Trust in Mobile Agent Systems

J. Todd McDonald[1], Alec Yasinsac[1]

[1] Florida State University, Department of Computer Science, Tallahassee, FL 32306-4530
{mcdonald, yasinsac}@cs.fsu.edu @cs.fsu.edu

Abstract. The protection of mobile agent systems continues to be an active area of research that will enable future applications to utilize this paradigm of computing. Agent systems and mobile applications must balance security requirements with available security mechanisms in order to meet application level security goals. We pose models for viewing security in application contexts that aid the mobile agent system design process. Our models relate security attributes for principals in mobile architecture to support reasoning about applicable security requirements, trust expression among principals, and mechanisms for protection.

1 Introduction

Mobile agents are autonomous programs with the capability of changing their location execution through a series of migrations and corresponding state updates. Applications based on the mobile paradigm must be able to specify and enforce security requirements given the unique interactions among hosts, static agent components, and dynamic agent components. Traditionally, mobile agent security has focused on protection mechanisms that keep malicious parties from altering the agent and on protection mechanisms that keep malicious agents from harming other parties. Several surveys have been written to categorize and describe the taxonomy of attacks against agent systems and the mechanisms that are available to combat those attacks [1–6]. Little has been done in research to bridge the gap between requirements, trust expression, and protection mechanisms at an application-centric level.

Mobile agents, despite many challenging security issues, are seen as a key concept to support future distributed applications based on roaming entities in large network infrastructures [18]. They are ideal as a programming paradigm to conceptualize operations in pervasive computing scenarios, but they still require a great deal of maturity in the area of security before being successful as an application development approach. In order to realize the vision for "anytime-anywhere computing" [20] or "sea-of-data" applications [29], the foundations for modeling secure mobile agent operations need to be established first. These foundations include models that express agent interactions, security requirements, and application trust properties in mobile contexts.

When dealing with application development, trust is used frequently to understand and define security requirements. Trust formulation has been given considerable thought both in the world of distributed networking applications [7–11] and mobile agents [1, 12–21, 25–29, 31–33, 35]. Mobility as an application feature complicates trust because a host receiving a mobile agent for execution must make distributed trust decisions in the face of little or no prior knowledge. Likewise, agents that act on behalf of users must evaluate trust relationships with agent execution environments

and support a wide range of possible trust scenarios. Applications based upon mobile agents must blend the security requirements of a user with the trust expectations of the environment where agents execute.

The contributions of the paper include initial development of a trust model for mobile agents with novel features: linkage of application security requirements with mechanisms based on trust, reasoning about trust properties for generic security mechanisms and generic trusted servers, incorporation of trust attributes from mobile ad-hoc networks, and consideration of human agents on trust levels. Our application models provide a transition between application level security goals, trust expression, and security mechanisms which are currently available for mobile agent systems. We propose different models that can initialize a set of trust relationships among principals in the mobile agent system.

The remainder of the paper is organized in the following manner. Section 2 discusses related works concerning trust and security requirements in the mobile agent paradigm. Section 3 presents our framework for expressing trust and security in mobile agent systems. Section 4 expounds three different application-level security models: the military model, the trade model, and the neutral services model. Section 5 concludes with a discussion and summary.

2 Related Works

Before describing our notion of trust and security at an application-level context for mobile agents, we first review foundational concepts appropriate to this realm. Defining *trust* is as precarious as defining the term *agent*—and though researchers do not agree on either term they do discern the importance of both concepts in framing research. We define trust loosely in a previous work [10] as the expectation of behavior among parties and classify several different infrastructures for trust in dynamic networks. Gambetta [22] defines trust as a subjective probability that is non-reflexive, changing, and context driven. We note also that trust is not necessarily Boolean, does not necessarily have to capture human intuition, and can involve third parties to facilitate certain issues. Trust can be transitive and therefore delegated [16] or can be acquired by direct observation.

The trust management problem, as defined in [8], seeks to formulate a coherent framework for defining policies, actions, relationships, and credentials in terms of trust. Trust management systems such as [7, 8, 18, 19, 20] are designed to support specification, acquisition, revocation, degradation, and evolution of trust according to some model. As we point out in [10], challenges in these systems revolve around observing trust-related actions and then translating those observations into a decision. In both mobile agents systems and dynamic networks, trust decisions have to be made before trust relationships are ever built. The overlap of trust models with the development and implementation of mobile agent security frameworks is a key to future support of pervasive computing scenarios. We consider next the adaptation of trust mechanisms specifically to the mobile agent paradigm.

2.1 Security and Trust Expression for Mobile Agents

Mobile agent applications and the idealized vision of a global computing scenario share many common characteristics with distributed trust models for dynamic networks: a large number of autonomous parties, principles can have no prior

relationship, a trusted or centralized computing base may not exist, virtual anonymity of principles may exist, different administration domains, and hosts have different capabilities for transmission, mobility, and computational power.

Grandison and Sloman in [9] point out that trust cannot be hard-coded in applications that require decentralized control in large scale heterogeneous networks. Mobile agents particularly need to separate the application purpose from the trust management framework if they are to scale well in such environments. Because there is a large commonality between dynamic networks and mobile agent applications, many proposals for defining trust infrastructures in ad hoc networks can be readily applied to mobile agents. Kagal et al. [31] suggested the addition of trust to enhance security for mobile computing scenarios and defined trust management activities as those defined by [8]: developing security policies, assigning credentials, checking credentials against policy requirements, and delegating trust to other parties.

Cahil et al. in [18] expound the research goals of Secure Environments for Collaboration among Ubiquitous Roaming Entities (SECURE)—a project focused primarily on building trust infrastructures for large ad hoc wireless networks. SECURE utilizes the use of both risk and trust to determine whether interaction can occur in a scenario where billions of possible collaborators may exist. In their authorization framework, a principle uses trust to decide an interaction based on the risk involved. Carbone et al. expound the trust model for SECURE in [19] and provide a formal way to reason about the trust relationship among principles. Of interest to us is their use of a trust interval from 0 to 1 which can be thought of as a measure of uncertainty. The trust level in SECURE is thus a range with some upper bound—which sets the maximum amount of trust within some factor of unknowing involved.

As Cahil and his colleagues also point out, traditional trust management systems delegate permissions using certificates (credentials) that reduce to very static decisions that do not scale well or allow change over time. They also point out as we do in [10] that trust decisions in pervasive scenarios *should* come instead from trust information based on two sources: personal observations (previous interactions) and recommendations from other parties we trust (transitive or delegated trust). However, in [18] there is no link provided between security requirements, trust, and application goals for mobile agents specifically—a goal we set forth to accomplish in this paper.

In a similar vein, Capra in [20] defines a formal model for hTrust—a mobile network trust framework that defines formation, dissemination, and evolution of trust. Capra reviews several trust management frameworks that have the following limitations: they are designed for centralized servers, they have too high of a computational overhead (for mobile devices), they lack dynamic trust evolution, they lack details about local policy usage, and they lack subjective reasoning capabilities. hTrust remedies these shortcomings and incorporates a human notion of trust directly into the framework.

hTrust models a range of trust values between principles so that a lack of evidence or knowledge decision can be distinguished from a trust-based decision to specifically distrust another party. The trust data model also incorporates the notion of time so that relationships degrade when not kept current. Recommendations are used when a principle has no past history or to partially rely or trust a third-party assessment. Just as in human interactions, hTrust captures the notion that we favor recommendations

from people who gave us good recommendations in the past and likewise reject or downgrade advice from those who have disappointed us in the past. Finally, a key aspect of this model is its incorporation of both a social context (the environment of principles arranged in a network by which recommendations can be used) and a transactional context (the network of services which are supplied by parties in the system). While hTrust provides a generic framework for mobile application trust expression, it does not directly deal with mobile agent specific security requirements or attempt to link mechanisms for security to trust levels or the agent lifecycle.

There has also been much work specifically focused on security evaluation and trust expression for *mobile agent* systems. Karjoth et al. were one of the first to describe a security model for Aglets—a specific type of mobile agents [12]. The Aglets model included a set of principles with distinct responsibilities and defined security policies which defined access to local resources. The notion of a policy database and user preferences are also included in the model to govern interactions of aglets that have unspecified or unknown security properties. However, the Aglets model does not address host-to-agent malicious interactions or incorporate the notion of trust levels or dynamic trust determination.

Other security management systems designed specifically for mobile agents suffer from the same limitations and focus on *malicious code* protection. Jansen posed a privilege management system for agents in [2] that uses traditional certificate-based policy framework. Security in this model was enforced through host-based policy specification compared to agent-based attribute certificates. When these policies merged during agent execution, the security context of the agent was determined. Other works such as [24] describe reconfigurable policy support and surveys such as [6] provide a good summary of issues and status with policy-based security implementation issues. Again, such mechanisms tend to not scale well, tend towards static security interactions, and do not model trust levels for specific security requirements.

Antonopoulos and his colleagues in [14] develop a general purpose access control mechanism that is distributed in nature and specifically focused on mobile agents. This approach comes closer to expressing trust relationships among principles, but is fundamentally based on access control lists and preventing malicious code activity. Kagal et al. extended their delegation trust model to mobile agent scenarios in [32] and address FIPA-specific weaknesses for securing the agent management system and directory facilitator services. Although they address how multiple agents can establish trust when they are previously unknown, their focus is primarily on authentication and they do not consider mobile-specific security requirements and trust expression.

Tan and Moreau [17] develop a more comprehensive model of trust and belief specifically for *mobile* agent-based systems based upon the distributed authentication mechanisms posed by Yahalom et al. [7]. Their trust derivation model is founded on the similarities between distributed authentication in pubic key infrastructures and mobile agent trust. The Tan/Moreau framework is limited and simple, however, because it only incorporates the notion of trust associated with using the extended execution tracing security mechanism [23] and does not account for generic security mechanisms or requirements. Their unique contribution in the area is one of the only works that link security mechanisms with

Borrell and Robles with several different colleagues have done significant work to incorporate trust in mobile agent system development [21, 25-30]. In [21], a model is presented that defines trust relationships expressed in the MARISM-A mobile agent platform [33]. Although initial work in the area by Robles and his colleagues assumed a static trust expression among agents [26-28], MARISM-A now uses trust relationships that are defined among entities and actions in a mobile agent transaction. We follow similarly with a formal model that defines trust relationships between entities and associates actions and attributes to each relationship. The MARISM-A model uses trust relationships to define decisions that permit, obligate, designate, or prohibit certain actions within a certain scope of a mobile agent program. We take a similar approach as Robles by associating agent security mechanisms with trust relationships and by classifying their role as either a deterrent, prevention, or correction. In addition to certain similarities with the MARISM-A approach, we expound more fully in our model the relationship between the agent applications, principles, security mechanisms, trust levels, and trust determination.

Ametller along with Robles and Ortega-Ruiz take the notion of policy-based security mechanisms even further in [30]. In this scheme, agents select the appropriate security mechanism to use by means of a security layer that exists within the agent itself and that is not tied to the underlying agent execution platform. Security layers of the agent interact with hosts to determine which mechanism will be used by an agent based on some predetermined criteria. Assuming a decryption interface and public key infrastructure are in place, the solution gives a novel security addition that is implemented in JADE and provides a flexible approach to agent security. We build also upon this approach by linking in our model and methodology the notion that security requirements for agent applications are met by security mechanisms—all of which are tied to the evaluation and evolution of trust relationships among principles in the mobile agent environment. We discuss next the specific formulation of security requirements for mobile agent applications that are referenced in our trust model.

2.2 Security Requirements for Mobile Agents

As much of the literature bears out, meeting security requirements for *mobile agents* is not as simple as just applying cryptographic primitives. Agents complicate and extend normal security requirements for a typical network-based distributed application. Requirements for security in mobile agent applications derive from the way in which mobile agents are defined as programs made up of three components: static program code, a dynamic data state, and a current execution thread. Agents execute only in context of a migration path (itinerary) among a set of hosts (servers).

We consider for our purposes observations from [34, 35] that define the agent computation and its corresponding security properties. In [34], Yee describes the host as a server S_i with a resource R_i. The agent is seen as a program that carries a collection of data states and a static program. The static code is further subdivided by Yee into a query function, $q_i(y)$, which takes a current state y as input, and a state update function, $f_i(x,y)$, which takes a query result x and current state y as input. The server resource R_i takes as input the query function and associated state, executes the query within the host environment, and returns a query result y. The agent's state is thus updated along a continuum of interactions between the host, state, and code. We

highlight this model because it was useful in [34] for defining precisely how certain security mechanisms could prevent state violations based on agent replay attacks.

Tate and Xu in [35] similarly view the computation as the interaction of three objectst: agent code, agent state, and host data input. For simplicity, we choose to analyze security requirements based on these three components as well. The agent itinerary, its collection of intermediate results, its unique identifier, its security policy, any activity log or history, and any other protocol specific information can be considered as part of the overall dynamic data state of the agent, though they might be referred to separately from a security context. One input to the agent computation at the current host s_i is the agent state y_{i-1} that is carried from the agent's previous execution on host s_{i-1}. The other input to the agent computation is the consolidated host data input (Yee's combination of interactions between host resource R_i and query function q_i).

In our proposed model, we establish a link between security mechanism and security requirements for mobile agent applications, using trust relationships as a basis for evaluation. Tan and Moreau [17] classify agent protection requirements in their trust model broadly as execution integrity, state/code integrity, state/code visibility, and resource provision. Here execution integrity includes correct transformation of current state to a new state (Yee's [34] state update function *f*), no violation of the semantics of the agent's code, and correct orderly execution of the agent code. Tate and Xu [35] do not deal with availability (resource provision) or anonymity but do specify three types of data privacy that apply to the host input, which we adopt by their definition. State and code visibility refer to the privacy of sensitive information in parts of the agent which are visible to the host. In [35], the host's data input might be considered worthy of protection as well. Table 1 lists the security requirements that follow from the traditional CIA (confidentiality/integrity/availability) model of describing security. We discuss each requirement briefly to provide a context for our application level security models.

Table 1: Mobile agent security requirements

Confidentiality	**Integrity**
Agent state privacy	Agent state integrity
Agent code privacy	Agent code integrity
Agent anonymity	Server integrity
Limited host data privacy	**Non-repudiation**
Verifiable host data privacy	Agent non-repudiation
Complete host data privacy	Server non-repudiation
Availability	**Authentication**
Agent accountability	Agent code authenticity
Agent authorization	Agent identification
Agent code verification	Host authenticity
Agent privilege revocation	
Host accountability	
Host availability	

Mobile agent *applications* are *constructed* by using an underlying architecture that *integrates* some form of agent mobility. Since real-world practical applications drive security requirements, it is apparent that not all mobile agent applications require the same level of security. In many cases, the expected environment of a given application sets the tone for which of security requirements it needs to focus on. These requirements stem from the

nature of the application itself and how it will leverage mobility to accomplish its overall task.

Confidentiality deals with keeping information private and in the mobile agent context this involves several issues: keeping server results from observation of other servers or agents, keeping parts of the agent data state safe from observation, keeping the algorithm or *function* of parts of the agent code private, keeping the itinerary secret (other than possibly the previous or next host), and keeping server data free from unauthorized agent observation. In certain contexts, the agent or server may want to keep their identity anonymous.

Privacy of agent data is difficult when prior execution results are embedded into the observable agent data state because the agent server must have access to the agent's code and state in order to execute it. To disallow sensitive information from being revealed in plaintext while executing in untrusted environments, computations that work on encrypted data might be required [37] or specialized encryption of partial results must be performed independently of the data state computation. Trusted hardware is a strong contender for protection at this level, but does not automatically guarantee secure protocols [39]. Secrecy of the code or algorithm is currently achievable only in very limited contexts using function hiding [40, 38] or secure multi-party computation techniques [35, 38, 41].

The requirement for integrity deals with detection or prevention of alterations to information—whether your own or someone else's. For mobile agents, an untrusted server environment can altar the intermediate data results or state of an agent that is under its execution control. It can also change the static code of an agent or create clones of agents that can be used for malicious purposes. Agents need to be protected against these types of attacks and other forms of alteration that change the normal, intended execution pathway of an agent. The agent itinerary, which can be considered a subset of the agent's code and data state, also requires protection from malicious alteration during the agent's lifetime. Some work has also been done to keep the path of an agent private and secure from alteration [42, 43, and 44]. Integrity of intermediate data results has been a widely researched field for mobile agents [39, 45, and 46], mainly because of its association with e-commerce applications and many solutions have been proposed and tested (see [36] for a thorough review).

In terms of availability, agents and servers need to be mutually protected from each other when they attempt to deny or delay service, use resources in an unauthorized way, or inflict damage via malicious code or services. Denial of service can be detected and averted in certain time-limited contexts [23] and both agents and servers must be held accountable for their actions—especially when malicious behavior is in view. Servers and agents also both need to be able to authenticate who they are. The authentication of an agent's code as belonging to a particular owner, however, does not make any statement about its trustworthiness or safety. The identification of an agent and the unique binding of an agent's static code to its dynamic state has been the topic of previous research as well [39]. Lastly, non-repudiation can be required so that an agent or server cannot deny transactions or actions that have been taken on behalf of their owner. Agents should not be able to wrongly attribute a transaction to a server that did not make it nor should servers be able to wrongly charge agents for actions they did not intend.

2.3 Security Mechanisms for Mobile Agents

Security requirements are defined routinely as the desire to guarantee one or more of the following: privacy, integrity, availability, authentication, and non-repudiation. *Security mechanisms* are targeted at enforcing one or more these requirements. In the mobile agent paradigm, security requirements are linked to an *application* which is designed to implement a particular task. The desire to see a set of security objectives enforced for this task comes not only from the dispatching application owner but also from the set of hosts that are visited. Requirements are seen as applying to either a host (where an agent will be executed) or to a mobile agent (which is dispatched to a set of hosts).

The code and state of an agent are distinguished when defining security requirements due to the intricacies of the mobile agent computation. On the side of the agent, an application can require code integrity, code privacy, state integrity, state privacy, agent authenticity, agent availability, agent anonymity, and agent non-repudiation. Code privacy refers to whether the algorithm is public or private while state integrity refers to the correct and unaltered execution of the agent. State integrity also includes the full range of partial result protection and the guarantee against insertion, deletion, or alteration of results from prior hosts. Code integrity provides assurances against alteration of the static code and authenticity guarantees the identity of an application owner or agent code developer can be verified. In some cases, an application may demand that agents act anonymously [42].

On the host side, an application can require host privacy, host availability, host integrity, host authenticity, and host non-repudiation. Host privacy can range from limited, verifiable, or complete [35], depending on the amount of information a host is willing to reveal regarding its data input. Host privacy can also refer to the protection of non-agent-application specific resources part of the local host platform. The other requirements follow those of traditional execution environments where integrity of the execution environment and resources needs to be preserved and the identity and actions of a server need to be verified.

A number of works in the literature define agent security requirements and categorize agent security mechanisms in different taxonomies [1–6, 17, 34, 56]. A voluminous number of works abound that detail *specific* proposed security mechanisms—whether host-based or agent-based. Host-based mechanisms protect a host from possibly malicious agents. A few mentioned in [2, 36] include sandboxing, safe interpreters, code signatures, state appraisal, path histories, and policy management. Agent-based mechanisms protect the agent from malicious activity outside of itself and are too numerous to name all proposed to date. Several commonly referenced mechanisms include function hiding [37, 38, 40], secure multi-party computation techniques [35, 41], intermediate data result protection [39, 45, 46], execution tracing [23], and tamper-proof hardware [44].

Several works have attempted to meld mechanisms and requirements at an application level already. Claessens et al. [38], for example, formulated an implementation-detailed study of security mechanisms that were appropriate for secure electronic transactions. Singelee and Preneel [47] went further by posing an actual security level solution for e-commerce using mobile agents. In their approach a mobile agent system in e-commerce settings could be secured using hash-chaining data integrity protocol for data collection [45], threshold undetachable signature

scheme for payment protection [48], and environmental key generation for agent data privacy [49].

As many authors have pointed out, no one security mechanism can address every security requirement. The use of security mechanisms in fact may establish a certain level of trust (or lack thereof) in the face of certain environmental assumptions about malicious parties. An application level view of security that we propose would bring together a process for selecting a combination of techniques to achieve certain trust levels within a mobile agent system. Even when using mechanisms that establish pure trust (such as tamper proof hardware), other assumptions must be taken into account to make sure security is guaranteed. Trusted hardware or multi-agent secure cryptographic protocols may be warranted or even feasible given certain application environment factors. When such mechanisms are not available, lower trust levels can be demanded and a more policy-driven approach can be required to make dynamic decisions about agent execution.

Models are used in many cases to help describe in more precise terms a particular set of relationships. Models in the security sense do several things such as: help test a particular security policy in terms of completeness or consistency, help document security policies, help conceptualize or design an implantation, or verify that an implementation fulfills a set of requirements [50]. We now present our trust framework for considering mobile agent applications and describe a model for viewing principles, actions, and trust relations in the mobile agent environment.

Our initial model for trust expression in mobile agent environments incorporates three separate notions: security requirements, security mechanisms, and trust in a mobile application setting. In the following sections, we briefly highlight references which expound each of these security facets.

3. Trust Framework

As Cahill describes in [18], we use trust normally in distributed applications to determine such things as whether to download a file, which service provider to chose, which access rights to give, and which information do we deem reliable. Tan and Moreau postulate in [17] that trust in mobile agent applications can enable further security-related objectives: choosing which set of code security mechanisms to use, choosing how to punish violators of security, scaling agent applications when large sets of unknown hosts may be visited. Capra points out further in [20] that traditional distributed trust systems are centralized with clearly defined administrative boundaries—with limited mobility of entities and high likelihood of trusted third parties that are available to help secure applications. To know an entity, in this respect, was to trust an entity. However, the mobile agent paradigm does not have such luxury: even knowing an entity (either a priori or through real time directives) does not automatically mean trusting an entity.

Mobile agent systems must deal with environments where partial knowledge and blind trust are common. Subjective determinations in many cases might need to be made and in some cases delegation of the trust decision may not be possible because of the network environment. The complexity of trust is compounded by the fact that every principal in an agent application can have varying levels of one-way trust for different security requirements and in different contexts between each other. To formally describe our framework for trust in the mobile agent environment we first

define the principals that can be assigned trust properties in our model, define next the nature of trust relationships between principals, and then formulate what trust relationships can be used to accomplish in mobile applications settings.

3.1 Defining Principals

There are three distinct groups from which principals can be derived in mobile agent systems: agents, hosts, and entities. The set of principals is now summarized as the collection of agents, hosts, and humans involved in a mobile agent applications.

$$\textbf{PRINCIPALS} = \{ \textbf{agent}^{+} \cup \textbf{host}^{+} \cup \textbf{entities}^{+} \}$$

We agree with the notion in [17] that agents will ultimately one day be composed of reusable components and that each component will have its own associated dynamic state and trust level. For simplification we define an agent as a composition of a (single) static code and a set of dynamic states that represent the migratory results of the agent. We further delineate the path of an agent as its itinerary, a unique identifier of the agent instance, a generic log of agent or host activity, and an agent security policy that includes any historical trust information for other principals in the agent application.

$$\textbf{AGENT} = (\ \textbf{code}\ ,\ \textbf{state*}\ ,\ \textbf{itinerary}\ ,\ \textbf{id}\ ,\ \textbf{log}\ ,\ \textbf{policy}\ )$$

The "id" of an agent encompasses more than one identity as well. We use the notion of the agent kernel from Roth in [39], which in essence uniquely binds a given instance of a mobile agent's dynamic state to its static code. This identification eliminates the possibility for cut/paste and oracle-style attacks that several security mechanisms were vulnerable to. The static code, the owner of the application, and the developer of the code all have unique identities as well—which are captured in the id component of the agent.

Hosts are defined as the execution environment which agents operate in. They are seen abstractly as encompassing the underlying physical hardware, runtime services, and agent-related middleware necessary to accomplish agent migration and execution. Hosts can be seen as a collection of resources or services that interact with a mobile agent. Host resources can fall into several categories: computational, communicational, informational, management, and security. Resources conceptualize the notion of services provided by local (non-itinerant) agents, host based software processes, host based physical resources such as memory and processor, and any information necessary for the accomplishment of the agent task. Hosts likewise have policy that support the trust formation and decision process while interacting with mobile agents that it executes.

$$\textbf{HOST} = (\ \textbf{resource*},\ \textbf{id},\ \textbf{log},\ \textbf{policy}\ )$$

There are at least three types of hosts which can be distinguished in mobile agent applications. The *dispatching host (DH)* is the launching point for a mobile agent application—it is considered the owner of one or more agents that are acting on its behalf and ultimately *partially* responsible for the actions of agents under its control. For simplicity, we assume that an agent application will have only one dispatching host. The *executing host (EH)* is a server which is part of the itinerary of a mobile agent and upon which an agent executes and updates its state. The third type of host

is the *trusted host (TH)*: it conceptualizes servers that provide security benefit for agents during their lifetime. They play the role of trusted third parties in many different security mechanisms—such as extended execution tracing or multi-agent secure computation schemes.

$$\mathbf{HOSTS = \{\ DH\ \cup\ \{\ EH^*\ \}\ \cup\ \{\ TH^*\ \}\ \}}$$

In terms of humans involved in mobile agent applications, we identify three that play security roles in the mobile agent interaction. First, agent code derives from a specific source—humans that program them. We refer to the creator of the agent as the *code developer (CD)*. The person that actually *uses* mobile agent code to accomplish a task on their behalf is termed the *application owner (AO)*. In many instances, the code developer and application owner are the same person. Hosts also do not run without human influence and design. The owner of a computer (the company you work for), the manager of a computer (a company employee such as a system administrator), and the user of a computer (a company employee that works in payroll) are normally separate individuals. We assume for simplicity that the human owner/manager/user of a host are synonymous and use the term *host manager (HM)* for this human role.

$$\mathbf{HUMAN = \{\ CD\ |\ AO\ |\ HM\ \}}$$

In terms of trust, humans trust machines (hosts) and software (agents) because they trust the source or manager of these principals. To simplify trust relationships we will assume that all hosts have equivalent trust to their corresponding host manager and only refer to hosts in discussion of principals. The interesting case is the fact that the *host manager* for the *dispatching host*, the *code developer*, and the *application owner* can all be different individuals or could be the same person. For simplification, we will equate the trust in the agent code as the trust we place in the code developer and we will equate the trust we have in the application owner as the trust we have in the dispatching host.

$$\mathbf{A \approx CD}$$
$$\mathbf{DH \approx (D)HM}$$
$$\mathbf{EH \approx (E)HM}$$
$$\mathbf{TH \approx (T)HM}$$
$$\mathbf{DH \approx AO}$$

As a final definition, we consider the concept of an application. Since applications are where security requirements of a user merge with the actual use of mobile agent code, we define an application as the collection of all possible hosts that will be involved in the agent task and the set of uniquely identifiable agents that implement a user function. This intuition captures the notion of multiple agents that can interact to accomplish an application function: multiple agents with the same static code and different itineraries, multiple agents with different static code, or a single agent.

$$\mathbf{APPLICATION = (\ host^*, agent^*, trust^*\ )}$$

Applications, not agents, therefore become the focal point for trust determination, security requirements, and security mechanisms. We now define our notion of trust relationships that are shared among principals in our model.

3.2 Defining Trust Relationships

One task of security is to rightfully attribute observed actions within the system to a given party. This task is complex in the mobile agent environment because the current data state and code of a mobile agent is influenced by many different parties: the code developer, the dispatching host, all executing hosts the agent has visited, and all trusted hosts the agent has visited. When considering the agent life cycle and the binding of trust at various stages we formulate the following connections: creation and development of code bind trust to a code developer, ownership of an agent binds trust to an application owner, dispatching an agent binds trust to a dispatching host, execution of an agent binds trust to all prior hosts an agent has visited plus its dispatcher, migration binds trust to the next host in the agent itinerary, and termination binds trust of the entire application to the entire set of execution hosts and the network environment.

The mobile agent trust problem can be likened to a person walking up to you in your office and handing you a floppy disk. They say to you, "I have heard that you like sharing security research information with others. Please run the file on this disk called 'virus.exe' on your computer for me and then I'll walk it to the next security researcher on my list of contacts when you are done. Don't worry about the output—the program keeps its state every time it executes." You, acting as the host, must ask a few probing questions of course. Before considering whether to run the file or not, the mental process begins first by considering the trust you have in the person who brought the file. This represents the application owner in the mobile agent environment. Do you know them personally? If not, do you know other people who know them? If not, how will you assess their trustworthiness? Will you ask them for a resume or a list of references first?

If the person is unknown to you personally you may want to see their credentials so that when you ask for references you are sure you are talking about the right person. In the mobile agent context, the application owner determines the *agent signature*: the binding between the dispatcher (sender) of the agent program and the identity of that particular *instance* of the agent code execution. When parties are completely unknown, you may determine trust based on some type of testing suite you require them to accomplish first. If you know them to be untrustworthy or assume that anyone asking to run files on your computer from a floppy is untrustworthy by default, then the emphatic answer will be "No, I will not run that program."

If the person is known to you or has a history of dealings with you, you may (after verifying they do not have an evil twin) entertain the idea of running 'virus.exe' on your computer. Assuming you pass this first hurdle, the next mental process turns to the question of the code itself. Where did 'virus.exe' come from? Who authored it? Have they authored other programs that have proven trustworthy? Are they identified with hacker groups or malware? In the mobile agent paradigm, there are limited ways to ascertain such trust relationships. If the person indicates there is a software clearing house that has reviewed and tested his code for safety and malicious behavior, passing with flying colors, some of your fears may be allayed. This is equivalent to using a trusted third party to assess trustfulness of the agent code or the code developer a priori. A *code signature* links the developer with the code itself (even though many application owners may use that code in multiple ways and

instances over time). However, some other method of proving or verifying safety of code must still be used if those requirements are important.

If you know the person carrying the disk (let's say they are a good friend) AND you know they authored the software AND you trust them enough to execute the file, the remaining questions might focus on the nature and purpose of 'virus.exe'. What does 'virus.exe' do? If the algorithm is private, the person may say "I can't tell you, you just have to trust me"—which at that point you determine that you still won't run the program even for a good friend. If the algorithm is public, then you may be relieved to find out that 'virus.exe' is a statistical data gathering program that queries your anti-virus software to see when the last time you updated your virus protection was and how many viruses have been detected in the last month.

The agent dispatcher also may have questions for you as the host to answer. How will they know whether or not you have modified their program? How will they know whether or not you have spied on results from other security researchers? How will they know you don't have a vendetta against other researchers and want to skew their results? How will they know whether or not you will just destroy their floppy or just not run the program like you said you would? If you are known to the person or have a history of dealing with you, they may not worry with certain detection mechanisms for security violations—maybe they trust you with those. Even if you are known, maybe it's the nature of the beast that they would still protect certain information associated with the program.

These trust questions are exactly the same as those which must be modeled in mobile agent systems—except there is no luxury of face-to-face interaction and in many cases no possibility for human response when the decision needs to be made. Trust models must be able to handle environments that have different overall characteristics among participants as well. Agents and hosts can have varying degrees of trust that include highly trusted, trusted, unknown, untrusted, or highly untrusted [10]. Whether trust can be increased is an application-specific question that a trust management system should be able to handle. Rules for promotion of trust in mobile agent systems can be based upon the security mechanisms that are used in the application environment and the observed behavior of both agents and hosts in the environment.

TRUST = (foreknowledge, level, timeliness, security*)

We represent the trust relationship among principals as a 4-tuple of properties: foreknowledge, trust level, timeliness, and security. For each principal, there can be a prior knowledge of the entities or no prior relationships whatsoever. For agents that travel in a dynamic free-roaming itinerary, it is very possible to encounter a host that is completely unknown. Likewise, hosts have a high likelihood of encountering agents with no prior history or experience. Foreknowledge in this regard is well known, known, or unknown. Well known corresponds to principals that have well established histories with a high level of trust basis.

Trust levels are deemed to be a range from totally untrusted to totally trusted. Timeliness is a degree of relevance of the trust level based on time that is represented as expired, stale, or fresh—incorporating such notions as recent or frequent interactions based on historical timestamps of interactions. Trust level,

foreknowledge, and timeliness only bind principals to a certain context of security—a requirement.

Security requirements dictate what an application owner feels is necessary for their task to be accomplished successfully and securely. They also describe the level of security a prospective host would like to have concerning agents that they execute. This element captures the intuition that two principals can have different trust levels concerning different security aspects—i.e., Alice can trust Bob in regards executing her agent with code privacy but cannot trust Bob in regard to state privacy.

foreknowledge = **{ well known | known | unknown }**

level = **{ highly untrusted**
| untrusted
| non-determined trust
| trusted
| highly trusted }

timeliness = **{ expired | stale | fresh }**

security = **(principal type, objective)**

principal type = **{ host | agent }**

objective = **{ state privacy**
| code privacy
| anonymity
| …. }

In a similar manner as [18, 19, and 20], we define a trust relationship as a mapping δ between two principals (P) with an associated 4-tuple trust relationship (T).

$$\delta: P \rightarrow P \rightarrow T$$

Principals are drawn from the set of all possible parties of concern in the mobile agent application. Based on our simplifying assumptions, this reduces to the following four principal sets: dispatching host/application owner (DH/AO), all execution hosts (EH), all trusted hosts (TH), all agents/code developers (A/CD). If a more precise trust relationship needs to be expressed for human principals, code developers and application owners can be treated separately—though it adds complexity to the trust matrix. Given an application G with associated set of hosts H_x, a set of uniquely identifiable agents A_y, and a set of trust relationships $T_{i,j}$, we now formulate the trust actions and decisions that can be based upon trust relationships among all principals within the application space.

3.3 Defining Trust Actions and Decisions

In our trust framework for mobile agent applications, actions build relationships and relationships determine decisions or mechanisms that need to be used to achieve the level of security desired by the application owner. Trust in the mobile agent environment can ultimately affect allowed security mechanisms, itinerary of agents, policy decisions, and code distribution.

Our model describes actions that mobile agents and hosts do which can altar trust levels. Trust can be earned or degraded in this regard based on actions observed over

time. Decisions about trust come from a set of trust based on the following collection: an initial set of trust relationships, recommended trust from others, and our observations of trust related actions over time. Given a set of trust relationships that exist between all principals, a principal can make a trust decision: which security mechanism do I use, do I migrate to this host, do I execute all of my code, do I share my policy information or trust recommendations, do I access your policy information, and so forth.

The hTrust system proposed in [20] establishes a comprehensive method for trust dissemination, formation, and evaluation. Our model presupposes a similar underlying structure that provides functionality to 1) evaluate recommended trust from social interactions; 2) disseminate trust information to prospective principals in future mobile agent applications; 3) weigh relevance of social recommendations to prevent malicious hearsay. We assume that a pre-existing structure exists in both hosts and agents that store historical trust information which can be used in the decision making process for future applications and current execution decisions. We focus in our model more particularly on the relationship between trust expression, security requirements of the application, and the appropriate security mechanisms that are used.

The ultimate goal of our trust framework is that given initial trust relationships we can derive new ones or modify existing ones according to rules. In a previous work discussing trust in ad-hoc networks [10], four different categories of trust acquisition are formulated which we use in the mobile application context as well: trust-earning actions over time, trust-earning actions by count, trust-earning actions by magnitude, and trust-defeating actions.

To measure trust actions over time, timestamps from the first interaction to the current time can be used, with longer histories of interaction being considered more favorable. The longer we know someone and more extensive their good track record is with us, the more trust we have in them. Of course the volume and frequency of interactions will flavor this as well: 500 interactions in the last year provide a better basis for trust than only 1 interaction in the last 5 years. Given the same volume of interaction, trust certainly is higher when the interaction period is longer. When the elapsed time since the last interaction is short, we also have higher confidence in the freshness of the trust level.

Trust-earning actions by count in the mobile environment can be based upon the number of times interaction has taken place with principal without any detection of malicious behavior. If the average time between interactions is small, this assumes more interactions and more opportunity for dishonest behavior to manifest. Trust can also be influenced by working cooperatively and non-maliciously in the past on actions that are great in magnitude. This can involve a number of transactions that have a certain dollar amount or involve significant trust such as an EEPORM hardware upgrade. Trust defeating actions are similar to the notion of a bankruptcy on a credit report that negatively affects the trust of a lender towards a borrower. Once trust defeating actions have been entertained, policy can dictate that the agent or host never regain a trusted status. Trust might be regained only after a (long) time period of demonstrated, small trust gaining actions have been observed. It is also important to distinguish between first-hand violations of trust and recommendations of trust that are negative toward a party.

As we describe in our toy example, trust in a person executing a program is not the same as trust or assurance of the program itself. In mobile agent paradigms, the identity of the application owner, the identity of the code developer, and the identity of the instance of that particular agent run all need to be authenticated. Integrity of the agent can be accomplished on two specific components: the static code can be verified to make sure it has not been altered and the initial state of the agent when it was dispatched can be verified that it was not altered. Detecting other state alterations requires other mechanisms to give assurance that an agent's state has not been unduly altered by other executing hosts in the itinerary. The safety of the agent code or the assurance that it meets a predefined security policy can be accomplished by methods such as proof carrying code [51].

Other means for establishing trust—of either the current agent state or the code itself—must be used in this case and several methods have been proposed which are reviewed in [36]: holographic proofs, state appraisal, proof-carrying code, path histories, detection objects, oblivious hashing, protective assertions, state transition verification, reference states, execution tracing, and sliding encryption.

Roth was one of the first to point out [39] that mobile agent protocols (even those that involve trusted hardware) are vulnerable to cut and paste attacks. By interleaving mobile agent sessions and using honest hosts as oracles to decrypt parts of the agent data state that are part of ongoing agent sessions, a malicious adversary can foil many attempts at data protection. To avoid this problem, Roth points out that the data state of the agent needs to be bound to the static code of the agent. In order to accomplish this, an agent's static code or program can be tied to a random number or nonce and encrypted with a signature key. The hash of this signed pair (referred to as the kernel by Roth) can be the basis for the agent's identity. By associating a unique identity with the execution run of a mobile agent, the problem of cut and paste attacks can be avoided. This property distinguishes among multiple runs of the same static agent code and introduces the finest level of trust relationship among principals in the agent environment.

3.4 Defining Trusted Hosts

A unique concept in our trust model is the generalization of the notion of trusted hosts. These hosts conceptualize properties normally associated with trusted third parties in various security mechanisms and protocols. Trusted hosts first have specialized, static, and pre-determined trust relationships with principals. What makes them special is that trust levels do not change with agents or hosts that they interact with. If a trusted host is only partially trusted in a give environment, then we represent that host as an execution host with certain trust relationships just like any other host.

Trusted hosts are distinguished from dispatching or executing hosts in an agent application. Execution hosts that are trusted (termed trusted nodes by some) can be used to provide some level of detection ability for malicious behavior as long as it is part of the agent's normal execution and itinerary traversal. A *trusted host* provides a third-party service such as information lookup, mediation, brokering, communication service, or middle-agent hosting. Whether an agents communicates statically or dynamically migrates to the trusted host is not considered important in our model. The primary intuition that is captured by their presence in the model is that they

provide a means for either increasing or decreasing trust levels of other principals, and we trust them to do so.

For example, in the Tan/Moreau model, the trusted server is used to provide a verification service for hosts that have just executed an agent. They also provide a migration service and in essence are the only means by which agents can move from one executing host to another. When a host has been found to violate the integrity of the agent, the trust level of that host is automatically communicated via the trust framework to other agents and hosts in the system.

What normally produces high trust levels in mobile applications include the presence or use of trusted or tamper-proof hardware, reputation, being under the same management domain, and having an established history of trusted/non-malicious interaction. Executing hosts can have a trust level based on one or more of these factors—i.e., an application owner trusts *greatly* a host in its own management domain that has tamper-proof hardware (TPH) installed. TPH can be used to offset trust levels when hosts are unknown and untrusted as well—i.e., if a host will install TPH to support agent execution, application owners may assign them a trusted status.

In our model the trusted host can influence several factors: they can increase trust or decrease trust among one or more principals, they can make hosts trust agents more or less based on their services, they can make agents trust hosts more or less based on their services, and they can make hosts trust other hosts more or less based on their services. Trusted hosts can also be the implementer of a particular security mechanism, as in extended execution tracing [23] and multi-agent secure computation [41].

When the interaction mode with an agent is via migration, trusted hosts can inspect the agent, run tests, or other interact with the agent as any other host. Agents can then proceed to their next destination in the itinerary. When the interaction mode is via static communication, agents can pass on to the trusted host information for data logging, result protection, integrity checks, or to phone home information in lieu of the dispatching host. The net effet of trusted hosts in our model are as principals with authority based on their particular service to increase or decrease trust level among other principals. Whether this effect is realized by communication or migration of agents (which can be seen as a specialized form of communication) is not critical.

Trusted hosts may also be used to enforce a particular security requirement in a mobile agent context. When all agents and hosts interact with a trusted host that offers a particular service then a trust relationship can be formed in which that particular requirement is given a certain trust level between principals. They can also have a net result on the system by altering an agent's state, an agent's itinerary, or an agent's security policy. For example, an agent may use a trusted host to determine the next host to visit, thereby altering its itinerary based on this interaction. The notional assumption is that trusted hosts act in the best interest of the security requirements of *both* agents and hosts—which define an agent application.

3.5 Defining the Trust Algorithm

Given the definitions for principals and trust relationships for mobile agents, we summarize the following constructs of a mobile agent application:

G Application G implementing owner's task or function
(H, A, φ)

DH Application owner's dispatching host

EH Set of all executing hosts

TH Set of all trusted hosts

H Set of all hosts
h_x: h is a *possible* target of migration for application G
$H = \{ DH \cup EH \cup TH \}$

A Set of all agents
a_y: a is a unique agent implementing functionality of application G

P Set of all principals
$P = A \cup H$

φ Set of all 4-tuple trust relationship mappings between principals in the system:
$\boldsymbol{\delta}(p_i, p_j) \rightarrow (f, l, t, s)$
f = foreknowledge (WK, K, UK)
l = level (HT, T, ND, U, HU)
t = timeliness (E, S, F)
s = security (H/A, R)

Trust relationships are linked to one or more security mechanisms (s) in our model. As part of the trust evaluation process, our model supports three different notions of trust acquisition: initial trust, first-hand trust, and recommended trust. Initial trust is the set of relationships which an agent or host has to begin with—before a history of interactions takes place over time. We argue that such a set of initial relationships can be generalized based on the application environment of the mobile agent and present three such models in section 4. Next, first-hand trust is that which is gathered over time through specific interactions between principals. Recommended trust comes when we accept or rely on trust levels offered by other principals.

General Trust Notion: *Mobile agent applications use stronger security mechanisms for principals with less trust/knowledge and use weaker mechanisms for principals with greater trust/knowledge*

Corollary: *The application environment determines trust levels which in turn dictate the initial set of requirements for security.*

We now discuss the notion of an application level security model for mobile agents.

4. Application Level Security Models

Even when using mechanisms that establish pure trust (such as tamper proof hardware), other assumptions must be taken into account to make sure security is guaranteed. Trusted hardware or multi-agent secure cryptographic protocols may be warranted or even feasible given certain application environment factors. When such mechanisms are not available, lower trust levels can be demanded and a more policy-driven approach can be required to make dynamic decisions about agent execution.

Models are used in many cases to help describe in more precise terms a particular set of relationships. Models in the security sense do several things such as: help test a particular security policy in terms of completeness or consistency, help document security policies, help conceptualize or design an implantation, or verify that an implementation fulfills a set of requirements [50]. We now present our trust framework for considering mobile agent applications and describe a model for viewing principles, actions, and trust relations in the mobile agent environment.

Almost a decade ago, Chess and his colleagues discussed three models of applications that itinerant mobile agents would be useful for [53]: information dispersal/retrieval, collaboration, and procurement. Poslad et al. in [54] proposed a series of application models for multi-agent interactions such as publisher directory, courier/broker, task allocation, multi-services domain, personalization, and mobile tasks. [54] also links security issues such as authentication, authorization, denial of service (DoS), and privacy with such application scenarios.

There are many other examples of mobile agent architectures (too numerous to name) that set forth and support specific application security goals. Fischmeister [55], for example, presents an analysis of the supervisor-worker framework as implemented via mobile agents and discusses the responsibilities and security aspects of the various principals in the system (agents, regions, places, agent systems, owners, and developers). We expound on these general ideas for describing agent-based applications and define the term *application model*. These describe interactions that can generalize security requirements for mobile agent applications. We introduce three such models which include the following: the military model, the trade model, and the neutral services model.

Application Model: *a **set of rules** that govern characteristics of **principals** in an application and a common **set of initial trust relationships** for a given set of security objectives*

There are three ways in which trust is formed in the mobile agent environment: initial trust, acquired trust, and recommended trust. We define an application model as a set of initial trust relationships that can be assumed for a given mobile agent application. It is obvious that applications have their own unique requirements, but many applications have a common purpose and environment which can be used to reason about trust relationships. An application model also has implicit understanding of how principals act towards one another and how principals are derived. We use application models to set boundaries on whether trust can be acquired over time—whether we can promote entities from unknown to known or whether we can ever promote entities from untrusted to unknown or trusted. We pose three such models next and expound how they can be used to initialize a trust management framework for mobile applications.

4.1 The Military Model

The military model is based upon the notion that a wall exists between friendly and adversarial entities. In this particular paradigm, a logical "Maginot Line" separates friendly and adversarial parties distinctly. Within the borders of the line, entities are essentially known to each other as authenticated, highly trusted principals. Much like the "Identification Friend or Foe" (IFF) signal that can be used to distinguish friendly and hostile parties in military scenarios, the military model presupposes that principals are accounted for and controlled within a single domain of control.

At some point, however, a given principal may be taken captive by an adversary. This captured entity could continue to function passively as a normal principal for the purpose of discovering information or leaking secrets. Captured entities can also become overtly malicious by delaying and denying service, corrupting friendly communications, and attacking privacy and integrity of group operations. Malicious

adversaries may also infiltrate and assume the role of an "insider" in this model and work either passively or aggressively to compromise security. In both cases, whether outsiders successfully infiltrate or insiders are successfully captured, groups of malicious parties could be interacting.

The military model (like other application sets we describe) assumes that applications have certain common characteristics among principals that are mapped to a given set of security requirements. First of all, hosts may arrange themselves in ad-hoc or mobile configurations, but they must come from a pool of systems that are verified by a managerial or supervisory entity within the environment. In other words, principals are managed, configured, tracked, and deployed in such a way that trust relationships are implicit as long as the identity of the principal can be verified. Unlike e-commerce trade models, principals are not necessarily complete strangers.

The military model fits requirements and trust relationships where the use of trusted third parties, trusted hardware, group security operations, multiple security levels, multiple trust levels, and distinct organizational structures exist. This environment is found in many corporate infrastructures (as well as the military itself) where a trusted computing based is not only possible monetarily but is mandated because of high concern for maintaining secrecy and integrity of information. Because of the implicit trust relationship among principals in this environment, hosts can also work in cooperation with agents to provide mutual prevention and detection services.

The military model also suggests a common set of agent characteristics. Agents in this case are designed, developed, and deployed by a centralized authority of *some* kind. In the corporate world, this could reduce to an outsourced development team on contract or an IT department with in-house programmers. This model represents programming environments where only approved or authorized mobile agent applications would be used to start with—and only those that have been certified or tested by an internal organization responsible for configuration management and software development. In other models, agents can take on adversarial roles with one another; in this model, however, agents are all considered as peers in terms of agency. Of course even departments within a corporation have proprietary or sensitive information that should be protected by confidentiality and integrity.

Agents in this regard still have security requirements, but in general, their identity, safety, authorization, and authentication can be verified within the circle of trust. The military model also places less emphasis on distinction between hosts which execute agents and hosts which dispatch agents. In this type of environment, agent servers are used interchangeably in some cases as the dispatcher and in other cases as the execution host for other agents. Military models represent many real world paradigms of computing where group collaboration, systems management, and information gathering are accomplished through agent-based applications. The key feature is that a centralized management domain exists. Typically, the military model also involves applications that have static (whether ordered or unordered) itineraries. Figure 1 summarizes the initial trust relationships that are associated with a military model security context.

	DH AO	EH	TH	A CD
DH AO	k / HT	k / T	k / HT	k / T
EH	k / T	k / T	k / HT	k / T
TH	k / HT	k / HT	k / HT	k / HT
A CD	k / T	k / T	k / HT	k / T

Figure 1: Military Model Trust Relation Matrix

Based on this initial trust relationship set, the trust algorithm can make dynamic determination concerning acquired trust or recommend trust over time in the model. It will primarily be through acquired trust mechanisms (the acquisition of negative trust) that infiltrators will eventually be discovered. These relationships also determine the security mechanisms required by host or agent. To describe the essence of the military model in the formal sense, we have the following bounds for initial trust relationships:

$DH \subseteq EH$
$TH \neq \varnothing$
$\forall\ p_i,p_j \in P: i \neq j$ and $\forall\ \boldsymbol{\delta}\ (p_i,p_j) \in \varphi$:
$\boldsymbol{\delta}(p_i,p_j) \rightarrow$ f = 'known' or f = 'well known'

There are two primary security-related tasks that consume the majority of time in the military model: 1) protecting insiders from outsiders and 2) detecting whether or not an agent, host, or trusted-third party has been compromised or captured. We assume that trusted third parties will not be compromised. The underlying assumption of the military model is that some agents or agent platforms will fall under "enemy control" or be subverted for use by "insiders" at some point in time. It's not a matter of "if" but rather "when" in this particular application scenario. As in any development organization, the possibility exists that an insider programmer can create and use malicious agents for their own benefit.

The question of security then becomes related to detection of anomalous or malicious behavior and removal of malicious parties from the circle of trust once they are detected. This scenario best represents application environments where there is a peer (non-adversarial) relationship within the community of trust of the mobile agent application—and applies to both agents and hosts. The role of trusted third-parties and trusted hardware, as well as coalition security mechanisms, becomes focused on identifying those principals that have violated the circle of trust or are attempting to gain access to the circle of trust (either through denial of service, falsification of authentication, etc.). We also distinguish a strong military model where all executing hosts are equipped with tamper-proof hardware and thereby have equivalent trust levels and behavior as that of trusted hosts.

4.2 The Trade Model

The trade model is designed to capture the intuition of a competitive interaction among actors that are all bargaining for resources. This concept could be termed an

economic model, a buy/sell model, or a supply/demand model where economic benefits are in view. This environment is indicative of the Internet model of computing where mobile agent applications might be deployed. It is the application environment where disjoint communities of mobile agent dispatchers want to use services or obtain goods from a set of host commodity or service providers. Agent literature routinely represents such a model as an agent dispatched to find an airline ticket among a group of airline reservation servers—accomplishing the transaction autonomously while finding the best price within the predefined constraints of the user.

As figure 2 illustrates the initial trust relationships for security requirements in the trade model and depicts the adversarial relationship among principals: buyers do not trust sellers to deal honestly with them, sellers do not trust other sellers to work for their best interest, buyers do not trust sellers to act non-maliciously, and buyers are in competitive relationships with other buyers for the same goods and services. The largest number of perceived mobile agent application possibilities typically fall into the trade model in terms of security requirements. The loosely formal rules in this model are expressed as follows:

$$DH \cap EH = \varnothing$$
$$TH = \varnothing$$
$$\forall\, p_i,p_j \in P: i \neq j \text{ and } \forall\, \boldsymbol{\delta}\,(p_i,p_j) \in \varphi:$$
$$\boldsymbol{\delta}(p_i,p_j) \rightarrow f >= \text{‘unknown’}$$

In this view, principals are not necessarily known before interaction takes place. Whether we are referring to agents or hosts, there is in most cases no trust or foreknowledge between users that want to execute agents and servers that would like to execute agents. It is also a model which mirrors security concerns on both sides. For instance, agent hosts that are considered “sellers” are as equally distrusting of agents originating from dispatching hosts of “buyers” or from other executing hosts that are sellers. Buyers see commercial hosts as untrusted in the sense that there is economic benefit to gain by altering the execution integrity of their agents.

	DH AO	EH	TH	A CD
DH AO	k / HT	k / U uk / HU	k / T uk / ND	k / ND uk / U
EH	k / U uk / HU	k / U uk / HU	k / T uk / ND	k / U uk / HU
TH	k / ND uk / U	k / U uk / HU	k / T uk / ND	k / ND uk / U
A CD	k / T uk / ND	k / U uk / HU	k / T uk / ND	k / ND uk / HU

Figure 2: Trade Model Trust Relation Matrix

4.3 The Neutral Services Model

As a third notion to capture application level security requirements, we define the neutral services model with the intuition that a service (or set of information) is acquired by one or more agents. Providers of services do not themselves have an adversarial relationship, but they may be viewed as having disjoint communities of

trust. The primary difference in the neutral services model and the trade model is that communities of hosts exist with no adversarial relationship among themselves. These communities are essentially *neutral* in terms of their commitments to each other—neither friendly nor hostile.

This model describes well application environments that are designed around information or database services. In this regard, providers of information typically have no economic gain from altering the results or influencing the itinerary of agents that they service. Hosts provide services honestly in the sense that they would not alter the path or intermediate data results of an agent. Service providers can and in most cases do charge a small fee for the use of their service, however. What might be of interest to a dispatching application owner in this model is whether or not its agent is billed correctly for services that are used. In this respect, if information providers charge for their service it is to their benefit to alter the execution integrity of an agent so that the agent is charged for more than was legitimately received.

Figure 3 illustrates the initial trust relationships that are derived in this particular model. Adversarial relationships exist between agents from the "client" community and hosts in the "server" community, but there is not necessarily a trust or distrust of hosts within a given community. Neutral hosts see no benefit from altering an agent that might be carrying results from other hosts or from preventing them from visiting other hosts. Hosts in this realm are in essence a "one-of-many" provider of information.

	DH AO	EH	TH	A CD
DH AO	k / HT	k / ND uk / U	k / T uk / T	k / T uk / ND
EH	k / ND uk / U	k / ND uk / ND	k / T uk / T	k / T uk / ND
TH	k / ND uk / ND	k / ND uk / ND	k / T uk / T	k / ND uk / ND
A CD	k / ND uk / ND	k / U uk / HU	k / T uk / T	k / T uk / ND

Figure 3: Neutral Services Model Trust Relation Matrix

This paradigm may not fit a search engine model where a mobile agent visits and collates search results from let's say Google, Yahoo, and Alta Vista. In that case, it may be of interest to one of these engines (who get benefit from every agent hit since advertisers might pay more for a more frequently visited search engine) to alter the itinerary or search results of other hosts. It might also benefit a search engine in this example to maliciously alter search results of other engines carried by the agent to be "less useful" so that their results look better to the end user. In that instance, the trade model would fit better in terms of security requirements.

The type of protection that is needed in the neutral services model revolves primarily around the execution integrity of the agent. To that effect, hosts that bill customers for usage might be tempted to cheat and wrongly charge agents for resources they did not use. Likewise, agents may want to falsely convince a host that no service or information was gathered, when in fact it was. Trusted relationships between neutral third parties are also more conducive in this environment and trusted third parties may interact with various communities of service providers themselves

on behalf of other users. The essence of the neutral services model in loosely formal terms is as follows:

$DH \cap EH = \varnothing$
$TH \neq \varnothing$ or $TH = \varnothing$
$\forall\ p_i,p_j \in P: i \neq j$ and $\forall\ \boldsymbol{\delta}\ (p_i,p_j) \in \varphi$:
$\boldsymbol{\delta}(p_i,p_j) \rightarrow f >=$ 'unknown'

5.0 Discussion

When application development is in view, it is often helpful to have methods which help transform requirements into implementation. We present initial work on a trust model to support development of mobile agent applications that links trust relationships and expression with both security requirements and mechanisms. We further define a set of initial trust relationships and generalize rules for principals that fit three different models of application security requirements. The usefulness of such models is that both developers and researchers can reason about security requirements and mechanisms from an application level perspective.

We believe this paper presents the foundation for more precise elaboration of trust models for mobile agents and helps tie the view of the application owner with the underlying mobile agent architecture. Future work will involve the exact specification of the trust algorithm in mathematical terms with analysis of security properties for specific types of security mechanisms and their effect on trust.

References

[1] D.M. Chess. Security Issues in Mobile Code Systems. In G. Vigna, editor, *Mobile Agents and Security*, *LNCS 1419*, pp. 1-14. Springer-Verlag, June 1998.

[2] W. Jansen and T. Karygiannis. National Institute of Standards and Technology, Special Publication 800-19-Mobile Agent Security, August 1999.

[3] E. Bierman and E. Cloete. Classification of malicious host threats in mobile agent computing. In *Proceedings of 2002 Annual Research Conf. of the South African Institute of Computer Scientists,* Port Elizabeth, South Africa, pp. 141 - 148, 2002, ISBN:1-58113-596-3.

[4] C.F. Cubillos and F. Guidi-Polanco. Security Issues on Agent-Based Technologies In *VIP Scientific Forum of the International IPSI-2003 CONFERENCE*, 2003.

[5] J. Zachary. Protecting Mobile Code in the Wild. *IEEE Internet Computing*, pp. 2-6, March/April 2003.

[6] P. Bellavista, A. Corradi, C. Federici, R. Montanari, D. Tibaldi. Security for Mobile Agents: Issues and Challenges. Chapter in the Book "Handbook of Mobile Computing", to appear, 2004.

[7] R. Yahalom, B. Klein, and T. Beth. Trust relationships in secure systems-A distributed authentication perspective. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pp. 150-164, 1993.

[8] M. Blaze, J. Feigenbaum, J. Lacy. Decentralized Trust Management. In *Proceedings IEEE Conference on Security and Privacy*, Oakland, CA, May 1996.

[9] T. Grandison and M. Sloman. A Survey of Trust in Internet Applications. *IEEE Communications Surveys*, 4th Quarter, 2000.

[10] M. Burmester and A. Yasinsac. Trust Infrastructures for Wireless, Mobile Networks. *WSEAS Transactions on Telecommunications*, Vol. 3, #1, January 2004, pp. 377-82

[11] L. Rasmusson and S. Janson. Simulated social control for secure internet commerce. In *New Security Paradigms Workshop*, Lake Arrowhead, CA, Sept. 1996. ACM Press, pp. 18-26.

[12] D. Karjoth, D. B. Lange, and M. Oshima. A security model for Aglets. *IEEE Internet Computing*, 1(4):68–77, July-August 1997.

[13] L. Kassab and J. Voas. Agent Trustworthiness. In E*COOP Workshop on Distributed Object Security and 4th Workshop on Mobile Object Systems Secure Internet Mobile Computations,* Brussels, July 20-21, 1998.

[14] N. Antonopoulos, K. Koukoumpetsos and K. Ahmad. A Distributed Access Control Architecture for Mobile Agents. In *Proceedings of the International Network Conference 2000,* Plymouth, UK, July 2000.

[15] W. Jansen. A Privilege Management Scheme for Mobile Agent Systems. In *Proceedings of the International Conference on Autonomous Agents*, Montreal, Canada, May 2001.

[16] L. Kagal, T. Finin, and Y. Peng. A delegation based model for distributed trust. In *Proceedings of IJCAI-01 Workshop on Autonomy, Delegation, and Control*, August 01, 2001.

[17] H. K. Tan, L. Moreau. Trust Relationships in a Mobile Agent System. In G. Picco, editor, *Fifth IEEE International Conference on Mobile Agents*, LNCS 2240, pp. 15-30, Atlanta, Georgia, Springer-Verlag, December 2001.

[18] V. Cahill, et al. Using trust for secure collaboration in uncertain environments. In *Pervasive Computing Mobile And Ubiquitous Computing*, 2(3):52-61, July-September, IEEE, 2003.

[19] M. Carbone, M. Nielsen, V. Sassone. A Formal Model for Trust in Dynamic Networks. In *Proc. of 1st International Conference on Software Engineering and Formal Methods (SEFM'03)*, Sept. 2003, pp. 54–63, Brisbane, Australia.

[20] L. Capra. Engineering Human Trust in Mobile System Collaborations. In *Proc. of the 12th International Symposium on the Foundations of Software Engineering* (*SIGSOFT 2004/FSE-12*). November 2004, Newport Beach, CA.

[21] S. Robles and J. Borrell. Trust in Mobile Agent Environments. In *Proc. of the 7th Spanish Meeting about Cryptology and Information Security*, Oviedo, 2002.

[22] D. Gambetta. Can We Trust Trust? In D. Gambetta, editor, *Trust: Making and Breaking Cooperative Relations*. Basil Blackwell, Oxford, 1990, pp. 213-237.

[23] H. K. Tan and L. Moreau. Extending execution tracing for mobile code security. In Fischer, K. and Hutter, D., Eds. *Proc. of 2nd Intl Workshop on Security of Mobile MultiAgent Systems (SEMAS'2002)*, pp. 51-59, Bologna, Italy, July 2002.

[24] B. Hashii, S. Malabarba, R. Pandey, and M. Bishop. *Supporting reconfigurable security policies for mobile programs.* In 9th International World Wide Web Conference (WWW9), Amsterdam, Netherlands, May 15-19, 2000.

[25] J. Borrell, S. Robles, J. Serra, and A. Riera. Security the itinerary of mobile agents through a non-repudiation protocol. In *IEEE International Carnahan Conference on Security Technology*, pp. 461-464, 1999.

[26] S. Robles, J. Borrell, J. Bigham, L. Tokarchuk, and L. Cuthbert. Design of a Trust Model for a Secure Multi-Agent Marketplace. In *5th International Conference on Autonomous Agents*, Montreal, May 2001, pp. 77-78. ACM Press.

[27] S. Robles, S. Poslad, J. Borrell, and J. Bigham. A practical trust model for agent-oriented business applications. In *4th International Conference on Electronic Commerce Research*, 2:397-406, Dallas, USA, November 2001.

[28] S. Robles, S. Poslad, J. Borrell, and J. Bigham. Adding security and privacy to agents acting in a marketplace: a trust model. In *35th Annual IEEE International Carnahan Conference on Security Technology*, London, October 2001, pp 235-239. IEEE Press.

[29] G. Navarro, S. Robles and J. Borrell. An Access Control Method for Mobile Agents in Sea-of-Data Applications. *Upgrade*. III (December 2002), 47-51.

[30] J. Ametller, S. Robles and J.A. Ortega-Ruiz. Self-Protected Mobile Agents. In 3*rd International Conference on Autonomous Agents and Multi Agents Systems*. ACM Press, 2004.

[31] L. Kagal, T. Finin, and A. Joshi. Moving from security to distributed trust in ubiquitous computing environments. IEEE Computer, December 2001.

[32] L. Kagal, T. Finin, A. Joshi. Developing secure agent systems using delegation based trust management. In In Fischer, K. and Hutter, D., Eds. *Proc. of 2nd Intl Workshop on Security of Mobile MultiAgent Systems (SEMAS'2002),* Bologna, Italy, July 2002.

[33] S. Robles, J. Mir, and J. Borrell. MARISMA-A: An architecture for mobile agents with recursive itinerary and secure migration. In *2nd Information Workshop on Security of Mobile Multiagent Systems*, Bologna, July 2002.

[34] B. Yee. Monotonicity and Partial Results Protection for Mobile Agents. In *Proceedings of 23rd International Conference on Distributed Computing Systems*, Providence, Rhode Island, May, 2003.

[35] S.R. Tate and K. Xu. Mobile Agent Security Through Multi-Agent Cryptographic Protocols. In *Proceedings of the 4th International Conference on Internet Computing (IC 2003)*, pp. 462-468, 2003.
[36] J. McDonald, A. Yasinsac, W. Thompson. Taxonomy for defining mobile agent security. Technical report. Dept. of Computer Science, Florida State University. Available, http://ww2.cs.fsu.edu/~mcdonald/pubs/MYT04-acmSurveys.pdf, Dec. 2004.
[37] K. Cartrysse and J.C.A. van der Lubbe. Secrecy in Mobile Code. In *25th Symposium on Information Theory in the Benelux*, Rolduc, Kerkrade, The Netherlands, June 2004.
[38] J. Claessens, B. Preneel and J. Vandewalle. (How) can mobile agents do secure electronic transactions on untrusted hosts? – A survey of the security issues and the current solutions. *ACM Transactions on Internet Technology*. February 2003.
[39] V. Roth. Empowering mobile software agents. In *Proc. 6th IEEE Mobile Agents Conference*, LNCS Volume 2535, pages 47–63. Spinger Verlag, 2002.
[40] T. Sander and C.F. Tschudin. Protecting mobile agents against malicious hosts. In G. Vigna, editor, *Mobile Agents and Security*, LNCS 1419, pp. 44-61, Springer-Verlag, 1998.
[41] J. Algesheimer, C. Cachin, J. Camenisch, and G. Karjoth. Cryptographic Security for Mobile Code. In *Proc. IEEE Symposium on Security and Privacy (S&P 2001)*, pp. 2-11, May 2001.
[42] M. Reed, P. Syverson, and D. Goldshlag. Anonymous connections and Onion Routing. *IEEE Journal on Selected Areas in Communication, Special Issue on Copyright and Privacy Protection*, 1998.
[43] D. Westhoff, M. Schneider, C. Unger, and F. Kaderali. Protecting a Mobile Agent's Route against Collusions. LNCS 1758, Springer, 1999.
[44] G. Knoll, N. Suri and J. M. Bradshaw. Path-Based Security for Mobile Agents. *Electronic Notes in Theoretical Computer Science*, Vol. 58, No. 2, 2002.
[45] G. Karjoth, N. Asokan and C. Gulcu. Protecting the computation results of freeroaming agents. In Kurt Rothermel and Fritz Hohl, editors, *Proc. of the Second International Workshop, Mobile Agents 98*, LNCS 1477, Springer-Verlag, pp. 195-207, 1998.
[46] P. Maggi and R. Sisto. A Configurable Mobile Agent Data Protection Protocol. *AAMAS'03,* Melbourne, Australia. July 14–18, 2003.
[47] D. Singelee and B. Preneel. Secure e-commerce using mobile agents on untrusted hosts. Computer Security and Industrial Cryptography (COSIC) Internal Report, May 2004.
[48] N. Borselius, C.J. Mitchell and A. Wilson. Undetachable Threshold Signatures. In *Cryptography and Coding - Proceedings of the 8th IMA International Conference*, Cirencester, UK. LNCS 2260, pp. 239-244. Springer-Verlag, 2001.
[49] J. Riordan and B. Schneier. Environmental key generation towards clueless agents. In G. Vigna, editor, *Mobile Agents and Security*, LNCS 1419, Springer-Verlag, 1998, pp.15–24
[50] Pfleeger, C. and Pfleeger, S.L. Security in Computing. Third Edition. Prentice Hall, Upper Saddle River, NJ, 2003. ISBN 0-13-035548-8.
[51] G.C. Necula. Proof-carrying code. In *Proceedings of the 24th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pp. 106-119, Paris, France, January 1997.
[52] P. Lincoln and C. Talcott. Towards a semantic framework for secure agents. High Confidence Software and Sytems (HCSS 2003). SRI International. March 16, 2003. Available http://www-formal.stanford.edu/MT/03secagent.pdf.
[53] D.M. Chess, B. Grosof, C. Harrison, D. Levine, C. Parris, and G. Tsudik. Itinerant Agents for Mobile Computing. *Journal IEEE Personal Communications*, 2(5):34-49, Oct. 1995.
[54] S. Poslad, M. Calisti, P. Charlton. Specifying Standard Security Mechanisms in Multi-Agent Systems. *AAMAS 2002 workshop on Deception, Fraud And Trust*, Bologna, Italy.
[55] S. Fischmeister. Building Secure Mobile Agents: The Supervisor-Worker Framework. Diploma Thesis, Technical University of Vienna, p. 6-12, Feb. 2000.
[56] K. Schelderup and J. Olnes. Mobile agent security – issues and direction. In H. Zuidweg et al. (editors), IS&N '99, LNCS 1597, pp. 155-167, 1999. Springer-Verlag Berlin Heidelberg, 1999.