents, especially in the case of multiple hops, might be increased when portions of its functionalities or data only apply to a specific domain or network element.

3.2 Security Threats and Strategies

Taking into account the whole lifecycle of MA, the potential threats, from both mobile agent and host points of view, can be:

Threat *A*: During mobile agent *storage*, the MA repository might be invaded and the class for the mobile agent might be changed before the initiation of the mobile agent.

During mobile agent *transit*, mobile agent is under great possibility to be attacked, such as:

Threat *B1*: When a mobile agent transports confidential data, disclosure of the data can be fatal. Since the migration of mobile agent often takes place across the networks that are out of the control of both sender and receiver thus cannot be physically secured.

Threat *B2*: The execution logic of mobile agent might also be changed by the interrupter, which might cause damage to the destination host of mobile agent. It is especially dangerous in MANTRIP system when mobile agents for network management are usually granted the right to configure routers or firewalls where Simple Network Management Protocol (SNMP), whose commands have the root permission of accessed elements, is used to configure the network elements.

After mobile agent's *arrival* at destination, the following threats can come up:

Threat *C1*: the supposed to be "destination" might not be the correct destination. This destination may be a counterfeit one created by business rival to steal the important information carried by mobile agent.

Threat *C2*: even if the destination is correct, mobile agent may still be deceived by malicious destination host. For example, it might not be provided the contracted services or resources, or might even be maliciously changed before it goes for another hop.

Threat *C3*: At the same time, the landing host of mobile agent should also be sure that the mobile agent is from the correct service contractor and it will not cause any damage to the host.

Threat *C4*: even if the mobile agent does come from the correct peer, the host still needs to keep alarmed on mobile agent's behaviors in case it may do something beyond the contract or its right.

To address these threats, the MANTRIP MA platform must provide the following strategies:

Authentication: involves checking that the agent was sent from a trustworthy role and also enables mobile agent to be aware of the real identity of the receiver, i.e., the proper Service Level Agreement (SLA) contractor. Authentication can be mainly used for the solving Threat *C1* and Threat *C3*. Authentication can also be used to check users who want to access mobile agent repository, which involves in the Threat *A*. *Confidentiality*: implemented by encryption/decryption, can cope with the potential data disclosure of Threat *B1*. Encryption can also prevents the mobile agent repository from attack, i.e., Threat *A*. *Integrity* check can prevent mobile agent from the code modification attack in Threat *B2*. *Authorization* determines the mobile agent's access permissions to the host resources and, empowered by access control, can defeat the potential threat from *C4*. *Logging*: it is a kind of mechanism to keep track of any security relevant events, such as agent's trying to access system resources or the system itself, as well as authentication failures. These events should be logged to a file for later analysis. Logging can, in some degree, detect and therefore latterly prevent the cheat of mobile agent from host, as described in Threat *C2*.

The implementation of these features, for the protection of both mobile agent and host, is carried out in *Mobile Agent Security Facility (MASF)* service that locates in all top three layers as shown in Figure 3.

4. MOBILE AGENT SECURITY FACILITY (MASF)

4.1 Design Goals of MASF

Based on the analysis of potential security risks existing in the mobile agent based network management system, the MASF was developed with the following goals:

- All mobile agents are stored at the secure mobile agent repository.
- All mobile agents are digitally signed, which assures that the classes received by Agency are the same as the classes created by the specific customer or service provider.
- A secure mechanism is provided for dispatching agents to given domains or network elements from secured agent repository.
- A safe mobile agents execution environment that enables mobile agents different resource access permissions according to the result of authentication and authorization.
- Logging of security relevant events happened to mobile agents and Agency.
- The design of MASF is based on the available standard solutions and products (e.g. the IAIK [9] SSL and J2SDK 1.4.0).

4.2 Extended Conceptual Model of Agency and Agent

In order to fully support security in mobile agent system, the concepts and mechanisms for security should be taken into

account at the stage of mobile agent design. Both Grasshopper and Voyager have support for security, which adopted the same standard solutions for the security problems existing in the mobile agent systems. This also makes the design of MANTRIP mobile agent system much easier. In MANTRIP MA system, the security relevant information is further extended for both Agency and Agent, as straightforwardly illustrated in Figure 4. Both agent and agency have certificate feature because agent and agency need to exchange certificate to get authenticated by each other. Rights for agent define the operations this agent can perform at the destination host.

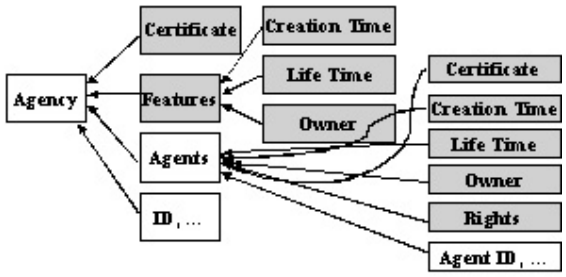

Figure 4: Security relevant Information in Agency and Agent

The actions that agents are authorized to perform depend on roles associated to agent principals. MASF permits the dynamic definition and control of a range of roles, from almighty administrators to normal users.

4.3 Mobile Agent Security Facility (MASF) Architecture

The previous section has given an idea of necessary components and services needed to build secure architecture for mobile agent based network management system. Figure 5 illustrates this architecture, called Mobile Agent Security Facility (MASF), together with major dependencies among components during the lifecycle of a multi-hop mobile agent. The MASF architecture is functionally divided into two layers, with the higher layer as *function layer* and lower layer as *base service layer*. The components or services in base service layer provide common functionalities needed by function layer.

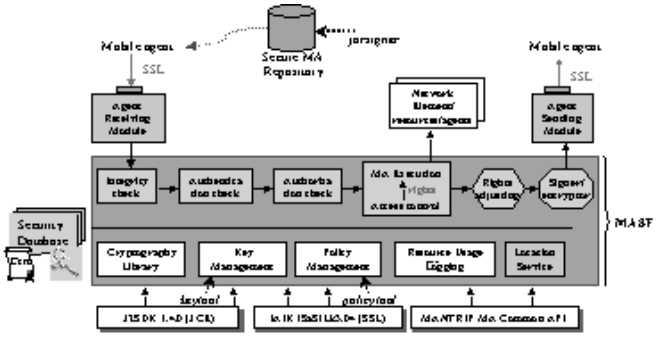

Figure 5: Mobile Agent Security Facility (MASF) Architecture

Obviously, many services of function layer depend on cryptographic functions using either symmetric or asymmetric keys to encrypt/decrypt and sign data. Therefore, MASF integrated a *cryptography library* in its base service layer.

The *Key Management* service enables roles to administrate their own public/private key pairs and associated certificates for use in self-authentication (where the user authenticates himself/herself to other users/services) or data integrity and authentication services, using digital signatures. The authentication information includes both a sequence (chain) of X.509 certificates, and an associated private key, which usually is referenced as "alias".

To achieve security, the MASF framework supports flexible security policies to govern the interactions of agents with both other agents and with the available resources in the execution sites. This function is performed by *Policy Management* service in the base service layer. The definition and enforcement of appropriate security policies can only proceed after a precise identification of the principals, i.e., roles that can be authenticated.

Resource Usage Logging service fulfils the logging requirement mentioned in previous section. Although not specific to MASF, location service is sometimes used by MASF to identify the role.

4.4 MASF Workflow

4.4.1 Preparation: Certificate and Key Creation and Exchange

First of all, *keystore* is generated to store the keys and certificates. A keystore is a database of private keys and their associated X.509 certificate chains authenticating the corresponding public keys. keystore appears as a file and private keys are protected with a password. keystore can be easily generated by using security tool *keytool* or Java API provided by J2SDK 1.4.

In MANTRIP, all the network configuration and management tasks are fulfilled by mobile agents and all mobile agent classes are stored in JAR files that are digitally signed either by *jarsigner* or Java API. Whenever the network administrator, or the software on his behalf, wants to fulfill network management task using mobile agents, he has to get his signature verified firstly before he can get access to these mobile agents stored in a protected JAR file. The agent factory for network management is usually set up by network administrator, therefore it is signed by the administrator. If the mobile agent is obtained from other parties, the certificate and public key of this administrator should be pre-imported in the signed JAR file guided by the contract.

According to the network management requirement expressed in the XML-based policy, Policy Decision Point (PDP) module decides the traveling route of mobile agent. At least the first hop destination is needed as the second and latter hops of mobile agent can be decided at the end of execution according to the execution result. In this case, source and destination Agent Systems exchange their certificate in the first place. Generally, the exchange of certificate can be performed either statically based on familiarity set that is pre-created according to the cooperation relationship or dynamically depending on the execution result of mobile agent. The powerful Java API provided by J2SDK 1.4 enables the fulfillment of dynamic certificate exchange.

4.4.2 Security Environment Setup for Agent System

As discussed above, before a mobile agent is deployed across network, two sides of security mechanisms need to be employed,

i.e., agent supporting platform (i.e., agent system) and mobile agent.

Regarding to the agent system side, firstly the *security* property of agent system is switched on in order to enable the security service; and then the agent system needs to be provided with the location of private key, public key and certificate that will be used. In MANTRIP, Mobile agent is dynamically created and sent off from *Agency*. Precisely speaking, Agency is the supporting environment for mobile agents. There can be multiple key-pairs and certificates in one agent system as multiple agencies can coexist in one agent system, which enables a multiple security controls within one agent system.

Access control of agent system also needs to be enabled and corresponding policy file and policy entries are created, e.g., via *policytool*. The following information can specified in a policy entry: *URL location* where the code originates from, *alias name* from the keystore used to reference the signer whose private key was used to sign the code, an optional *Principals* entry indicating the list of principals that the code has to be executed as in order for the permission(s) to be granted, and finally one or more *permission* entries indicating which permissions are granted to the code from the source indicated by the URL location and alias name. The policy entries can also be changed dynamically using Java programming based on the SLA.

4.4.3 Mobile Agent Workflow

After all the key-pairs and certificates are available and the security environment has been set up in agent system, it is time for mobile agent to start working. After getting the classes of mobile agent, the administrator or the software on his behalf signs the mobile agents and attaches his certificate to the agent in order to show the initiator information, the first agent system etc. Administrator can also supplies agent via agency with necessary rights if no complex access control, i.e., security policy, is applied. Then mobile agents can move itself from one agent system to another in order to fulfill the network management work. The communication between two agent systems are SSL-enabled using the key-pairs already existing in each agent system agency.

When a mobile agent system, i.e., agency on its behalf in MANTRIP Mobile Agent system, receives a mobile agent from the communication network via ATP (Agent Transport Protocol), it decrypts it and tests the integrity of the data received by checking the signature that the sender has appended. After successfully passing the integrity check, the following step is authentication. The mobile agent system verifies signature and certificates attached to this mobile agent and further gets the information such as, who wrote the mobile agent, who sent it at the very beginning or in the intermediate locations. The information can be further used for authorization and access control latterly.

Once authenticated, MASF authorizes the agent, i.e., it gets rights attached to the mobile agent or determines rights based on the security policies defined in advance. Using security policy based access control is more flexible, though it might cause some performance deterioration.

Then mobile agent can be executed under access control so as to enforce the network management task. When the mobile agent has finished its work and wants to migrate to another location, the mobile agent system stops the execution of mobile agent and packs the agent with its current state, as it normally does. According to the network management task, rights adjusting module may be called at this moment to adjust the current right of mobile agent, e.g., give mobile agent more rights at its next location. Then, signer/encryptor module is called by Agency to sign the mobile agent to confirm the execution or any change of mobile agent. Encryption may be applied as well by this module. Finally, Agency opens a communication channel to the new Agency (or Place) and sends the agent. The channel might be a secure one enhanced by Secure Socket Layer (SSL). Further discussion about this is available in next two sub-sections.

4.5 Implementation Issues

Regarding to implementation, MASF uses X.509 certificates for authentication, which ascertain the role of the agent principal before authorizing any interaction with resources. The issue as to the management of certificates and other related administrative tasks has yet been integrated, but a commercial Public Key Infrastructure (PKI) provided by Entrust [10] expects to be used. Confidentiality is granted by encrypting/decrypting communications with SSL. Secure Socket Layer (SSL) provides a means for securing communication interchanges between agents. This protocol authenticates and encrypts TCP streams. There are a number of third party implementations available in Java, e.g., IAIK SSL. The integrity check can employ either MD5 or SHA1, which are fully provided in J2SDK1.4.0. Access control of agent action can be performed by using Java Access Control mechanism.

Furthermore, three tools in J2SDK1.4 can greatly ease the key and certificate preparation work: *keytool* is used to create public/private keys; to display, import, and export certificates; and to generate X.509 certificates. *jarsigner* signs JAR (Java Archive Format) files. *policytool* creates and modifies the external policy configuration files of a role such as service provider.

4.6 Trade-off between security and performance

Security mechanisms unavoidably cause performance deterioration. In order to get a better trade-off between security needs and required performance, the following guideline is adapted in our system: agents in trusted environments, i.e., intra-domain such as a private Intranet of a division, could directly access resources after the authorization check; whilst agents moving in un-trusted environments such as the network belongs to other service provider or network provider (or so called DMZ: De-Militarised Zone), i.e., inter-domain, generally have to pass the confidentiality and integrity check apart from authentication and authorization.

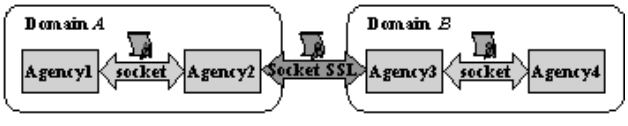

Figure 6: Intra-domain and Inter-domain Agent Communication and Migration

5. CASE STUDY: INTER-DOMAIN IP VPN CONFIGURATION SECURED BY MASF

Based on MASF architecture given above, this section presents a case study to evaluate this architecture, i.e., safe inter-domain IP VPN provisioning fulfilled by mobile agents that are enhanced by MASF, as shown in Figure 7. Inter-domain management is also a challenging research field in network management due to the more potential security problems. MASF can also provide, as a case study, a solution to the security problems occurring during inter-domain communication.

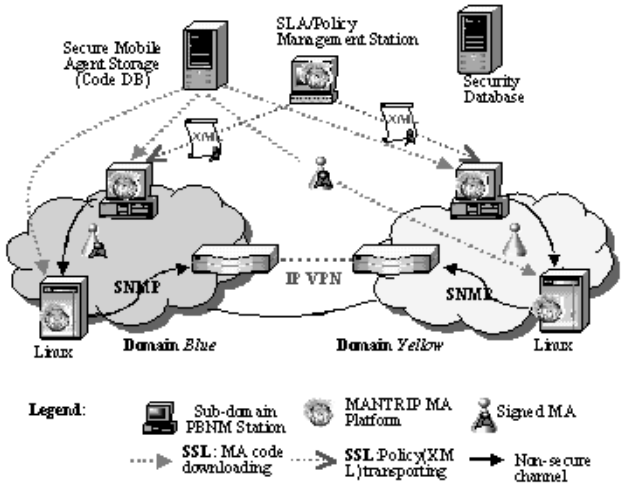

Figure 7: Inter-domain IP VPN Configuration

Network administrator uses SLA/Policy Management Station to manage the underlying network environment (including two domains with one Cisco router and one Linux machine next to Cisco router at each domain) by giving policies, which are further translated into XML files and transported to sub-domain PBNM station after digitally signed. The sub-domain PDP (Policy Decision Point) manager can download the proper PDP via SSL, which is in the form of digitally signed mobile agent, to make the policy decision. After this, the selected or/and generated policies are handed to PEP (Policy Enforcement Point) manager, which, also sitting on the sub-domain PBNM station, downloads the PEP codes, e.g., for new IP tunnel configuring, according to the requirement given in XML file. The PEP, also in the form of (digitally signed) mobile agent, moves itself to the Linux machine, on which it uses SNMP to configure the Cisco router so as to set up one end of IP VPN tunnel. Same procedure happens for the other end of the IP VPN tunnel, therefore set up the IP VPN tunnel.

6. CONCLUSIONS

Despite the advantages for network management offered by mobile agent technology, a wider application of this technology is greatly limited by the lack of a comprehensive security mechanism suitable to address the protection of both agents and hosts without introducing significant performance loss. In the large scaled network environment, the use of mobile agents to fulfil the management tasks may cause more serious security problems. This paper presents a practical solution for these kinds of security problems by introducing Mobile Agent Security Facility (MASF).

The mobile agent security facility (MASF) supports a wide span of security mechanisms: *authentication* permits to identify the communicating peers, both agent and Agency; *integrity* guarantees that agents and data have not been maliciously modified during transit; *authorization* recognizes whether an agent operation is permitted on a resource; *confidentiality* permits to protect entities from any exposure to malicious intrusions; *logging* of security relevant events supplies information for analysis of security actions and prevents the later repeating of the same problems. All these mechanisms are seamlessly integrated to secure the mobile agent based network management.

A practical network management application, inter-domain IP VPN configuration, integrated the MASF and provides a secure means for inter-domain network management. MASF is a very generic architecture and can be used in any mobile agent based network management applications.

Although a solution for trade-off between security and performance is slightly discussed, there is still a lot of work to be done about this. The integration of PKI into MASF is also the future work of this paper.

7. ACKNOWLEDGEMENTS

This paper describes work undertaken in the context of the EU IST project MANTRIP, a 2 years project during 2000-2002. The IST programme is partially funded by the Commission of the European Union.

8. REFERENCES

[1] N. Damianou, N. Dulay, E. Lupu, M Sloman. The Ponder Specification Language. Workshop on Policies for Distributed Systems and Networks (Policy2001), HP Labs Bristol, 29-31 Jan 2001.

[2] A. Bieszczad, T. White and B. Pagurek. Mobile Agents for Network Management. IEEE Communications Surveys, 1998

[3] A. Corradi, R. Montanari, C. Stefanelli. Security Issues in Mobile Agent Technology. Proceedings of the 7th IEEE Workshop on Future Trends of Distributed Computing Systems, (FTDCS'99), IEEE Computer Society Press, Cape Town, December 1999.

[4] Mobile agent system list: http://mole.informatik.uni-stuttgart.de/mal/mal.html

[5] Grasshopper http://www.grasshopper.de

[6] Objectspace Voyager http://www.objectspace.com

[7] MANTRIP Deliverable D2.2, "MANTRIP Functional Specifications and Validation Scenarios"

[8] D. Milojicic et al., "MASIF: The OMG mobile agent system interoperability facility" in Proc. 2nd Int. Workshop Mobile Agents: Springer-Verlag, Lecture Notes in Computer Science, vol. 1477, Sept. 1998.

[9] IAIK website. http://www.iaik.tu-graz.ac.at/

[10] Entrust Website. http://www.entrust.com

SINS: A Middleware for Autonomous Agents and Secure Code Mobility

[Extended Abstract]

Ramesh Bharadwaj
Center for High Assurance Computer Systems
Naval Research Laboratory
Washington, DC, 20375-5320 USA
ramesh@itd.nrl.navy.mil

1. INTRODUCTION

Building trusted applications is hard, especially in a distributed or mobile setting. Existing methods and tools are inadequate to deal with the multitude of challenges posed by distributed application development. The problem is exacerbated in a hostile environment such as the Internet where in addition applications are vulnerable to malicious attacks. It is widely acknowledged that intelligent software agents provide the right paradigm for developing agile, re-configurable, and efficient distributed applications. Distributed processing in general carries with it risks such as denial of service, Trojan horses, information leaks, and malicious code. Agent technology, by introducing autonomy and code mobility, may exacerbate some of these problems. In particular, a malicious agent could do serious damage to an unprotected host, and malicious hosts could damage agents or corrupt agent data.

SINS (Secure Infrastructure for Networked Systems) being developed at the Naval Research Laboratory is a middleware for secure agents intended to provide the required degree of trust for mobile agents, in addition to ensuring their compliance with a set of enforceable security policies. An infrastructure such as SINS is central to the successful deployment and transfer of distributed agent technology to Industry because security is a necessary prerequisite for distributed computing.

2. SECURITY REQUIREMENTS OF MOBILE AGENTS

The following requirements of secure mobile agents (see [5]) are addressed by SINS:

- The author and initiator of an agent must be authenticated.
- The integrity of an agent's code must be checked.
- Interpreters must ensure that agent privacy is maintained during data exchange.
- Interpreters must protect themselves against malicious agents.
- Interpreters must ensure that migrating agents are in a safe state.
- Agents must protect themselves from malicious hosts and interpreters.
- An initiator must be able to control an agent's flexibility; i.e., restrict or increase an agent's authorization in specific situations.
- Initiators must be able to control which interpreters are allowed to execute their agents.

3. SINS ARCHITECTURE

Figure ii shows the architecture of SINS. Agents are created in a special purpose synchronous programming language called SOL (Secure Operations Language) [ii]. A SOL application comprises a set of modules each of which runs on an Agent Interpreter (AI) which executes the module on a given host in compliance with a set of locally enforced security policies. A SOL application may run on one or more AIs, spanning multiple hosts across multiple administrative domains. Agents are created using a visual language vSOL (visual SOL) in an Agent Creation Environment (ACE), and are automatically translated into SOL. Agent Interpreters communicate among themselves using an inter-agent protocol [7], similar to SOAP/XML [8].

Currently, the idea of protecting an agent from a malicious host is still an area of ongoing research. Therefore, in our initial implementation of SINS, we assume a degree of trust among the hosts. This is reasonable, especially in a large organization such as the Department of Defense, where one may assume that other policing methods and techniques for intrusion detection are able to identify and isolate malicious hosts and eavesdroppers. We plan to address the more

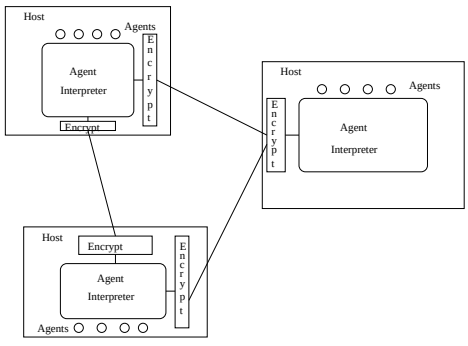

Figure 1: Architecture of SINS.

general problem of survivability and agent protection in our future work. Therefore, the current SINS implementation assumes the following:

- A host will run an agent interpreter to completion.
- All agent interpreters will run agents correctly.
- An agent interpreter will transfer agent data as requested.
- Agents' code and data cannot be kept private from hosts.
- Agent-to-agent communication cannot be kept private from hosts.
- Agents cannot carry secret keys.

4. TECHNICAL APPROACH

The SINS middleware and the associated Agent Creation Environment are designed to explicitly address requirements of security and high assurance described above, in addition to related problems of agent creation and deployment. Although security is our primary concern, we also address problems of efficiency, reconfigurability, and ease of agent creation and debugging. SINS addresses each of the security requirements as detailed below:

4.1 Authentication and Authorization

In SINS, code distribution is distinct from agent instantiation. Consequently, the issue of code tampering by possibly compromised hosts is addressed. Agent code resides in a SOL code repository and is digitally signed. Hosts retrieve the code either directly from the repository or from another host that has the most recent cached copy. The execution of a set of agents comprising a SOL application is initiated by a single host. SINS provides role-based access control and management and trust management to authenticate the agent initiator and to provide access to resources on a given host. The initiating host has fine-grained control over each agent in the application. This gives the initiator the ability to run the agents with restricted authority in most cases, but with greater authority in certain situations.

4.2 Integrity of Agent Code

All agents are programmed in SOL, a *verifiable* synchronous language [11]. As opposed to agents developed in a Turing-complete general purpose language such as Java, whose properties such as termination are undecidable, many properties of agents programmed in SOL, a more restricted language, are decidable. All analyzed and verified SOL agents are guaranteed to have no unbounded loops, violations of array index bounds, buffer overflows, etc. Therefore, as opposed to other agent systems where code integrity is based merely on trusting the author of the agent's code, integrity of agent code is ensured in SINS by proving (with mathematical certainty) the safety of SOL agents and their compliance with a host's local security policies. SINS includes a compliance checker (CC) [3] which establishes formally the compliance of the behavior exhibited by a SOL agent, or a set of agents, with the required security properties and the local security policies.

4.3 Agent Privacy

Agent Interpreters communicate among themselves using a secure protocol (SSL) which ensures agent privacy during data exchange and prevents casual intruders from eavesdropping on inter-agent message exchanges. Also, SINS implements a security architecture for monitoring and coordinating agents' activities.

4.4 Protection from Malicious Agents

Since SOL agents are composable and modular, CC can evaluate emergent behavior of agent communities, which is generally not possible in the absence of an agent aggregation framework. This capability enables early detection and prevention of an organized, cooperative attack in an environment in which each agent performs some action that falls beneath the threshold of most analysis techniques, but effects serious damage as a distributed attack. Currently these types of vulnerabilities have defied formal analysis.

4.5 Safe Agent Migration

Because a migrating agent can become malicious, we equip each agent in SINS with an appropriate state appraisal function which is used each time an interpreter starts an agent. The state appraisal function ensures that an agent will perform as required and that its data has not been tampered with in a malicious way. The static analysis tool CC can guarantee that the state appraisal function satisfies key safety properties and is in conformance with security policies being enforced at a given host.

4.6 Agent Protection from Malicious Hosts

As we mentioned before, SINS agents are currently not fully protected from a malicious host. However, the likelihood of agent corruption by a host is minimized by the introduction of a special class of agents called security agents [2] that police other agents such as application agents developed to support a given distributed application. Security agents protect a system against Information Operation (IO) attacks by implementing key security features such as encryption, authorization, policy enforcement, virus checking, and intrusion detection. Since security agents have more privileges than application agents, we need higher assurance during their development and deployment that they are safe and secure. This is achieved by the three-pronged approach of programming them in a safe language (SOL), by applying the compliance checker to establish formally their compliance with required safety properties, and by the security architecture for monitoring and coordinating agents' activities.

5. SECURITY AGENTS

In this section, we shall examine how *enforceable* safety and security policies [6] are expressed as *Security Agents* in SOL. The enforcement mechanism of SOL works by terminating all executions of an agent for which the safety or security policy being enforced no longer holds.

5.1 A Brief Introduction to SOL

A module is the unit of specification in SOL and comprises variable declarations, assumptions and guarantees, and definitions. The `assumptions` section typically includes assumptions about the environment of the agent. Execution aborts

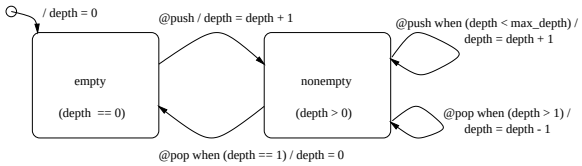

Figure 2: vSOL representation of `safestack`.

when any of these assumptions are violated by the environment. The required safety properties of an agent are specified in the `guarantees` section. The `definitions` section specifies updates to internal and controlled variables.

A variable definition is either a *one-state* or a *two-state* definition. A one-state definition, of the form $x = expr$ (where *expr* is an expression), defines the value of variable `x` in terms of the values of other variables *in the same state*. A two-state variable definition, of the form $x =$ `initially` *init* `then` *expr* (where *expr* is a two-state expression), requires the initial value of `x` to equal expression *init*; the value of `x` in each subsequent state is determined in terms of the values of variables in that state *as well as the previous state* (specified using operator `PREV` or by a `when` clause). A *conditional expression*, consisting of a sequence of branches "`[]` guard $\rightarrow$ expression", is introduced by the keyword "`if`" and enclosed in braces ("`{`" and "`}`"). A guard is a boolean expression. The semantics of the conditional expression `if` { $[]g_1 \rightarrow \text{expr}_1$ $[]g_2 \rightarrow \text{expr}_2 \ldots$ } is defined along the lines of Dijkstra's *guarded commands* [4] – in a given state, its value is equivalent to expression expr_i whose associated guard g_i is true. If more than one guard is true, the expression is nondeterministic. It is an error if none of the guards evaluates to `true`, and execution aborts. The *case expression* `case` *expr* { $[]v_1 \rightarrow \text{expr}_1$ $[]v_2 \rightarrow \text{expr}_2 \ldots$ } is equivalent to the conditional expression `if` { $[](expr == v_1) \rightarrow \text{expr}_1$ $[](expr == v_2) \rightarrow \text{expr}_2 \ldots$ }. The conditional expression and the case expression may optionally have an `otherwise` clause with the obvious meaning.

5.2 Safety Property Enforcement

We examine how SOL Security Agents are used to enforce safety properties. The example we shall use is a stack, which has the associated methods `push`, `pop`, and `top`. Informally, `push(x)` pushes the value of integer variable `x` on the stack and `pop()` pops the topmost value off the stack. The method `top()` returns the current value at the top of the stack. The stack can accommodate at most `max_depth` items. The safety policies we wish to enforce are: (i) No more than `max_depth` items are pushed on the stack. (ii) Invocations of methods `top` and `pop` are disallowed on an empty stack. Figure 3 shows a SOL module `safestack` which enforces these safety policies on *all* SOL modules which use the stack object (implemented in the embedding language). Figure 2 is the module `safestack` rendered in the visual syntax of SOL using ACE. Note that by deliberately omitting the `otherwise` clauses in the `if` statements, we abort the execution of an agent when none of the guards is `true` during execution. If this is too drastic, corrective action may be specified in an `otherwise` clause; for example, to ignore all `push` actions when the stack is full.

```
deterministic reactive module
  safestack(integer max_depth) {
interfaces
  void    push(integer x);
  void    pop();
  integer top();
internal variables
  {empty, nonempty} status;
  integer in [0:max_depth] depth;
guarantees
 INV1 =
  (status == empty) <=> (depth == 0);
definitions
  [status, depth] = initially [empty, 0] then
  case PREV(status) {
    [] empty ->
      if {
        [] @push -> [nonempty, PREV(depth) + 1]
        // other operations illegal!
      }
    [] nonempty ->
      if {
        [] @top ->
           [PREV(status), PREV(depth)]
        [] @pop when (depth > 1)  ->
           [nonempty, PREV(depth) - 1]
        [] @pop when (depth == 1) ->
           [empty, 0]
        [] @push when (depth<max_depth) ->
           [nonempty, PREV(depth) + 1]
        // @push when (depth == max_depth) illegal!
      }
  }; // end case
} // end module safestack
```

Figure 3: Security agent for safestack.

6. CONCLUSIONS

The goal of the NRL secure agents project is to develop enabling technology that will provide the necessary security infrastructure to deploy and protect time- and mission-critical applications on a distributed computing platform, especially in a hostile computing environment such as the Internet. Our intention is to create a robust and survivable information grid that will be capable of resisting threats and surviving attacks. One of the criteria on which this technology will be judged is that critical information is conveyed to principals in a manner that is secure, safe, timely, and reliable. No malicious agencies or other threats will be able to compromise the integrity or timeliness of delivery of this information.

7. REFERENCES

[1] R. Bharadwaj. SOL: A verifiable synchronous language for reactive systems. In *Proc. Synchr. Languages, Apps., and Programming, ETAPS 2002*, Grenoble, France, April 2002.

[2] R. Bharadwaj et al. An infrastructure for secure interoperability of agents. In *Proc. Sixth World Multiconference on Systemics, Cybernetics, and Informatics*, Orlando, Florida, July 2002.

[3] R. Bharadwaj and S. Sims. Salsa: Combining constraint solvers with BDDs for automatic invariant checking. In *Proc. 6^{th} TACAS, ETAPS 2000*, Berlin, March 2000.

[4] E. W. Dijkstra. *A Discipline of Programming*. Prentice-Hall, 1976.

[5] W. M. Farmer et al. Security for mobile agents: Issues and requirements. In *Proc. National Information Systems Security Conference*, October 1996.

[6] F. B. Schneider. Enforceable security policies. *ACM Trans. Infor. and System Security*, 3(1):30–50, February 2000.

[7] E. Tressler. Inter-agent protocol for distributed SOL processing. Technical Report To Appear, Naval Research Laboratory, Washington, DC, 2002.

[8] W3C. Simple Object Access Protocol (SOAP) 1.1. Technical Report W3C Note 08, The World Wide Web Consortium, May 2000.

Mobile Agent security using Proxy-agents and Trusted domains*

Nikola Mitrović
IIS Department, University of Zaragoza
Maria de Luna 3
50018 Zaragoza, Spain
mitrovic@prometeo.cps.unizar.es

Unai Arronategui Arribalzaga
IIS Department, University of Zaragoza
Maria de Luna 3
50018 Zaragoza, Spain
unai@posta.unizar.es

ABSTRACT

Commercial or wide-network deployment of Mobile Agent Systems is not possible without satisfying security architecture. In this paper we propose architecture for secure Mobile Agent Systems, using Trusted Domains and Proxy agents. Existing approaches are based on security services at the level of an agent system, library or specific objects. Our concept uses proxy agents to enable transparent security services both to security-aware mobile agents and legacy agents. Per-agent and domain-level security is provided. Proposed concept can be used with non-compatible environments and legacy systems.

Keywords

Mobile Agents, Security, Proxy Agents and Trusted Domain.

1. INTRODUCTION

Commercial or wide-network deployment of Mobile Agent Systems is not possible without satisfying security architecture. This paper outlines a design for a secure mobile agent architecture.

Mobile agents and mobile agent platforms are exposed to various security threats. Attacks on mobile agents and platforms usually come in two main forms: active and passive [8]. While passive attacks try to collect data without authorization (e.g., eavesdropping), active attacks try to modify system and cause different behavior of system. The trust [2] in mobile agent systems plays an important role. By establishing a trust relationship mobile agents can gain access to resources, perform specific actions or delegate their rights.

Existing solutions are focused on several approaches. Usually, the proposed solution is some kind of library [4] or service [1] that provide security mechanisms for mobile agents and mobile agent systems. Many of security problems are resolved by using Public Key Infrastructure (PKI) [9]. Some approaches uphold "smart objects" (that are self-aware) [10], or security agents that provide secure communication [6, 14, 15]. In addition, some authors as in [16], create specialized agents ("privacy guardians") that are meant to protect the data and communication of agents. Trust solutions [11, 12, 7, 4] are mainly focused on how to delegate and negotiate trust between systems or agents.

*This work has been supported by the spanish "Comisión Interministerial de Ciencia y Tecnologa" (CICYT), project TIC2001-1819.

In this paper we propose architecture for secure mobile agent systems, using trusted domains [3] and proxy agents [13]. We propose usage of a proxy agent paradigm for security services together with trusted domain and directory services for rights and authenticity distribution. Specialized *Security Proxy Agents* are used to provide security mechanisms to both mobile agent systems and mobile agents. This concept enables security-context for legacy systems, simplifies development of the agents and provides both domain-level and per-agent security. In addition, proposed architecture gives possibility of protecting the devices that does not have sufficient processing power (e.g. wireless devices).

This paper is organized as follows: in Section 2 we present proposed architecture. Sample scenarios are discussed in Section 3. In Section 4 we present conclusions and future work.

2. PROPOSED SECURITY ARCHITECTURE

Security is a delicate issue. As the system is more secure, it gets more difficult to build, more complex to maintain. Complex systems with more components have higher possibility of failure or breach; on the other side, too simple systems can be vulnerable.

2.1 Security Proxy Agents and Trusted Domains

Having this as an idea, we propose architecture that eliminates certain aspects of complexity. We introduce *security proxy agents* as facilitators of security services for mobile agents and mobile agent systems. Notion of proxy-agents is not new [13]. Many authors used proxy-agents as agents that help other agents to do something, or to do something on the behalf of other agent [13]. Security proxy agent is mobile agent that provides security services to both agents and/or agent systems. This agent contains extensible set of security and cryptographic mechanisms that can be used by agent systems or agents autonomously. In addition, these specialized agents contain set of automatic actions that are transparently performed upon agents and agent systems. Each mobile agent system have one *proxy factory* that creates and associates the agents with the security proxy agent created within factory. Also, system assigns one or more security proxy agents to guard the "entrance" to the system. These security proxy agents check all incoming and outgoing agents in order to apply adequate trust policy and security checks. In addition, security proxy agents can be extended

to support special requirements of some systems.

Our architecture relies on the concept of Trusted Domains. Fig. 1 shows proposed security architecture organized organized as trusted domains. We can see that every domain has one or more places (agent systems) that deal with security.

One domain has responsibility to authenticate agents and agent systems, and to apply appropriate trust policy. Once in the domain, the agent can travel freely without any further security checks, since it is considered trusted. Local access restrictions are applied (the user may not be willing to share some of the resources with others). Exceptionally, additional tests can be forced, and agents or agent system can require additional services from security proxy agent.

If the agent is member of more than one domain, malicious agents could enter from another domain. In this case, trust relationship must be established between domains, and such agent system should have installed proxy factory in order to check and apply adequate trust policy for incoming agents from different domain.

2.2 Security Proxy Agents' operations

Security proxy agents perform several transparent functions. By checking the agents' signature [15] and by encrypting it, they provide secure transport and identification of the agent. Also, using agents' signature or the signatures of the agent systems' modules, the alteration of agent or system code is automatically detected. Similar actions are done on the security proxy agents to ensure authenticity and non-alteration. Other security or cryptography services that agents can request such as state appraisal [5] or transaction logging [8] are provided by security proxy agents upon request. Number of security functions supported by security proxy agents is extensible as some systems may provide or require additional security mechanisms. Agent systems also enjoy transparent trust and security verification of the incoming and outgoing agents, and if needed, can request additional services from security proxy agent.

Legacy agents, or agents that are not aware of security context will enjoy transparent services of security proxy agents. This leads to faster and easier development of mobile agents that do not require some specialized levels of security.

This architecture is built with public systems in mind. It is not focused on how to solve some of the specific attacks on agent or host, but on the architecture of a system that will include the features of the known solutions, and expose them in more transparent and efficient manner to the system. Proposed architecture can be applied on the public systems such as Internet Service Providers (ISPs), or the systems that require some levels of security, such as Local Area Networks (LANs).

3. SAMPLE SCENARIOS

In this Section, we present common scenarios that occur within this security model.

Let us suppose that one agent from the home-system A wants to travel to the remote-system B. From the Figure 1 we can see that Home-system A belongs to the domain D1 and remote-system B to the domain D2.

As the agent (from the system A) is in its home-platform, the agent will move to its domain controller (DC1). The proxy factory service located at domain controller will create security proxy agent that will be assigned to our traveller agent, and will equip it with agent's credentials. The agent itself do not have to be aware of this process. Security proxy agent will perform the signing and optionally enveloping of the agent-traveller. Then, agent-traveller and security proxy agent will travel to the domain controller (DC2) of the destination domain D2. Upon arrival at domain controller DC2, security proxy agent will check the alteration of the agent-traveller and itself. If there is no alteration detected, security proxy agent of the agent-traveller and security proxy agent of the domain controller DC2 will negotiate possibilities of cooperation.

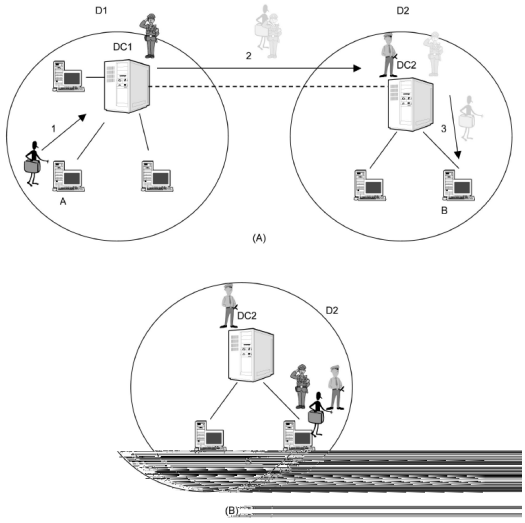

Figure 1: Sample scenario – agent trajectory.

If the cooperation is possible and the security requirements are met, the traveller-agent will be prepared for execution (e.g., decrypted). Then it will continue its journey to the remote-host B. Once in the domain, as described in Section 2, the traveller-agent can run without limitation and within his environment, without any needs to be decrypted, checked or bounded in any way, except for the current host access privileges (sandbox). The security proxy agent will remain at the domain's entrance (DC2), waiting for agent to finish its journey at domain. This can be suitable for devices that do not have sufficient processing power, such as wireless devices. If the agents are trusted by the domain, it can travel to any mobile device within the domain without having to perform security or trust negotiations.

In case that agent-traveller needs some extra security operations, like transaction logging, encryption or signing, the traveller-agent will call its own security proxy agent to assist him (see Figure 1(B)). Similar behavior will occur if the agents from systems B want to use some security operations. In this case, domain D2's controller (DC2) will use its proxy factory to create specialized security proxy agent that will assist mobile agents that are "owned" by domain D2; in this case, for the agents from system B.

Upon completion of its journey on the domain D2, traveller-agent will meet once again with its security proxy agent, and upon authenticity and alteration check, the traveller-agent will be returned to "safe to transport" mode, and the agents will continue their journey to another plat-

form. Similar behavior will be exercised on the mobile agent systems.

We examined normal operation of the system. However, we expect that the agents and systems are exposed to attacks. Here, we will discuss some of the situations when agents and agent-systems are malicious.

If the malicious agent is launched from the very domain, this agent can do harm only to a limited number of hosts. This kind of agents will be detected first time they meet with the domain controller, or a security proxy agent. The malicious agent will be detected, and the agent origin will be tracked. Alternatively, every agent that is launched from one system can be forced to pass trough domain controller, where it could be checked and approved. However, our approach is based on the idea that one domain is internally safe, so this kind of a measure is considered as unnecessary. Also, if some of the systems detect an malicious agent or vice versa, these systems or agents can be easily tracked and eradicated from the domain. The possibility of malicious agents performing unauthorized actions is also limited by the sandbox mechanism that is used on every platform, as described earlier.

Similar situations will occur in the malicious mobile agent system scenario. Tampered agents will be detected as soon as they arrive on domain controller, or perform an operation with security proxy agents. The malicious host will be detected and eradicated.

Legacy agents that are not aware of the proxy agents will be transparently processed by proxy agents. Agent systems that are not familiar with security proxy agents will treat them just as ordinary agents. Therefore, some levels of security will be conserved for legacy agents and systems.

4. CONCLUSIONS AND FUTURE WORK

In this paper, we have presented a security architecture that uses security proxy agents and distributes security over trusted domains. The main features of this approach are:

- This architecture uses known solutions to security problems (known mechanisms).
- Proxy agents are used to provide security functions to both agents and platforms.
- Security proxy agents can be extended to support additional (and specific) features.
- Security is distributed over trusted domains, which facilitates management and trust tasks.
- This architecture supports legacy agents and systems.
- This concept can enable security context for devices that cannot perform security computation (e.g, mobile devices).
- Security proxy agents act transparently (easier agent development).

Our future work will be focused on implementing proposed concept and in implementing autonomic logic to the prototype. Also, as a continuation of this work, some form of distribution and caching of certificates for mobile agents should be investigated.

5. REFERENCES

[1] J. Bacon, K. Moody, and W. Yao. Access control and trust in the use of widely distributed services. *Middleware*, 2001.

[2] M. Blaze, J. Feigenbaum, and A. D. Keromytis. The role of trust management in distributed systems security. In *Secure Internet Programming*, pages 185–210, 1999.

[3] B. Crispo. How to build evidence in a public-key infrastructure for multi-domain environments. In *Security Protocols Workshop*, pages 53–65, 1997.

[4] Q. He and K. Sycara. Towards a secure agent society. *ACM AA'98 Workshop on Deception, Fraud and Trust in Agent Societies*, 1998.

[5] F. Hohl. A framework to protect mobile agents by using reference states. In *International Conference on Distributed Computing Systems*, pages 410–417, 2000.

[6] F. Hohl and K. Rothermel. A protocol preventing blackbox tests of mobile agents. In *ITG/VDE Fachtagung Kommunikation in Verteilten Systemen (KiVS'99)*. Springer-Verlag, Berlin Germany, 1999.

[7] Y. J. Hu. Some thoughts on agent trust and delegations. *The Fifth International Conference on Autonomous Agents*, May 28-June 1 2001.

[8] W. Jansen and T. Karygiannis. Nist special publication 800-19 - mobile agent security. *NIST*, 2000.

[9] U. Maurer. Modelling a public-key infrastructure. In *ESORICS: European Symposium on Research in Computer Security*. LNCS, Springer-Verlag, 1996.

[10] C. Meadows. Detecting attacks on mobile agents. *Foundations for Secure Mobile Code Workshop*, pages 64–65, March 1997.

[11] L. Moreau. A Fault-Tolerant Directory Service for Mobile Agents based on Forwarding Pointers. In *The 17th ACM Symposium on Applied Computing (SAC'2002) — Track on Agents, Interactions, Mobility and Systems*, Madrid, Spain, Mar. 2002.

[12] C. Ono, D. Kanetomo, K. Kim, B. C. Paulson, M. Cutkosky, and C. J. Petrie. Trust-based facilitator for e-partnerships. *Fifth International Conference on Autonomous Agents (AGENTS'01), AAAI*, pages 108–109, May 2001.

[13] S. Papastavrou, G. Samaras, and E. Pitoura. Mobile agents for WWW distributed database access. In *ICDE*, pages 228–237, 1999.

[14] V. Roth. Mutual protection of co-operating agents. In *Secure Internet Programming 1999*, pages 275–285, 1999.

[15] V. Roth and V. Conan. Encrypting java archives and its application to mobile agent security. In *AgentLink 2001*, 2001.

[16] R. Serban and R. van de Riet. Enforcing policies with privacy guardians. *SEMAS*, 2001.

MARISM-A: An Architecture for Mobile Agents with Recursive Itinerary and Secure Migration

Sergi Robles
Dept. of Computer Science
Universitat Autònoma de Barcelona
08193 Bellaterra, Spain
Sergi.Robles@uab.es

Joan Mir
Dept. of Computer Science
Universitat Autònoma de Barcelona
08193 Bellaterra, Spain
Joan.Mir@uab.es

Joan Borrell
Dept. of Computer Science
Universitat Autònoma de Barcelona
08193 Bellaterra, Spain
Joan.Borrell@uab.es

ABSTRACT

MARISM-A is a secure mobile agent platform providing complex security mechanisms to protect migration, confidentiality and integrity of agents.Agent itinerary, data and code are protected through a new model of mobile agents, involving new agent architectures. Moreover, MARISM-A accepts user defined agent architectures in run time. Main aims in the designing of this platform have been security and extendability. Mobile and nomadic computing can be implemented with MARISM-A, even though it has been specially designed to develop sea-of-data applications.

Keywords

Mobile Agent Security, Applications, Implementations.

1. INTRODUCTION

During the last times, multi agent systems have proliferated to fill the demand for applications based on mobile agent technology. There is no doubt that mobile agents make feasible the implementation of a wider variety of applications than using classic paradigms. The price of this is a new branch of open issues concerning security.

The MARISM-A platform [1] benefits from all intrinsic advantages of mobile agent technology, such as execution of code near the data or the possibility for the owner to be offline while remote resources are accessed. Furthermore, MARISM-A adds some others that make it especially suitable for sea-of-data applications. Some of the new features include secure migration, secure agent communication and the coexistence of different agent architectures (including our new recursive agents) in the same heterogeneous framework. MARISM-A converges several mobile agent security techniques we have developed into a flexible scheme. All these traits confer to the platform the capability of easily creating new applications to solve difficult problems hardly faceable without a framework like MARISM-A.

From the new architectures introduced in MARISM-A a new paradigm for mobile agent programming has emerged. This new programming model is oriented to agencies and makes complex applications with itinerant agents be very easy to be coded. Furthermore, the new agent architectures we have designed allow to independently keep confidentiality on all agent components, and prevents attacks against agent integrity. This is indeed one of the more novel features of our platform. This paradigm extends the classical object oriented programming to a new location oriented programming in a highly intuitive fashion. Our new programming paradigm makes possible a new type of developing environment. In this IDE, the programmer graphically defines all components and destinations of agents.

To date, MARISM-A is not yet completely finished. To date, we have developed some agent architectures and some other complex schemes are planned to be added soon, such as resource access control achieved through agent certificates over a SPKI. MARISM-A observes most of the FIPA specifications [2] Mobility mechanisms, not very concerned in the FIPA specifications, have been implemented taking into account the MASIF mobility standard [3].

The rest of the paper is structured as follows. Section 2 provides a description of the MARISM-A platform. Different implemented agent architectures are described in section 3. In section 4 some implementation details are shown. Finally, section 5 is devoted to the conclusions.

2. PLATFORM DESCRIPTION

MARISM-A uses the traditional concepts of agent platforms. A Mobile Agent System consists of several internetworked agencies. The agency is the basic environment for the execution of agents. An agency consists of a directory service, an agent manager and some internal facilities, such as a message transport service.

Agents are software units executing in the agencies on behalf of their owners. Parts of the agent, security mechanisms to protect it from attacks and other agent features are defined by its architecture. We have designed some architectures for MARISM-A including static and mobile agents, explicit and implicit itineraries, and with different types of explicit itineraries.

2.1 Mobility

We have designed our own migration protocols. We have tried to observe the MASIF standard as much as possible. Our mobility solution uses FIPA Agent Communication Language (ACL) for migration negotiation. Agent requirements about execution and the beneath transport protocol are dealt with at this level. Because our migration protocol is based on ACL the negotiation can be established between agencies observing FIPA specifications. If an agency has not the execution capabilities demanded by the agent, or a common transport protocol, it will refuse the migration request. This flexibility makes MARISM-A able to inter-operate with

other FIPA platforms.

If the agent destination agency accepts the migration request the second level of the migration protocol will be started. The destination agency reconstructs the agent in the remote side and resumes its execution. The whole migration protocol is shown in figure 1.

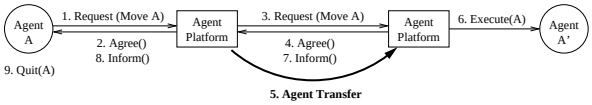

Figure 1: Migration Protocol

Step 5 of figure 1 represents the agent transfer process. To carry out this process we have used the reflection mechanisms of the Java language (getting information about classes and interact with themselves), and its capabilities for network programming. Basic steps of the process are serialisation, codification and transmission. The remote agency carries out the inverse process when receives the agent.

MARISM-A agencies have four components: a code server and client and a data server and client. Agent classes are stored on a class cache memory, so they are downloaded only once. When the agency receives a migration request it instantiates a data and a code clients to obtain the data and classes from the source agency. These clients use a special class loader designed for MARISM-A (*AgentClassLoader*), which is a subclass of the default Java class loader. This MARISM-A loader downloads remote classes, while the default Java class loader is still used to load local classes. When both code and data are obtained, the agent is reconstructed and its execution is resumed.

2.2 Security Infrastructure

MARISM-A is based on two main pillars: security and extendability. Some basic security requirements, such as confidentiality, authenticity and non-repudiation in communications between agents (within the same agency) are provided by the agency. This is easy to achieve, since the agency controls the execution of all agents and provides the inter-agent communication service.

In MARISM-A we are more interested in security aspects of migration (secret, authentic and non-repudiable migration), and protection of all components of agents. In order to use some cryptographic mechanisms, all elements in MARISM-A share a Public Key Infrastructure (PKI).

Migration security is achieved by using SSL beneath our migration protocol. This is enough to assure secrecy and authentication in the migration process. To avoid the chance of repudiation after migration we use a simple protocol which finishes with a receipt (or non-falsifiable proof of reception).

Protection of agents is achieved through specific agent architectures which, at the same time, provide extendability to the platform. These architectures are analysed below.

3. AGENT ARCHITECTURES

One of the novel aspects introduced in our platform is the flexibility of the agent architecture. Instead of focusing on a specific type of agent, we have created different agent architectures. Some security mechanisms are applicable only for certain types of agents. Even mobility is a feature of only some agent architectures. Moreover, our design allows to have several types of agents coexisting at the same time in a heterogeneous environment.

Most bibliographic references on agents do not make a clear distinction between different parts of an agent. Some of them suggest the need of considering some internal parts independent, especially for mobile agents. This is the case of agent data in [4], of agent code in [5], or agent itinerary in [6] and [7]. Independence of these parts plays an important role for some agent protection mechanisms. In MARISM-A, the architecture of the agent follows an adaptable model that determines the different parts in which an agent is divided and the combination of security, integrity, or other mechanisms included in it.

All mobile agent architectures share some basic aspects, such as the differentiation of internal parts and migration mechanisms. A mobile agent consists of code, data, state, and an implicit or explicit itinerary. Code is the set of instructions describing the execution of the agent. Data is an information storage area that can be used by the agent at any moment for reading and writing and goes with it all the time. Results of executions are stored separately in this area. State is like the data part of the agent but reserved to store the agent information related with its state. Explicit itinerary is a structure containing all agencies that are going to be visited by the agent on its life cycle [8]. In MARISM-A, itineraries are constituted by several basic structures that can be combined to build complex itineraries. These itineraries allows the use of a wide range of mechanisms to protect confidentiality and integrity.

The implementation of the agent management methods is included in the agent itself. This allows, for example, to delegate specific mobility functions to the agent, freeing the agency from this responsibility. Using this implementation, MARISM-A becomes more generic, simultaneously supporting a wider range of agent architectures.

3.1 Results

If confidentiality and integrity for the results is required, MARISM-A provides a protection mechanism based on hash chains. Results are firstly encrypted using agent's owner cryptographic information. Only the owner of an agent will be able to read its results. Once the result has been written, a hash of the Result and previous hashed information is calculated, signed and written next. This hash has information about the identity of next agency in the itinerary, therefore no agency can neither modify the result area nor remove some result. Each agency verifies during immigration that all hashes in the Results Data are correct. This is the format of the Results Data area:

$$ResultsData = E_o(nil, Id_1), S_o(H(E_o(nil, Id_1))),$$
$$E_o(R_1, Id_2), S_1(H(E_o(R_1, Id_2))), ...$$
$$E_o(R_n, Id_o), S_n(H(E_o(R_n, Id_o)))$$

where $E_o(m)$ is an encryption of m that can only be decrypted by the owner of the agent (o); $S_i(m)$ is a signature made by i; R_i is the result of agency i; Id_i is the identity of the next agency, i; and $H()$ is a hash function.

3.2 Mobile with explicit nested itinerary

This is only one of the architectures off-the-shelf in MARISM-A, and shows the main concepts of designing.

In this architecture, agent code is split into several pieces: there is a main code that will be executed in all agencies

(Common Code), and as many code fragments as agencies are in the itinerary, each one to be executed in a particular agency (Local Code). This feature makes MARISM-A very useful in some types of application where execution is context dependent. The agent changes after a migration. This dynamic aspect of the agent allows several security mechanisms to be applied. This is the recursive architecture:

Agent_i = CtrlCode, State, CommCode, GblData, Itinerary_i
Itinerary_i = LclCode_i, Data_i, Agencies_i, Itinerary_{i+1} | *Nil*

Agencies_i is the agency (or agencies, depending on the type of itinerary) the agent is going to visit (migrate) next. The agent that is sent to the next hop of the itinerary (Agent_{i+1}) has the same structure. The last host is identified because of a *Nil* next agent. CommonCode is executed by all agencies when the agent immigrates and before the specific LocalCode. Programming is simplified by using this common code to include the non agency dependant code only once. The control code in the agent deals with the functions of agent management, in this case extracting the relevant parts of the agent.

This architecture allows to implement several protection mechanisms, in addition to the mechanism presented in the implicit itinerary architecture to protect results data. Furthermore, code and itinerary are also protectable for agents using this architecture. The idea is to take advantage of the nested structure of the agent and make available to an agency (possible to decrypt) only the portion of the agent needed for the local execution. This is the structure of the agent when using data, code and itinerary protection:

Agent_i = CtrlCode, State, CommCode, GblData
Itinerary_i, $S_o(H(ControlCode, CommonCode))$
$\text{Itinerary}_i = E_i(LclCode_i, Data_i, Agencies_i,$
$E_{i+1}(Itinerary_{i+1}))$ | *Nil*

where $E_i()$ is an encryption using public key of agency i.

4. IMPLEMENTATION

The implementation of MARISM-A has been done in Java and consists of three different parts: agencies, agents, and an set of tools for developing agents, including an IDE.

A hierarchy of classes represents different agent architectures. Although MARISM-A provides some of them, this hierarchy may be extended with new user-defined architectures, as long as root classes are kept. Each class implements specific features of the architecture. Classes implementing basic security mechanisms for each architecture have also been included with MARISM-A basic classes.

In the IDE, the programmer graphically defines the itinerary of the agent and adds the code to be executed in the nodes representing agencies. To date, we have developed a first version of this IDE. Figure 2 shows a screenshot of the Itinerary Creation Tool, part of the IDE.

5. CONCLUSIONS

In this paper we have presented a novel secure platform for mobile agents. MARISM-A implements complex mechanisms to protect data, code, itinerary and migration of mobile agents. One of the most novel features is the flexibility and extendability of the platform. MARISM-A provides some off-the-shelf agent architectures, implementing several security solutions. Programmers can also add their own at any time, therefore extending the platform.

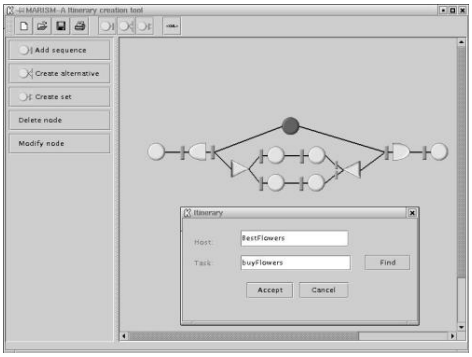

Figure 2: Itinerary Creation Tool

The new conception of migration mechanisms has lead us to a new programming paradigm based on location. We are developing an IDE that allows an agency-oriented programming of mobile agent applications.

We have not implemented solution for all attacks, but an extendable platform in which many solutions can be implemented. To date, we are implementing a new resource access control mechanisms based on SPKI.

6. ACKNOWLEDGEMENTS

This work has been partially funded by the Spanish Gov. Commission CICYT (grant TIC2000-0232-P4), and Catalan Gov. Department DURSI (grant 2001SGR 00219). Many thanks to Camper S.A. for their financial support.

7. REFERENCES

[1] CCD Research Group: MARISM-A, An Architecture for Mobile Agents with Recursive Itinerary and Secure Migration. `http://www.marism-a.org` (2002)

[2] Foundation for Intelligent Physical Agents. FIPA Specifications (2000) `http://www.fipa.org`

[3] Milojicic, D., Breugst, M., Busse, I., Campbell, J., Covaci, S., et al.: MASIF: The OMG Mobile Agent System Interoperability Facility. Proceedings of the Second Intl. Workshop on Mobile Agents. (1998) 50–67

[4] Straßer, M., Rothermel, K.: Reliability Concepts for Mobile Agents. International Journal of Cooperative Information Systems (IJCIS) **7:4** (1998) 355–382

[5] Baumann, J., Hohl, F., K. Rothermel, Straßer, M.: Mole - Concepts of a Mobile Agent System. Special Issue on Distributed World Wide Web Processing: Appl. and Techniques of Web Agents. **1:3** (1998) 123–137

[6] Borrell, J., Robles, S., Serra, J., Riera, A.: Securing the Itinerary of Mobile Agents through a Non-Repudiation Protocol. IEEE International Carnahan Conference on Security Technology (1999) 461–464

[7] Karjoth, G., Asokan, N., Gülcü, C.: Protecting the Computation of Free-Roaming Agents. Proceedings of the Second International Workshop on Mobile Agents. LNCS 1477, Springer-Verlag (1998) 194–207

[8] Mir, J., Borrell, J.: Protecting General Flexible Itineraries of Mobile Agents. Proceedings of ICISC 2001. LNCS 2288, Springer Verlag (2002)

[9] Straßer, M., Rothermel, K., Maifer, C.: Providing Reliable Agents for Electronic Commerce. Proceedings of the International IFIP/GI Working Conference. LNCS 1402, Springer-Verlag (1998) 241–253

Coalition Signature Scheme in Multi-agent Systems

Yiming Ye
IBM TJ Watson Research Center
PO Box 704
Yorktown Heights, NY 10598, USA
yiming@us.ibm.com

Xun Yi
School of EEE
Nanyang Tech. Univ.
Singapore 639798
exyi@ntu.edu.sg

Santhosh Kumaran
IBM TJ Watson Research Center
PO Box 218
Yorktown Heights, NY 10598, USA
sbk@us.ibm.com

ABSTRACT
The Internet will never reach its full potential as an electronic marketplace unless e-commerce agents, or proactive Web Programs, are used to automatically or semi-automatically perform e-commerce tasks. The dynamic and heterogeneous interactions among them and the automations brought by them will create tremendous opportunities as well as introduce the risk and vulnerability for e-commerce. In this paper, we study the security issues with respect to an important multi-agent interaction – the multi-agent coalition formation where e-commerce agents dynamically forming coalitions to exploit the benefits of grouping. We propose a coalition signature mechanism to provide a means for a coalition to bind its identity to a piece of information during its interactions within an e-commerce marketplace.

Keywords
E-commerce agent, coalition security, coalition signature.

1. INTRODUCTION
The pervasive connectivity of the Internet and the powerful architecture of World Wide Web are changing many market conventions and are creating tremendous opportunity for conducting business on Internet. Electronic commerce activities, such as on-line exchange of information services and products etc are bringing business to a whole new level of productivity and profitability. In parallel with the emergence of electronic commerce, there have been interesting developments in the area of intelligent software agents, or software entities that are capable of independent actions in open, unpredictable environments. The Internet will never reach its full potential as an electronic marketplace unless e-commerce agents, or proactive Web Programs, are used to automatically or semi-automatically perform e-commerce tasks such as negotiation, bidding, auction, transaction, and matchmaking etc. As e-commerce agent technology becomes more mature and standardized, we may envision that tens of thousands of e-commerce agents will be seamlessly embedded in everywhere of the Web. The dynamic and heterogeneous interactions among them and the automations brought by them will dramatically reduce certain types of frictional costs and time incurred in the exchange of commodities. However, before we fully enjoy the benefits brought by e-commerce agents, we must realize that the risk and vulnerability is also imminent: the ubiquitous existence of various software agents designed by all kinds of people can work, interact, and also *attack* at any time from anywhere of the "wild web", where the "distance" among them has collapsed to near zero and the transactions can be done instantly. The electronic linking, either wired or wireless, among various agents has made security an issue that must be woven into any agent-based service environment, especially when digital agents migrating through wireless or wired linkage from one network computer or device to another and sensing, executing, transacting, and interacting along the way. The security issues, we believe, that related to agents include but not limited to the following: keeping data or preference of an agent secret from other agents except those who are authorized to access; ensuring data that belong to an agent not been altered by unauthorized agents; identifying and authenticating agents during agent interaction; verifying the source of data received by an agent; binding information to an agent; authorization during agents interaction; validation during agent interaction; access control in agent community; agent certification; acknowledging the receiving of data or services by agents; avoiding Internet worms floods; the reputation and trust in mobile agent environment; and agent security against malicious hosts in the network, etc.

It is important to study the security issues for a single agent. However, it is even more important to study the security issues with respect to a group of interacting e-commerce agents. Because it is the sheer interconnectedness and dynamic interactions among these agents that will make the future web a virtual dynamic marketplace, and at the same time, make the security issues one of the top issues to be addressed. In this paper, we study the security issues with respect to a very important multi-agent activity: the dynamic multi-agent coalition.

The remainder of this paper is organized as follows. The next section details the issue of coalition security. Section 3 proposes the coalition signature scheme for e-commerce multi-agent systems. Section 4 briefly concludes the paper.

2. COALITION SECURITY
Cooperation and sharing resource by creating coalitions of agents are an important way for autonomous agents to execute tasks and to maximize payoff. For example, e-commerce agents that represent self-interested real world parties such as buyers or sellers may explore the benefits of grouping by forming coalitions. Coalition formation has been addressed by researchers from both the game theory community and the multi-agent community. Game theory emphasizes the issues of N-person games formation under different settings and the distribution of the benefits among players, without providing algorithms that agents can use to form coalitions [2]. It concentrates on the stability and fairness issues for given coalitions. Multi-agent research emphasizes the special properties of a multi-agent environment and considers the effects

of communication costs and limited computation time on the coalition formation process [4][5].

Here we study a very important issue that has not been addressed before – the issue of coalition security. The advancement of technologies such as EDI, KQML, FIPA, bluetooth, Semantic Web, Peer-to-Peer, SOAP, Concordia, Voyager, Odyssey, Telescript, Java, and Servelet etc. will soon made the dynamic and heterogeneous interactions between tens of thousands of e-commerce agents a reality [3][4]. Under this environment, it will be a desired behavior for different agents to form coalitions for their own benefits. When coalitions are formed, securities will be an issue for agents belonging to different coalitions. Thus, how to define signatures for different coalitions will be an issue that must be addressed.

In order to make our discussions easier, we define some concepts here. Suppose that there are totally n agents: $A = \{a_1, \cdots, a_n\}$. Here A is the set of the agents. The index for agent a_i is i. A coalition $C = \{a_{i_1}, \cdots, a_{i_m}\}$ is a subset of A such that $\forall i_j, a_{i_j} \in A$, or in other words, $C \subseteq A$. If agent a is alone, we assume that it forms a unit coalition $\{a\}$. A coalition structure CS at time τ is the set of all the coalitions formed by agents in A, $CS(\tau) = \{C_1, ..., C_\alpha\}$. Where α is the number of coalitions at time τ, $\alpha = |CS(\tau)|$. C_i $(1 \le i \le \alpha)$ is a coalition formed by agents of A, $C_i \subseteq A$. The coalitions discussed in our paper are dynamic. An agent might leave its current coalition and either becomes a unit coalition or join another coalition, or an agent that belongs to a unit coalition might join another coalition.

There are three basic kinds of coalition structures. The first kind is non-overlapping coalition structure, in the sense that there is no agent that belongs to two different coalitions. This is the coalition structure in the traditional sense - partitioning the set of agents into exhaustive and disjoint coalitions. For example, suppose that $A = \{a_1, a_2, a_3\}$ is the set of agents, then $\{\{a_1\}, \{a_2, a_3\}\}$ is a non-overlapping coalition structure. A lot of research in coalition formations is related to non-overlapping coalition structure. The second kind is overlapping coalition structure, in the sense that an agent can appear in different coalitions, but no coalitions can contain another coalition. For example, $\{\{a_1, a_2\}, \{a_1, a_3\}\}$ is an overlapping coalition structure. The third kind is nested coalition structure, in the sense that partitioned coalitions can contain each other. For example, $\{\{a_1\}, \{a_1, a_2\}, \{a_3\}\}$ is a nested coalition structure. Sometimes, a coalition structure might be a combination of the above-mentioned kinds of coalition structures.

Here we present a coalition signature mechanism that can be used by any kinds of dynamically changing coalition structure $CS(\tau)$. A coalition signature mechanism, we believe, is fundamental in authentication, authorization, and non-repudiation for coalitions in an e-commerce multi-agent system. The purpose is to provide a means to bind identity of a multi-agent coalition to an agreement reached by the multi-agent coalition. The signature scheme for a given coalition C is based on identities of all members of C. A coalition signature $Sig_C(M)$ on message *M* is just some bits that reflect the structure of the coalition. Only a coalition itself can generate its own coalition signatures on messages. Other coalitions or parties cannot forge any coalition signature of the given coalition. The authenticity of a coalition signature can be verified by any parties. The following section presents details of our approach.

3. COALITION SIGNATURE SCHEME

The proposed coalition signature scheme is built on Guillou-Quisqater signature scheme [1] and comprised of certificate issuing, coalition certifying, coalition signing and coalition signature verifying as follows.

3.1 Certificate Issuing

Based on RSA, a Certification Authority (CA) randomly chooses two distinct big primes p and q and computes $n=p\times q$ and $\Phi(n) = (p-1)(q-1)$. Then CA randomly chooses its public key *e* such that $\gcd(e, \Phi(n)) = 1$ and $1 < e < \Phi(n)$ and computes its secret key d such that $e \times d = 1 (\mathrm{mod}\, \Phi(n))$. Finally, CA distributes (*e, n*) to all participants, but keep (*d, p, q*) secret. The following is the process for *CA* to issue a secret certificate to an individual agent a_{i_j}.

Step 1: Computer the hashed value h_{i_j} of the identity ID_{i_j} of agent a_{i_j}, i.e., $h_{i_j} = \mathrm{H}(ID_{i_j})$, where H is an one-way hash function that hashes an arbitrary length message into 160-bit message, such as Secure Hash Standard (SHS).

Step 2: Generate the signature s_{i_j} for agent a_{i_j} by transforming the hashed value h_{i_j} to

$$s_{i_j} = h_{i_j}^{-d} \ (\mathrm{mod}\, n).$$

Step 3. Issue the secret certificate (ID_{i_j}, s_{i_j}) to a_{i_j} through a secure channel. Here ID_{i_j} is the identity of a_{i_j}, and s_{i_j} is the secret key of a_{i_j} which is known only to the agent a_{i_j} besides *CA*.

3.2 Coalition Certifying

Suppose that at time T_C, a coalition $C = \{a_{i_1}, \cdots, a_{i_k}\}$ is formed. The representative of C is a_{i_1} and the identity set of C is $\Omega_C = \{ID_{i_1}, \cdots, ID_{i_k}\}$. CA certifies the coalition C by

signing $I_C = \{\Omega_C = \{ID_{i_1}, \cdots, ID_{i_k}\}, T_C, L_C\}$ where L_C is the lifetime of the coalition. The signature S_C of CA on the message I_C is given by

$$S_C = H(I_C)^d \pmod{n}$$

where $H(I_C)$ stands for the hash value of I_C with *SHS*. I_C and S_C are passed to the coalition representative a_{i_1} over a public channel.

The signature S_C of CA on the message I_C can be verified with the public key e of CA by checking whether

$$S_C^e = H(I_C) \pmod{n}$$

holds or not. If so, the coalition certified by *CA* is authentic. Otherwise, it is a forged coalition.

3.3 Coalition Signing

Here we illustrate the procedure for the k agents in C to cooperatively sign a message M. We assume here that all the agents in C have already obtained their secret keys as described in Section 3.1.

Step 1: Each agent a_{i_j} first randomly chooses an integer r_{i_j} $(0 < r_{i_j} < n)$ and then computes $T_{i_j} = r_{i_j}^e \pmod{n}$.

Step 2: Each agent a_{i_j} submits T_{i_j} to the representative a_{i_1} of coalition C.

Step 3: a_{i_1} computes $T = T_{i_1} \times \cdots \times T_{i_{k-1}} \times T_{i_k} \pmod{n}$ on behalf of the coalition and then broadcasts T to members of the coalition.

Step 4: After receiving T, each agent a_{i_j} first computes $m = \mathrm{H}(M, T, S_C)$ and then computes

$$D_{i_j} = r_{i_j} s_{i_j}^m \pmod{n}$$

and submits D_{i_j} to a_{i_1}.

Step 5: a_{i_1} computes $D = D_{i_1} \cdot D_{i_2} \cdot \cdots \cdot D_{i_k} \pmod{n}$ on behalf of the coalition C and constructs the coalition's signature $Sig_C(M)$ on M. $Sig_C(M)$ consists m, D, S_C and I_C, i.e., $Sig_C(M) = \{m, S_C, D, I_C\}$.

3.4 Coalition Signature Verifying

Here we illustrate the coalition signature verification process. The coalition signature $Sig_C(M) = \{m, S_C, D, I_C\}$ on a message M can be verified by any verifier V in the following way:

Step 1: V Checks the authenticity of the coalition according to equation $S_C^e = H(I_C) \pmod{n}$ where e is the public key of CA.

Step 2: V Computes $h_{i_j} = \mathrm{H}(ID_{i_j})$ for all the agents a_{i_j} $(1 \le j \le k)$ that belong to C. Then computes the value of h, $h = h_{i_1} \cdots h_{i_{k-1}} h_{i_k} \pmod{n}$.

Step 3: V Computes $T^* = D^e \cdot h^m \pmod{n}$.

Step 4: *V* computes $m^* = \mathrm{H}(M, T^*, S_C)$.

Step 5: If the above calculated value m^* equals to the value of m in the coalition signature, then the coalition signature is valid.

4. CONCLUSION

The pervasive connectivity of the Internet and the powerful architecture of World Wide Web will create a virtual marketplace where tens and thousands of agents can work, interact, and also attack from anywhere and at any time. The electronic linking, either wired or wireless, among various agents has made security an issue that must be woven into any agent-based service environment. In this paper, we propose a coalition signature scheme within a multi-agent environment. We believe that there are a lot of security issues when multiple agents interact with each other and we plan to explore more in the future.

5. REFERENCES

[1] L. C. Guillou and J. J. Quisqater. A 'paradoxical' identity-based signature scheme resulting from zero-knowledge. In *Proc. CRYPTO'88*, pages 216-231, Springer-Verlag, 1990.

[2] S. Kraus and J. Wilkenfeld. Negotiation over time in a multi-agent environment: preliminary report. In *Proc. IJCAI'91*, pages 56-61, Australia, 1991.

[3] K. Lerman and O. Shehory. Coalition formation for large-scale electronic markets. *International Conference on Multi-agent Systems*, Boston, 2000.

[4] T. Sandholm. Negotiation among self-interested computationally limited agents. *Ph.D. Thesis*, University of Massachusetts, Amherst, MA, USA, 1996.

[5] O. Shehory and S. Kraus. Feasible formation of coalitions among autonomous agents in non-super-additive environments. *Computational Intelligence*, 15(3): 218-251, August 1999.

Proceedings of the 2nd International Workshop on Security in Mobile Multiagent Systems

associated to AAMAS-2002
Bologna, Italy

Klaus Fischer and Dieter Hutter (Eds.)

RR-02-03
Research Report