Mobile Agents and Security

Michael S. Greenberg and Jennifer C. Byington, Theophany Holding
David G. Harper, Tufts University

ABSTRACT The practicality of mobile agents hinges on realistic security techniques. Mobile agent systems are combination client/servers that transport, and provide an interface with host computers for, mobile agents. Transport of mobile agents takes place between mobile agent systems, which are located on heterogeneous platforms, making up an infrastructure that has the potential to scale to the size of any underlying network. Mobile agents can be rapidly deployed, and can respond to each other and their environment. These abilities expose flaws in current security technology. This article surveys the risks connected with the use of mobile agents, and security techniques available to protect mobile agents and their hosts. The inadequacies of the security techniques developed from the information fortress model are identified. They are the result of using a good model in an inappropriate context (i.e., a closed system model in a globally distributed networking computing base). Problems with commercially available techniques include: 1) conflicts between security techniques protecting hosts and mobile agents, 2) inability to handle multiple collaborative mobile agents, and 3) emphasis on the credentials of software instead of on the integrity of software to determine the level of trust.

Any years ago, a computer called omega.univ.edu was decommissioned. The work it was doing was moved to other computers, and omega.univ.edu was turned off. A few years later, a new computer was purchased. The operators decided to reuse the old network name omega.univ.edu. They activated the new computer. To their surprise, electronic mail arrived, much of it over three years old. The mail had been stored “pending delivery” in mail relays on the Internet, waiting for omega.univ.edu to come back online.

Consider the same story with mobile agents in place of e-mail. These would not be rogue mobile agents, but would carry proper authenticating credentials, authorizing them to do legitimate work. Assuming their credentials had not expired, or there is an error in the local clock time, they would execute that work on arrival. The work would be done completely out of context, due to neither system failure nor abnormal procedure. One of these mobile agents might deliver an out-of-date password file, effectively locking out the operators. A nother might reconfigure the network interface, taking the computer off the net. A third might introduce an archaic upgrade to part of the operating system, thus destroying it. Imagine this scenario in mission-critical systems, such as the banking or telecommunications industries. Aachromisms introduced into telephone switches by out-of-context mobile agents could take down an entire network.

Mobile agents can be powerful tools, but, as the above anecdote illustrates, they can easily become destructive if proper safety precautions are not in place. We therefore face a conundrum: how do we simultaneously provide effective security techniques for both mobile agents and the hosts frequented by them? Before we can approach these issues we need to establish a common lexicon for the purposes of this article.

TERMS AND CONCEPTS

There is as of now no rigorous definition of the term agent [1]. For the purposes of this article, we define an agent as software which is autonomous, has one or more goals, a scope of competence, and that may, or may not, collaborate and communicate with other software and humans. Ideally, an agent is versatile and robust in changing environments. Agents can be programmed to work in cooperative teams. Team members may have different complementary specialties, or be duplicates of one another.

Mobility is not required for software to be considered an agent. Agents are mobile when designed to be transported from one device to another. They are similar to programs submitted via Remote Job Entry, or macros embedded in e-mailed documents. Contrary to popular belief, mobile agents do not transport themselves, but depend on mobile agent systems to move their binary images over a variety of media. A mobile agent system is a combination of a client and a server which, when located on a host computer, runs, sends, and receives mobile agents, and attempts to guard against mobile agents which attempt misuse. It provides or interfaces with the environment in which the mobile agent runs. To be effective, mobile agent systems must be deployed on all the devices to which a mobile agent may travel. Mobile agent systems may implement special network protocols, header formats, and security techniques. To leverage already ubiquitous technology, standards such as HTTP, MIME, or transport via e-mail are often used.

Mobile agents do not have a monopoly on mobility; all software is mobile. Whether loaded into RAM from a hard disk, network drive, CD-ROM, or the Web, or transported via a mobile agent system, all software is moved prior to execution (Fig. 1). This process can be automatic, as in a mobile agent system, or manual, such as in the installation of a word processor. The method and media do not matter, as computer viruses have proven.

Unlike agents, mobile agents are almost universally written in interpreted machine-independent languages, so they can run in heterogeneous environments. We shall call the interpreters for these languages execution layers. They may run to completion before being transported by one mobile agent system to another, or use special instructions to the mobile agent system to initiate transportation. Some mobile agent systems preserve the mobile agent’s execution state during its transport. An executing mobile agent is a continuously executing program, interrupted briefly during transport between a series of machines.

A host’s security system typically comprises three layers: mobile agent, operating system, and hardware. The operating system security provides a safety net for mobile agent system security techniques that have failed while running at low privilege levels, but not necessarily while running at high privilege levels. If there are two mobile agents executing in one execution layer, and one mobile agent violates the other’s memory...
space, the hardware security protections will not be triggered because the execution layer is a virtual machine and has its own memory management.

Mobile agents have high survivability. Some mobile agent systems deliberately enhance this survivability by implementing persistence, where a mobile agent is stored on disk while executing in case the computer crashes, so it can be automatically restored. Unlike other software, mobile agents can survive hardware failures simply by having moved before the machine goes down. Others survive by sheer numbers of duplicates deployed on a network. A mobile agent can be programmed to search for new paths if a network goes down between it and its goal. These features, desirable in a robust program, make mobile agents a significant security challenge.

A mobile agent system implements a number of security techniques, and may use other security techniques implemented in the execution layer. When a mobile agent system is running in the same execution layer as the mobile agents, all the security techniques are located in the execution layer. Within this framework, a reference monitor is the program or part of a program that controls a mobile agent’s access to information, services, and resources on a host computer. The reference monitor refers to a set of guidelines, called the security policy, to determine whether or not access should be granted to the mobile agent. The person responsible for the host computer defines the security policy in order to prevent potential misuse. Some mobile agent systems include security domains, which are a method to organize resources and services (files, printers, databases, etc.) into groups that have their own security policies and access lists. They make it easy to segregate a group of mobile agents with one access level from a group of other mobile agents with a different access level. Different security policies can be chosen specifically depending on where the agent is from and who sent it.

We shall examine these, and the security techniques currently employed to provide protection against intentional and unintentional misuse.

**MISUSE INVOLVING MOBILE AGENTS**

In this section, we will explain the ways that mobile agents and hosts may intentionally or accidentally misuse one another. There are two categories of misuse involving mobile agents:

- Misuse of hosts by mobile agents
- Misuse of mobile agents by hosts and other mobile agents

First, we will describe how a mobile agent can abuse the information, software, hardware, or resources of a host computer. This is an insidious form of attack because the user may never know that the mobile agent has visited the host computer. Then we cover how a mobile agent can be destroyed, stolen from, subverted, and trapped by other mobile agents and host computers. Types of attacks are often categorized as follows: damage, denial of service, breach of privacy, harassment, and social engineering, as defined below. In addition we include two more complex forms of attacks: event-triggered attacks and compound attacks. For each category we provide a definition, an explanation, and an example of an attack against a host and an attack against a mobile agent (Table 1).

**Damage** — Destruction or subversion of a host’s files, configuration, or hardware, or of a mobile agent or its mission.

- To hosts — A mobile agent can destroy or change resources or services by reconfiguring, modifying, or erasing them from memory or disk. When a mobile agent deliberately damages a mobile agent system, it inadvertently destroys all the other mobile agents executing there at the time. Examples of destruction include deleting or writing randomly into files, or ordering an “unscheduled” hardware upgrade to a host. An example of subversion is modification of the system configuration to change the security policy.

- To mobile agents — A host can destroy a mobile agent by erasing it, in the process losing anything it has gathered and possibly leaving the mobile agent’s work in an unstable state. A mobile agent can be subverted by selectively manipulating its code and/or data and caused to radically change its function. A host which fails to provide necessary resources to a mobile agent jeopardizes the mobile agent’s mission. Mobile agents sharing the same execution layer can attack each other. Since they are both in the same program memory space, any underlying hardware segment protections are bypassed. Many mobile agent system designers use one Java Virtual Machine (JVM) to execute multiple mobile agents; however, this can allow one Java thread to tamper with another.

**Denial of Service** — Partially or completely impeding one or more computer services, or a mobile agent’s access to some resources or services.

- To hosts — A network component that overloads a resource or service, such as by constantly consuming network connections, or a mobile agent blocks another process by overloading its buffers to create deadlock.

- To mobile agents — An execution layer may fail to provide access to system resources or services (files, network, etc.). For example, if a host fails to give a mobile agent network access, present mobile agents will be stranded.

**Breach of Privacy or Theft** — Illegal or undesired access, or removal of data from a host or a mobile agent.

- To hosts — A mobile agent accesses and steals private information, for example, secretly recording the input of a computer’s microphone, then transmitting it over a network to an unauthorized site. A nother form of theft uses...
covert channels to transmit data in a hidden way that violates a host’s security policy [2, 6]. For example, a mobile agent can use a covert timing channel, where it alternates its state (busy/idle), to signal a binary pattern to a collaborator who is monitoring the mobile agent’s CPU consumption rate.

- To mobile agents — At any point when visiting a host, a mobile agent is at risk of having portions of its binary image (e.g., monetary certificates, keys, secrets) copied unless it is encrypted. Since it has to be decrypted to execute, execution provides a window of vulnerability.

Harassment — Annoying people with repeated attacks.

- To hosts — A mobile agent programmed to attempt to offend people can display unwanted pictures, or write to the screen at intervals that cause the display to flicker at a frequency known to induce seizures in sensitive people.

- To mobile agents — A mobile agent afield can be attacked in ways that will bother its sender. Examples include delaying the mobile agent, or tracking a mobile agent to discover information about it or its sender.

Social Engineering — Manipulation of people, hosts, or mobile agents by using misinformation, coercion or other means.

- To hosts — A mobile agent can play a role in social engineering. For instance, it could request users’ passwords under false authority of the system administrator. For example, a mobile agent could travel to the system administrator’s computer and then replicate 100 times. Next, the mobile agents could go to 100 employees’ computers with the appearance of stemming from the system administrator. Each of them could then concurrently ask the employees for their passwords, and then disappear with revealed passwords before word could be passed that a fraudulent mobile agent had been detected (if it were, in fact, detected).

- To mobile agents — Mobile agents can be given misinformation with the objective of manipulating them or their senders. Hosts can misdirect mobile agents to unwanted destinations. One scenario involves hosts sending purchasing mobile agents only to collaborating vendor’s hosts.

**COMPLEX ATTACKS**

**Logic Bomb or Event-Triggered Attack** — Initiation of any of the previously described attacks based on an external event.

- To hosts — A logic bomb “goes off” when code, concealed within an apparently benign mobile agent, is triggered by a specific event, such as time (e.g., the Columbus Day virus), location, or the arrival of a specific person. This type of attack is the hallmark of a Trojan Horse program. An example is a mobile agent within the perimeter of a firewall using a modem to periodically give access to collaborators on the Internet.

- To mobile agents — Mobile agents can be attacked for something they are carrying, or because they are from a specific sender.

<table>
<thead>
<tr>
<th>Locale of misuse</th>
<th>Damage</th>
<th>Theft</th>
<th>Denial of service</th>
<th>Harassment</th>
<th>Social engineering</th>
<th>Logic bomb</th>
<th>Compound attack</th>
</tr>
</thead>
<tbody>
<tr>
<td>Host machine misuse</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Display</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>CD ROM</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Internal data bus</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hard/floppy disk</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Keyboard</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Microphone</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Speakers</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Printer</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>RAM</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ROM</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Physical network wire</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mobile agent misuse</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RAM</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Disk files</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Execution layer</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Physical wire</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 1. Locale of misuse involving mobile agents — possible misuses are open sets since new attacks can always be created and errors are by nature unpredictable.
Compound Attack — A compound attack is composed of multiple attacks that can achieve more than a single attack.

- To hosts — Using cooperation techniques from agent research, mobile agents can collaborate with each other in order to commit a series of attacks. For instance, harassment has been labeled annoying [5], but can be used as a part of social engineering with the ultimate aim of damage or breach of privacy.

- To mobile agents — Hosts or mobile agents can track or stalk a mobile agent with the intent of stealing from it, discovering its source or destination, and so on. Multiple collaborating hosts can track the mobile agent until it has something of interest, then attack it.

The biggest actual threat is a compound attack strategically aimed toward a goal. Collaborating mobile agents can easily launch a compound attack. Simple examples of collaborative mobile agent attacks include:

- A denial-of-service attack by a mobile agent need only be momentarily successful to expose a weakness a collaborator mobile agent can exploit (Fig. 2).
- A team of mobile agents within a firewall can set up bogus servers (for information dissemination, data back-up, etc.) while other mobile agents search out vulnerable hosts, or reconfigure computers to use those servers.
- A popular security scheme implemented in Web browsers, where an “unsigned” downloaded applet is allowed network access to its source computer, can be frustrated when a mobile agent is on the source computer to receive and carry, or transmit, the applet’s communication elsewhere.

The scenario in Fig. 2 includes network hosts α, β, δ, ε, and φ, with two collaborating malicious mobile agents posing as network management agents. They are temporarily located on β and δ, and have been granted minimal access to network resources.

1. α broadcasts an auto-configuration request. The recipients of this broadcast include the auto-configuration server on β and the two malicious mobile agents.
2. The malicious agent on β interferes with the auto-configuration server, possibly by flooding the local network interface with traffic.
3. The malicious agent on δ sends an almost normal auto-configuration reply to α, except that it contains the wrong name server address. α accepts the malicious agent’s configuration reply.
4. After a short while the malicious agent on β stops interference, allowing the auto-configuration server to reply. The reply is ignored by α, which is no longer listening on that port.

α is now receiving subtly subversive name service, which facilitates further security violations. The malicious mobile agents can depart to search for other vulnerable machines. They leave little or no trace. The malicious mobile agents used an event-triggered attack, coordinating their activities based on an external event: the auto-configuration request. This requires no overt communication and is hard to detect. Once access to local network resources has been granted, there is little or no way of detecting their nefarious activities.

**Figure 2. Collaborative mobile agents violating security.**

---

**SECURITY TECHNIQUES**

In this section we will define, describe, exemplify, and briefly critique the security techniques that mobile agents and hosts use to protect themselves. First we will cover the techniques designed to protect hosts, and then those to protect the mobile agents.

**Security Techniques Protecting Hosts**

The following techniques are designed to protect hosts: authenticating credentials, access-level monitoring and control, code verification, time limits, range limits, duplication limits, and audit logging. These techniques are all based on the information fortress model [7]. They seek to protect hosts by maintaining a closed system accessed through well-defined and regulated interfaces. When considering which host protection technique should be included in a mobile agent system to protect a host, one cannot assume the security features of any single operating system. Since mobile agents are intended to execute on different types of platforms, one must provide for the worst possible case. For this reason, mobile agent system security often duplicates, and augments, operating system security.

**Authenticating Credentials** — A mobile agent is digitally signed by one or more parties using one of a number of algorithms, such as a public key signature algorithm. Depending on the algorithm, this process either generates a certificate, which is then appended to the mobile agent’s binary image for use as a credential, or encrypts the mobile agent’s binary image for privacy during transportation. In both cases, digital signatures can be used to verify the identity of the mobile agent’s author and of its sender, where and when it was sent, and that it has not been tampered with in transit. Although many mobile agent systems implement this functionality, some execution layers implement it as well (Java, GCJ, etc.).

Authenticating credentials do not guarantee that the mobile agent will be harmless, or even useful. Forgery, cryptanalysis, theft of cryptographic keys, or poor implementation can compromise cryptographic techniques. However, even if genuine, credentials only provide assurance that someone...
else vouches for the mobile agent. In other words, the recipient is basing his or her trust in a mobile agent's integrity on another person's word, not on knowledge of the mobile agent. A person needs to know that a mobile agent has been proven to do what it is supposed to do, as well as that it is intact. Trust in software does not guarantee that software will be harmless on execution.

Access-Level Monitoring and Control — A reference monitor is used to restrict the information, system resources, and services that mobile agents are allowed to access and use. The reference monitor grants permission to mobile agents based on their level of authorization as shown by their authenticating credentials. A mobile agent, attempting to read a file it is not authorized to access, will be stopped by the reference monitor. The reference monitor is called a security manager in the Java runtime and a master interpreter in Safe-Tcl. The reference monitor consults a security policy to determine if access is to be granted. A access to restricted files, communications, peripheral devices, system configuration, information, and so on. Some mobile agent systems put a restriction on the number of times an object can be accessed [3, 8].

Access-level monitoring and control places restrictions directly on what a mobile agent can do. This is effective because it directly controls a mobile agent’s activity. A mobile agent cannot cause harm if it is not allowed to do anything harmful. A mobile agent without access is useless, so an authenticating credential is used to select which access privileges to grant. This approach is flawed because credentials cannot guarantee that a mobile agent will not cause harm, for the reasons stated above. It does guarantee that a mobile agent will only have access to certain commands and tools with which to cause harm. The access level only scratches the surface of the activity of any mobile agent, and will not detect many covert channels (Fig. 3). The mobile agent system administrator has to choose whom to trust concerning the question of reliability. The reliability of a mobile agent cannot be fully known. A authenticating credential state where an agent comes from and whether it has been tampered with on its route. Research is being done on proof-carrying code, which carries a "safety proof that attests to the code's adherence to a previously defined safety policy" [9]. These techniques cannot accurately vouch for the integrity of the mobile agent because they cannot provide enough information. In other words, unless the mobile agent has been tested on the present computer, there is no way of knowing if its software will cause harm to its new environment.

The flow diagram and abbreviated instruction chart in Fig. 3 depict the operation of a covert timing channel potentially employed by a mobile agent. Consider a host that is using an "inside" security model: the agent is allowed to access files, but not allowed to transmit anything over the network. The agent iterates through the bits of restricted information and uses an alternate method — consumed CPU cycles — to signal each bit's binary state to an observer. The colored boxes show the points of execution monitored by access-level monitoring and control security. The unocolored boxes show points of execution that are neither monitored nor detected by this technique. This simple example can be made harder to detect by obfuscating the data or adding noise to the channel.

Code Verification — The binary image of a mobile agent is scanned to determine if the mobile agent is a valid program. A code verification program, which finds illegal instructions in a mobile agent's code, will not invoke the execution layer. This technique is relevant for execution layers that are vulnerable to subversion or sabotage by mobile agents executing

---

**Figure 3.** Covert timing channel — one example of a security violation due to unmonitored actions.
within them. One example is the Java runtime environment, which includes a “byte-code verifier” to ensure a Java-based mobile agent will not perform illegal instructions, such as writing out of its memory space [10]. The Safe-Tcl execution layer, by contrast, does not require code verification [11].

Code verification only works as well as its implementation. It is slow, and not always used. The byte-code verifier does not check the Java code residing in a special area of a host’s disk called the classpath. The assumption is that this code is from a trusted source, or has already been checked when it was downloaded through the Java class loader and byte-code verifier. Nevertheless, the code is vulnerable to subversion where it resides, or may have arrived through a non-Java-controlled channel (such as an FTP server).

**Limitation Techniques** — The following three techniques are a natural extension of the need to control the persistent survivability of mobile agents.

**Time Limits** — The amount of time, elapsed or absolute, that a mobile agent system allows a mobile agent to run in an execution layer. A mobile agent that has been roaming the network past its relevant life span can be destroyed, or returned to its origin because of an encryption-protected timestamp embedded in its corpus.

**Range Limits** — The number of destinations or network “hops” that a mobile agent system will allow a mobile agent to be transmitted. Travel may be restricted to hosts listed on the mobile agent’s itinerary, such as only hosts within an organization.

**Duplication Limits** — The number of times that a mobile agent system will allow a mobile agent to be transmitted, or to transmit another mobile agent. For instance, a mobile agent duplicating recursively can easily swamp the hosts on a network.

Time, range, and duplication limits are effective techniques to control mobile agents and prevent them from “running wild,” because they directly control what a mobile agent does. The limitations imposed cannot be chosen for every eventuality, and may interfere with a mobile agent’s ability to complete its mission.

**Audit Logging** — Recording all mobile agent activities. An audit trail is kept so that after abuse is detected, the responsible party can be identified and called to account. An audit trail is essential for forensic examination. Unfortunately, this is not foolproof because all possible activities cannot be logged.

These host protection techniques are applied by the mobile

---

**Table 2.** Host protection techniques applied at each stage of mobile agent work.
agent system during the cycle of mobile agent work in the following sequence (Fig. 4, Table 2). Note that audit logging is done at each stage of the cycle.

1. **Arrival** — The mobile agent is transported by one mobile agent system to another on a different computer. Before accepting the mobile agent, the mobile agent system verifies the authenticity of its credentials. The reference monitor will make decisions based on the credentials, such as whether to accept the mobile agent, which security policy to use, or if the mobile agent will be allowed within a firewall. Before the execution layer starts the mobile agent, it may scan the mobile agent with code verification techniques to ensure its integrity.

2. **Execution** — While the mobile agent is executing, the reference monitor will enforce the security policy. Depending on its capabilities, it will monitor and control accesses to system resources (files, network, process control, communication channels, mobile agent transmission channels, etc.), and limit the time the mobile agent is allowed to run.

3. **Departure** — When a mobile agent departs from a computer, it may carry new or modified data. The mobile agent system may apply encryption to protect the integrity of this data in transit (although cryptographic key distribution issues can complicate this). The reference monitor may restrict where this mobile agent is allowed to travel, and if it is allowed to duplicate.

### SECURITY TECHNIQUES PROTECTING MOBILE AGENTS

There are two categories of mobile agent protection techniques: fault tolerance and encryption. The techniques based on fault tolerance aim to make mobile agents robust in unpredictable environments, thus protecting them from malfunctioning hosts, intermittent network connectivity, and so on. The techniques based on encryption hide the mobile agent, code, or sensitive data so that it cannot be recognized and thus will be less likely to be destroyed, stolen, or otherwise misused (Table 3).

#### Techniques Based on Fault Tolerance

- **Replication and Voting** — This technique ensures a mobile agent gets to its destination intact [12]. A mobile agent traveling through a network replicates at each node with the nodes only letting past intact mobile agents (a form of fault masking). A shared secret carried by mobile agents is one method to determine if they are intact. This technique can be used to ensure a critical message is delivered through a network of potentially faulty or malicious hosts. It is appropriate for tasks that can be safely duplicated (e.g., message delivery). In this scheme mobile agents are expendable, and impostor mobile agents can be tolerated.

- **Persistence** — This technique involves temporary storage of a running mobile agent and its execution state against host failure. When a mobile agent arrives on a host it uses a persistent storage facility in the mobile agent system (on hard disk) to store its binary image for the duration of its visit. If the host crashes, destroying the executing mobile agent, the copy persists in storage. When the host returns to service it restarts the mobile agents in the persistent storage. Although this makes mobile agents tolerant of host failures, it can cause an unwanted duplicate mobile agent if the host is down for an extended time and the mobile agent, presumed to be destroyed or forgotten, is replaced.

The techniques of elapsed time limits and persistence work against each other. Valuable time is wasted when a host crashes, which may cause a mobile agent to time out before it can complete its work. For instance, a mobile agent is given 10 minutes to complete a task involving visiting eight hosts. The first host crashes while the mobile agent is resident. Five minutes later the first host comes back online, and restarts the mobile agent from the persistent storage. Now the mobile agent doesn’t have enough time to complete its job, and times out while on the sixth host. This would leave the work in an incomplete state, and it is possible that the administrator might not be notified. A cumulative time limit is more appropriate.

- **Redirection** — Mobile agents finding new paths around damaged hosts or networks to complete a mission. When a mobile agent attempts to travel to a destination and fails to open the communication channel (i.e., Ethernet), it can be programmed to try other channels (i.e., packet radio) to attempt to reach its destination. This is useful in mission-critical situations where the network is unstable.

#### Techniques Based on Encryption

- **Sliding Encryption** — A mobile agent uses this technique to encrypt acquired data by using a public key [13]. The key is public, so theft is not an issue. Decryption can only be performed with the corresponding private key. The mobile agent uses sliding encryption to hide what it is carrying, so potentially malicious hosts cannot steal any data.

- **Trail Obscuring** — Changing a mobile agent binary image to make it hard to identify by pattern matching [13]. A mobile agent attempts to obscure its path through the network by constantly modifying its own binary image so that it cannot be identified as the same mobile agent by different hosts which are colluding in an attempt to track the mobile agent. This works in a situation where anonymity is required, such as an anonymous monetary donation or auction bid. It may also aid in surviving malicious hosts trying to stop specific behavior that can be identified by analyzing the mobile agent’s path.

- **Code Obfuscation** — A method to obscure a mobile agent’s code to make it hard to reverse engineer. There are many forms of code obfuscation. One theoretical but promising method is called black box security [14]: when an encrypted mobile agent is sent by a mobile agent system, a special execution layer is sent with it to run on each host the mobile agent visits, and to execute the encrypted mobile agent while it is encrypted. The host never has direct access to the code or data of the mobile agent. This deters theft or subversion, but not destruction. This technique makes it impossible for mobile agent systems to verify the mobile agent’s code, and is therefore very risky from a host perspective.

- **Encrypted Data Manipulation** — In this technique, the binary data of a mobile agent is encrypted in a way that allows it to be manipulated while still encrypted [15]. This allows the mobile agent to carry binary data that cannot be read by a host. The data is decrypted when the mobile agent returns to its sender. This technique deters theft or nondestructive tampering, and allows the mobile agent’s code to be inspected, and thus is not very risky from a host perspective.
State Appraisal Functions—These methods are used to ensure that a mobile agent’s unencrypted dynamic data is not tampered with [16]. State appraisal functions can be merged into a mobile agent’s binary image by its author and sender to verify that acceptable parameters are used during execution. Encrypting or digitally signing a mobile agent is simple if it has one destination, but complex if it roams to many hosts. Because a roaming mobile agent’s execution state can change as it runs, a portion of its data cannot be encrypted and is therefore vulnerable to subversion. A normally harmless mobile agent can be made dangerous by manipulating its execution state. These functions reside in the encrypted or signed portion of the mobile agent, and therefore have some measure of protection from tampering. The mobile agent and the host can both use these functions to determine if the mobile agent has been subverted. Consider a mobile agent searching for the best price for an apple. It may visit many fruit vendors’ hosts looking for the best price before making a purchase. One vendor’s host may modify the mobile agent’s execution state data to cause it to look for 100 apples rather than 1, in an attempt to make any competitor’s asking price appear larger. State appraisal functions, which check values in the mobile agent, can detect this type of abuse.

**Discussion**

Host protection techniques used in mobile agent systems are well known because they have their roots in the traditional information fortress model, where a trusted computer system is protected behind an invulnerable barrier. These systems have the following host protection techniques in common. Encryption is used to ensure the authenticity, integrity, and secrecy of data, communication, programs, and users. Security policy enforcement mechanisms are used to manage compartmentalized access to information and resources. Access limits are imposed on users and their programs. Audit logs are made to track system activity.

On the other hand, mobile agent protection techniques are still in their infancy. The problems brought out by trying to protect mobile agents and hosts, however, point to deficiencies in the classic security models we have. To make matters worse, information-fortress-model-based host protection techniques are in direct conflict with many mobile agent protection techniques. Once a mobile agent is within a host it cannot be protected from destruction, denial of service, and so on. Thus, many mobile agent protection techniques attempt to make the mobile agent hard to misuse without detection. The state of the art host protection techniques are concerned with treating each agent as an isolated threat. They are not, as yet, designed to handle collaborative attacks. For example, there is no commercially available host protection process that tracks the path of potentially harmful mobile agents as they travel between computers. Tracking is sometimes implemented for the purposes of inter-mobile agent communication, but not for real-time protection. Tracking is also done in some distributed intrusion detection systems under research. Some implementations embed a record within the mobile agent of where it has been. This allows the possessor of a mobile agent to determine its travel history, but does not facilitate location.

A flaw that all of these host protection techniques have in common is that they are based on the information fortress model. In this model a clear security perimeter was maintained around an organization’s computers and information. Infor-
information fortresses were created, validated, and operated by a cadre of highly trained and screened professionals, with a significant support infrastructure. Much of the basis for trust of an information fortress’ trusted computing base was founded on confidence in these people. This model is no longer viable due to the changes that have occurred in the computing environment since the time it was developed.

“The fundamental principles of the information fortress model are:
1. Policy, enforced by the system, protects resources from unauthorized manipulation.
2. Integrity of the physical system and its code guarantees that policy is enforced.
3. Secrecy of crypto keys and sensitive data underlies policy enforcement mechanisms.

“... No viable security system design can be based on the principles of Policy, Integrity, and Secrecy, because in the modern world Integrity and Secrecy are not achievable and Policy is not manageable” [7].

Today, any programmer can create a mobile agent. Non-programmers will send and receive mobile agents, and operate mobile agent systems. When people choose to use mobile agents, they do so on the basis of trust in the author of the software and the person who has sent it to them. As a mobile agent travels further from its source, the identities of the author and original sender become obscure. The mobile agent may be passed on by a number of senders. It becomes impossible to know whom one is trusting. The potential for error is staggering.

We agree with Ken Thompson’s assertion that “You can’t trust code that you didn’t totally create yourself. (Especially code from companies that employ people like me.) No amount of source-level verification or scrutiny will protect you from using untrusted code” [17]. We extend this to say you can’t even trust code you created yourself. The fact that a mobile agent carries authenticating credentials says that someone (the sender, author, etc.) was willing to vouch for its integrity at the time they digitally signed it. It does not mean that the mobile agent will work now, or on any specific machine, because the modern computing environment changes constantly.

An author of a mobile agent cannot anticipate where or when the code will be run, especially if it’s a mobile agent that travels to many different environments. Senders will not have access to mobile agent source code, nor would they generally know how to review it. With code reuse techniques it is likely that no single person will know what a mobile agent will do. A venerable mobile program, the notorious “form virus,” is a good example. On the operating system it was designed to run over, it periodically emits a harmless but annoying “click.” When running on a subsequent release of that operating system, it destroys parts of the hard disk’s format.

The subtle chaos that could result from the accumulation of small errors in mobile agent management and operation would be insidious and expensive. This is the largest problem connected with the use of mobile agents. It is likely that it would be more damaging on a regular basis than intentional hostile attacks. The scenarios we referred to in the introduction are examples of this type of problem.

Most mobile agent systems are currently implemented as independent entities, and do not communicate information about mobile agents. Therefore, each mobile agent system must independently detect harmful mobile agents through the process of being attacked after the mobile agent has been allowed onto the host. This issue would be addressed by information sharing among mobile agent systems. The authors and others are researching these approaches [18]. This would be similar to the CERT advisories posted on the Internet about known security threats. Secure data communication could be used, and harmful mobile agents “fingerprinted,” possibly using a message digest of the mobile agent’s binary image. A standard way to identify the severity of harm would also be necessary for warnings to be useful to multiple mobile agent systems with different security policies.

A combination of fault tolerance and encryption techniques could be used to build robust mobile agents. Some of the most promising approaches we’ve seen treat the mobile agent as expendable, using duplication or retransmission to tolerate abuse and failure. These mobile agents would have extremely high survivability. This would be wonderful for mission-critical applications, as long as a legitimate authority had a reliable method to stop the mobile agents. If not, controlling the mobile agents, or even locating them, would be a nightmare. If they had access to a large network, there would be the risk of creating unstoppable “worms.” From a host security perspective, this type of mobile agent would be dangerous. These types of mobile agents must be made securable from a host perspective before they become practical for realistic applications. The two security efforts of protecting hosts and mobile agents must be coordinated if mobile agent security is to mature and mobile agents are to be considered safe enough for wide deployment.

CONCLUSIONS

Security involving mobile agents remains in its infancy. Much good work has been done in this area. Unfortunately, most host protection techniques are based on the information fortress model. These techniques are now insufficient on their own for most applications due to a rapidly evolving computing environment, which has expanded beyond the scale that humans can easily track. Mobile agent traits augment production of all kinds, including the production of security breaches. Thus, mobile agents bring attention to security problems that already exist throughout the computing environment.

Current mobile agent security measures are not adequate because they are not geared toward software that is mobile, works cooperatively, interacts with its environment, and reacts unpredictably to unexpected events (e.g., software flaws, human error). Most currently used host protection techniques concentrate on identifying the source of the mobile agent software, limiting the mobile agents range and duration or operation, or verifying that it is a valid program. All of these security measures are effective to some degree, and the use of them should be retained. Access-level monitoring is the only form of security that monitors and controls the activity of the mobile agent itself. It only monitors the mobile agent at various points along the mobile agent execution path, so it is not sufficient to prevent security violations from happening between those monitored points.

Mobile agent systems must be able to continuously monitor, control, and report on the flow of information and execution within a running agent from the mobile agent’s inception, through its various changes in locale, to its completion. Ideally, in the future technology will exist to allow an operator to change or repair a mobile agent’s instructions during execution. Once mobile agent systems can do this, communication and collaboration between mobile agent systems become more useful because information about harmful activity can be more specific. As mobile agent technology evolves, and mobile agents become sophisticated, a cooperative security infrastructure can be developed. Eventually, this infrastructure
could expand to parallel globally deployed mobile agent systems using a standardized communication language, protocol, and context.

The difficulty of protecting mobile agents and the computers on which they run is in direct proportion to the promise they hold. Adequate protection is not impossible, it just remains to be conceived. Profitable directions for further development of mobile agent system security are:
- Tracking of mobile agent locations while the mobile agent is running
- Inter-mobile-agent-system communication and collaboration
- Harmful pattern monitoring and identification
- Mutual authentication between a mobile agent and its host
- Monitoring the flow of information and execution of a running mobile agent

The following ideas could be useful for mobile agents that run in execution layers within which an operator cannot monitor the full flow of information and execution, and therefore cannot be aware of hidden activity such as covert channels:
- Tracking of mobile agent activity for intermittent human observation
- Notification of mobile agent infiltration of a computer or firewall

Any one of these approaches is unlikely to be able to cover all of the risks inherent in the use of this technology. However, integrating and amalgamating these and other approaches and ideas will allow a comprehensive solution that will not only protect mobile agents, but will also solve security problems of software in general.

ACKNOWLEDGMENTS
Many thanks are due Ahmed Karmouch, Radia Perlman, Larry Backman, Robert Gray, Bob Strong, Richard Johnston, and Phil Gross for their invaluable support. The authors deeply appreciated the insightful criticisms and suggestions of the three anonymous reviewers. In addition, the authors thank Mary MacNicol, and Gavin, Liam, and Christopher Harper for their invaluable support.

REFERENCES

BIographies
MICHAEL S. GREENBERG (mgreenberg@acm.org) received his B.A. in computer science from Hampshire College in 1982. He is a founding member of the board of directors of the Agent Society. He was a consulting engineer at FTP Software (1988–1998) where he introduced mobile agent technology. His work with mobile agents began in 1983 with the late Calvin N. Mooers. His interests include distributed and decentralized systems, software engineering, self-modifying symbolic manipulation languages, and security.

JENNIFER C. BYINGTON (jbyington@theophanyholding.mv.com) received her B.A. in writing and psychology from Hampshire College. She worked with the primary author on the architecture of a distributed operating system and various mobile agent applications. Her current research interests include the relationship between the use of computers and human perception, and security models applicable to remote computing and self-modifying code.

DAVID G. HARPER (david@phoenix psy.tufts.edu) is a graduate student in the Department of Psychology at Tufts University where he also serves as a system administrator. He specializes in the mathematical modeling of complex functions in behavior and physiology. His interest in agents extends from work he did with the primary author on the architecture of the initial release of FTP Software’s agent product.