NIST Special Publication 800-19 – Mobile Agent Security

Wayne Jansen, Tom Karygiannis
National Institute of Standards and Technology
Computer Security Division
Gaithersburg, MD 20899
{jansen, karygiannis}@nist.gov

Abstract: Mobile agent technology offers a new computing paradigm in which a program, in the form of a software agent, can suspend its execution on a host computer, transfer itself to another agent-enabled host on the network, and resume execution on the new host. The use of mobile code has a long history dating back to the use of remote job entry systems in the 1960's. Today's agent incarnations can be characterized in a number of ways ranging from simple distributed objects to highly organized software with embedded intelligence. As the sophistication of mobile software has increased over time, so too have the associated threats to security. This report provides an overview of the range of threats facing the designers of agent platforms and the developers of agent-based applications. The report also identifies generic security objectives, and a range of measures for countering the identified threats and fulfilling these security objectives.

Table of Contents

1. Introduction

Over the years computer systems have evolved from centralized monolithic computing devices supporting static applications, into client-server environments that allow complex forms of distributed computing. Throughout this evolution limited forms of code mobility have existed: the earliest being remote job entry terminals used to submit programs to a central computer and the latest being Java applets downloaded from web servers into web browsers. A new phase of evolution is now under way that goes one step further, allowing complete mobility of cooperating applications among supporting platforms to form a large-scale, loosely-coupled distributed system.

The catalysts for this evolutionary path are mobile software agents – programs that are goal-directed and capable of suspending their execution on one platform and moving to another platform where they resume execution. More precisely, a software agent is a program that can exercise an individual's or organization's authority, work autonomously toward a goal, and meet and interact with other agents. Possible interactions among agents include contract and service negotiation, auctioning, and bartering. Agents may be either stationary, always resident at a single platform; or mobile, capable of moving among different platforms at different times. The reader is referred to [1] for a more extensive introduction to the subject. This paper focuses mainly on the security issues that arise when mobility comes into play.

A spectrum of differing shades of mobility exists, corresponding to the possible variations of relocating code and state information, including the values of instance variables, the program counter, execution stack, etc. [2]. For example, a simple agent written as a Java[1] applet [3] has mobility of code through the movement of class files from a web server to a web browser. However, no associated state information is conveyed. In contrast, Aglets [4], developed at IBM Japan, builds upon Java to allow the values of instance variables, but not the program counter or execution stack, to be conveyed along with the code as the agent relocates. For a stronger form of mobility, Sumatra [5], developed at the University of Maryland, allows Java threads to be conveyed along with the agent's code during relocation. The mobile agent systems under discussion in this paper involve the relocation of both code and state information.

2. Security Threats

Threats to security generally fall into three main classes: disclosure of information, denial of service, and corruption of information. There are a variety of ways to examine these classes of threats in greater detail as they apply to agent systems. Here, we use the components of an agent system to categorize the threats as a way to identify the possible

[1] Certain computer software and hardware products and standards are identified in this paper for illustration purposes. Such identification is not intended to imply recommendation or endorsement by the National Institute of Standards and Technology, nor is it intended to imply that these computer software and hardware products and standards identified are necessarily the best available.

source and target of an attack. It is important to note that many of the threats that are discussed have counterparts in conventional client-server systems and have always existed in some form in the past (e.g., executing any code from an unknown source either downloaded from a network or supplied on floppy disk). Mobile agents simply offer a greater opportunity for abuse and misuse, broadening the scale of threats significantly.

A number of models exist for describing agent systems [2, 6, 7]; however, for discussing security issues it is sufficient to use a very simple one, consisting of only two main components: the agent and the agent platform. Here, an agent is comprised of the code and state information needed to carry out some computation. Mobility allows an agent to move, or hop, among agent platforms. The agent platform provides the computational environment in which an agent operates. The platform from which an agent originates is referred to as the home platform, and normally is the most trusted environment for an agent. One or more hosts may comprise an agent platform, and an agent platform may support multiple computational environments, or meeting places, where agents can interact. Since some of these detailed aspects do not affect the discussion of security issues, they are omitted from the agent system model depicted in Figure 1.

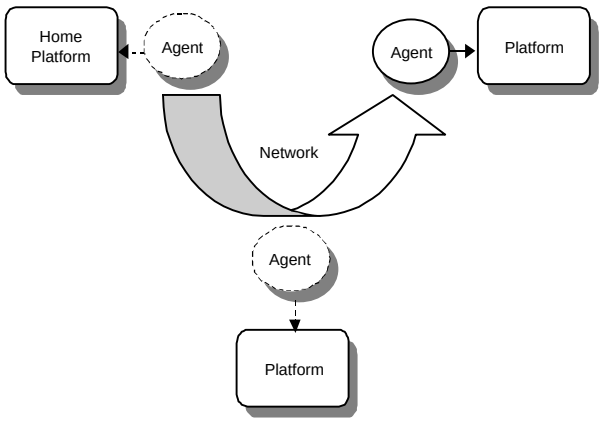

Figure 1 Agent System Model

Four threat categories are identified: threats stemming from an agent attacking an agent platform, an agent platform attacking an agent, an agent attacking another agent on the agent platform, and other entities attacking the agent system. The last category covers the cases of an agent attacking an agent on another agent platform, and of an agent platform attacking another platform, since these attacks are primarily focused on the communications capability of the platform to exploit potential vulnerabilities. The last category also includes more conventional attacks against the underlying operating system of the agent platform.

2.1. Agent-to-Platform

The agent-to-platform category represents the set of threats in which agents exploit security weaknesses of an agent platform or launch attacks against an agent platform. This set of threats includes masquerading, denial of service and unauthorized access.

2.1.1. Masquerading

When an unauthorized agent claims the identity of another agent it is said to be masquerading. The masquerading agent may pose as an authorized agent in an effort to gain access to services and resources to which it is not entitled. The masquerading agent may also pose as another unauthorized agent in an effort to shift the blame for any actions for which it does not want to be held accountable. A masquerading agent may damage the trust the legitimate agent has established in an agent community and its associated reputation.

2.1.2. Denial of Service

Mobile agents can launch denial of service attacks by consuming an excessive amount of the agent platform's computing resources. These denial of service attacks can be launched intentionally by running attack scripts to exploit system vulnerabilities, or unintentionally through programming errors. Security threats resulting from programming errors and intentional flaws have been written about since the early 1970's [35]. Program testing, configuration management, design reviews, independent testing, and other software engineering practices have been proposed to help reduce the risk of programmers intentionally, or unintentionally, introducing malicious code into an organization’s computer systems. The mobile computing paradigm, however, requires an agent platform to accept and execute an agent whose code may have been developed outside its organization and has not been subject to any a priori review. A rogue agent may carry malicious code that is designed to disrupt the services offered by the agent platform, degrade the performance of the platform, or extract information for which it has no authorization to access. Depending on the level of access, the agent may be able to completely shut down or terminate the agent platform.

2.1.3. Unauthorized Access

Access control mechanisms are used to prevent unauthorized users or processes from accessing services and resources for which they have not been granted permission and privileges as specified by a security policy. Each agent visiting a platform must be subject to the platform's security policy. Applying the proper access control mechanisms requires the platform or agent to first authenticate a mobile agent’s identity before it is instantiated on the platform. An agent that has access to a platform and its services without having the proper authorization can harm other agents and the platform itself. A platform that hosts agents representing various users and organizations must ensure that agents do not have

read or write access to data for which they have no authorization, including access to residual data that may be stored in a cache or other temporary storage.

2.2. Agent-to-Agent

The agent-to-agent category represents the set of threats in which agents exploit security weaknesses of other agents or launch attacks against other agents. This set of threats includes masquerading, unauthorized access, denial of service and repudiation. Many agent platform components are also agents themselves. These platform agents provide system-level services such as directory services and inter-platform communication services. Some agent platforms allow direct inter-platform agent-to-agent communication, while others require all incoming and outgoing messages to go through a platform communication agent. These architecture decisions intertwine agent-to-agent and agent-to-platform security. This section addresses agent-to-agent security threats and leaves the discussion of platform related threats to sections 2.1 and 2.3.

2.2.1. Masquerade

Agent-to-agent communication can take place directly between two agents or may require the participation of the underlying platform and the agent services it provides. In either case, an agent may attempt to disguise its identity in an effort to deceive the agent with which it is communicating. An agent may pose as a well-known vendor of goods and services, for example, and try to convince another unsuspecting agent to provide it with credit card numbers, bank account information, some form of digital cash, or other private information. Masquerading as another agent harms both the agent that is being deceived and the agent who's identity has been assumed, especially in agent societies where reputation is valued and used as a means to establish trust.

2.2.2. Denial of Service

In addition to launching denial of service attacks on an agent platform, agents can also launch denial of service attacks against other agents. For example, repeatedly sending messages to another agent, or spamming agents with messages, may place undue burden on the message handling routines of the recipient. Agents that are being spammed may choose to block messages from unauthorized agents, but even this task requires some processing by the agent or its communication proxy. If an agent is charged by the number of CPU cycles it consumes on a platform, spamming an agent may cause the spammed agent to have to pay a monetary cost in addition to a performance cost. Agent communication languages and conversation policies must ensure that a malicious agent doesn't engage another agent in an infinite conversation loop or engage the agent in elaborate conversations with the sole purpose of tying up the agent's resources. Malicious agents can also intentionally distribute false or useless information to prevent other agents from completing their tasks correctly or in a timely manner.

2.2.3. Repudiation

Repudiation occurs when an agent, participating in a transaction or communication, later claims that the transaction or communication never took place. Whether the cause for repudiation is deliberate or accidental, repudiation can lead to serious disputes that may not be easily resolved unless the proper countermeasures are in place. An agent platform cannot prevent an agent from repudiating a transaction, but platforms can ensure the availability of sufficiently strong evidence to support the resolution of disagreements. This evidence may deter an agent that values its reputation and the level of trust others place in it, from falsely repudiating future transactions. Disagreements may arise not only when an agent falsely repudiates a transaction, but also because imperfect business processes may lead to different views of events. Repudiation often occurs within non-agent systems and real-life business transactions within an organization. Documents are occasionally forged, documents are often lost, created by someone without authorization, or modified without being properly reviewed. Since an agent may repudiate a transaction as the result of a misunderstanding, it is important that the agents and agent platforms involved in the transaction maintain records to help resolve any dispute.

2.2.4. Unauthorized Access

If the agent platform has weak or no control mechanisms in place, an agent can directly interfere with another agent by invoking its public methods (e.g., attempt buffer overflow, reset to initial state, etc.), or by accessing and modifying the agent's data or code. Modification of an agent’s code is a particularly insidious form of attack, since it can radically change the agent's behavior (e.g., turning a trusted agent into malicious one). An agent may also gain information about other agents’ activities by using platform services to eavesdrop on their communications.

2.3. Platform-to-Agent

The platform-to-agent category represents the set of threats in which platforms compromise the security of agents. This set of threats includes masquerading, denial of service, eavesdropping, and alteration.

2.3.1. Masquerade

One agent platform can masquerade as another platform in an effort to deceive a mobile agent as to its true destination and corresponding security domain. An agent platform masquerading as a trusted third party may be able to lure unsuspecting agents to the platform and extract sensitive information from these agents. The masquerading platform can harm both the visiting agent and the platform whose identity it has assumed. An agent that masquerades as another agent can harm other agents only through the messages they exchange and the actions they take as a result of these messages, but a malicious platform that masquerades as an authorized platform can do more harm to the duped agent than a

single agent can do on its own. The threat of a malicious platform altering an agent's code, state, or data is discussed in more detail in section 2.3.4.

2.3.2. Denial of Service

When an agent arrives at an agent platform, it expects the platform to execute the agent's requests faithfully, provide fair allocation of resources, and abide by quality of service agreements. A malicious agent platform, however, may ignore agent service requests, introduce unacceptable delays for critical tasks such as placing market orders in a stock market, simply not execute the agent's code, or even terminate the agent without notification. Agents on other platforms waiting for the results of a non-responsive agent on a malicious platform must be careful to avoid becoming deadlocked. An agent can also become livelocked if a malicious platform, or programming error, creates a situation in which some critical stage of the agent's task is unable to finish because more work is continuously created for it to do. Agent livelock differs from agent deadlock in that the livelocked agent is not blocked or waiting for anything, but is continuously given tasks to perform and can never catch up or achieve its goal.

2.3.3. Eavesdropping

The classical eavesdropping threat involves the interception and monitoring of secret communications. The threat of eavesdropping, however, is further exacerbated in mobile agent systems because the agent platform can not only monitor communications, but also can monitor every instruction executed by the agent, all the unencrypted or public data it brings to the platform, and all the subsequent data generated on the platform. Since the platform has access to the agent's code, state, and data, the visiting agent must be wary of the fact that it may be exposing proprietary algorithms, trade secrets, negotiation strategies, or other sensitive information. Even though the agent may not be directly exposing secret information, the platform may be able to infer meaning from the types of services requested and from the identity of the agents with which it communicates. For example, someone's agent may be communicating with a travel agent, although the content of the message may not be exposed, this communication may indicate that the person on whose behalf the agent is acting is planning a trip and will be away from their home in the near future. The platform may share this information it has inferred with a suitcase manufacturer that may begin sending unsolicited advertisements, or even worse, the platform administrators may share this information with thieves who may target the home of the traveler.

2.3.4. Alteration

When an agent arrives at an agent platform it is exposing its code, state, and data to the platform. Since an agent may visit several platforms under various security domains throughout its lifetime, mechanisms must be in place to ensure the integrity of the agent's code, state, and data. A compromised or malicious platform must be prevented from modifying an agent's code, state, or data without being detected. Modification of an

agent's code, and thus the subsequent behavior of the agent on other platforms, can be detected by having the original author digitally sign the agent's code. Detecting malicious changes to an agent's state during its execution or the data an agent has produced while visiting the compromised platform does not yet have a general solution. The agent platform may be running a modified virtual machine, for example, without the agent's knowledge, and the modified virtual machine may produce erroneous results.

A mobile agent that visits several platforms on its itinerary is exposed to a new risk each time it is in transit and each time it is instantiated on a new platform. The party responsible for the malicious alteration of an agent's code, state, or data if not immediately detected may be impossible to track down after the agent has visited other platforms and undergone countless changes of state and data. Although checkpointing and rollback of mathematical computations may be possible in non-agent environments, mobile agent frameworks make this task extremely difficult since an agent's final state and data on a platform may be the result of a series of non-deterministic events that depend on the behavior of autonomous agents whose previous behavior cannot be recreated.

The security risks resulting from an agent moving from its home platform to another is referred to as the "single-hop" problem, while the security risks resulting from an agent visiting several platforms is referred to as the "multi-hop" problem. The risks associated with the single-hop problem are easier to mitigate than the risks associated with a multi-hop scenario, since the protection mechanisms within the trust environment of the home platform are more difficult to use in the latter situation.

Agent platforms can also tamper with agent communications. Tampering with agent communications, for example, could include deliberately changing data fields in financial transactions or even changing a "sell" message to a "buy" message. This type of goal-oriented alteration of the data is more difficult than simply corrupting a message, but the attacker clearly has a greater incentive and reward, if successful, in a goal-oriented alteration attack.

2.4. Other-to-Agent Platform

The other-to-agent platform category represents the set of threats in which external entities, including agents and agent platforms, threaten the security of an agent platform. This set of threats includes masquerading, denial of service, unauthorized access, and copy and replay.

2.4.1. Masquerade

Agents can request platform services both remotely and locally. An agent on a remote platform can masquerade as another agent and request services and resources for which it is not authorized. Agents masquerading as other agents may act in conjunction with a malicious platform to help deceive another remote platform or they may act alone. A

remote platform can also masquerade as another platform and mislead unsuspecting platforms or agents about its true identity.

2.4.2. Unauthorized Access

Remote users, processes, and agents may request resources for which they are not authorized. Remote access to the platform and the host machine itself must be carefully protected, since conventional attack scripts freely available on the Internet can be used to subvert the operating system and directly gain control of all resources. Remote administration of the platform's attributes or security policy may be desirable for an administrator that is responsible for several distributed platforms, but allowing remote administration may make the system administrator’s account or session the target of an attack.

2.4.3. Denial of Service

Agent platform services can be accessed both remotely and locally. The agent services offered by the platform and inter-platform communications can be disrupted by common denial of service attacks. Agent platforms are also susceptible to all the conventional denial of service attacks aimed at the underlying operating system or communication protocols. These attacks are tracked by organizations such as the Computer Emergency Response Team (CERT) at the Carnegie Mellon University and the Federal Computer Incident Response Capability (FedCIRC).

2.4.4. Copy and Replay

Every time a mobile agent moves from one platform to another it increases its exposure to security threats. A party that intercepts an agent, or agent message, in transit can attempt to copy the agent, or agent message, and clone or retransmit it. For example, the interceptor can capture an agent’s "buy order" and replay it several times, having the agent buy more than the original agent had intended. The interceptor may copy and replay an agent message or a complete agent.

3. Security Requirements

The users of networked computer systems have four main security requirements: confidentiality, integrity, availability, and accountability. The users of agent and mobile agent frameworks also have these same security requirements. This section provides a brief overview of these security requirements and how they apply to agent frameworks.

3.1. Confidentiality

Any private data stored on a platform or carried by an agent must remain confidential. Agent frameworks must be able to ensure that their intra- and inter-platform communications remain confidential. Eavesdroppers can gather information about an

agent's activities not only from the content of the messages exchanged, but also from the message flow from one agent to another agent or agents. Monitoring the message flow may allow other agents to infer useful information without having access to the actual message content. A burst of messages from one agent to another, for example, may be an indication that an agent is in the market for a particular set of services offered by the other agent. The eavesdropping agent or external entity may use this information to gain an unfair advantage. Agents may be able to detect an Agent Communication Language (ACL) conversation signature pattern to infer further meaning from an agent conversation. A procurement agent may, for example, send three messages to a vendor agent, followed by one message to its home platform, and conclude the transaction with two messages to the vendor agent. The vendor agent may send two messages to a credit-checking agency and conclude with a message to its banking agent. Although the contents of the messages have never been disclosed, an eavesdropper may be able to infer that the procurement and vendor agents have successfully completed a sale of some commodity or service.

Mobile agents may also want to keep their location confidential. Mobile agents may communicate through a proxy whose location is publicly known if the agents want to conceal their presence on a particular platform. Agents must be allowed to decide if their presence will be publicly available through platform directories, and platforms may enforce different security policies on agents that chose to be anonymous. An agent shopping for goods and services may wish to do so in privacy, but when a financial transaction is to be carried out the platform may require some form of authentication. Agents can assume a pseudonym to conceal their identity from other agents, but this pseudonym is reversible and cannot be used to conceal its identity from the platform itself. In addition, an agent may not want to disclose which platforms it has visited on its itinerary, but the platform security policy of its host may not be willing to accept agents that have been on certain platforms outside the authority of certain approved security domains.

Since audit logs maintain a detailed record of an agent's activities on the platform, the contents of the audit log must also be carefully protected and remain confidential. Access to the audit logs must be restricted to authorized administrators. A mobile agents' audit log may be distributed across several security domains, each having a different audit policy, so some agents may want to carry certain parts of their audit log with them for future reference. In some cases the mobile agent may require the host platform to sign portions of the audit log for which it was responsible.

3.2. Integrity

The agent platform must protect agents from unauthorized modification of their code, state, and data and ensure that only authorized agents or processes carry out any modification of shared data. The agent itself cannot prevent a malicious agent platform from tampering with its code, state, or data, but the agent can take measures to detect this tampering.

The secure operation of mobile agent systems also depends on the integrity of the local and remote agent platforms themselves. A malicious host can easily compromise the integrity of a mobile agent that is visiting a remote agent platform. The trend towards releasing open source platforms and operating systems makes it easier for unscrupulous administrators or organizations to make unauthorized modifications to the agent platform and the underlying framework. The agent's user and developer may have no knowledge of the behavior of this modified framework and the effects it will have on the agent's code, state and data that are under the platform's complete control. A malicious platform may make subtle changes in the execution flow of the agent’s code and make changes to computed results that are difficult to detect. A malicious platform could interfere in transactions between agents representing different organizations and tamper with the audit trails. This could result in each organization involved in the transaction blaming the other for not delivering promised goods or services.

System access controls must be in place to protect the integrity of the agent platform from unauthorized users. Changes to the platform by unauthorized users are more likely to be detected over time than changes to the platform by authorized users. Intentional modification or tampering with the agent platform by authorized or unauthorized users places additional burden on mobile agents to be able to detect and recover from intentional corruption or alteration. Many security mechanisms available to the agent to protect itself may come at a serious performance and development cost and must be carefully weighed against the advantages offered by code mobility. If the agent is carrying out a critical or sensitive task for which it must assume an unreasonable risk or for which it cannot provide suitable protection, the agent developer or user may choose to restrict the agent's mobility. Thus, given the current security mechanisms available to mobile agents, agents may choose to forego mobility during certain security-sensitive transactions, and limit the types of security-sensitive transactions a mobile agent is authorized to conduct.

Goal-oriented attacks against agent communications aim to compromise the integrity of an intra- or inter-platform message by changing the content of the message, replacing the entire message, reusing an old message, deleting the message, or changing the source or destination of the message. Goal-oriented malicious attacks on the integrity of agent communications can do far greater harm to the agent than transmission errors resulting from poor or intermittent communication channels. Agent platforms rely on lower-level protocols to ensure the integrity of the agent communications. For example, TCP/IP ensures the detection of most transmission errors, and processes in the communications stack detect errors and arrange for retransmission. This error detection and correction is transparent to higher-level applications that rely on these protocols to ensure the integrity of their communications.

3.3. Accountability

Each process, human user, or agent on a given platform must be held accountable for their actions. In order to be held accountable each process, human user, or agent must be uniquely identified, authenticated, and audited. Example actions for which they must be

held accountable include: access to an object, such as a file, or making administrative changes to a platform security mechanism. Accountability requires maintaining an audit log of security-relevant events that have occurred, and listing each event and the agent or process responsible for that event. Security-relevant events are defined in the platform security policy and should include, at a minimum, the following: user/agent name, time of event, type of event, and success or failure of the event. Audit logs must be protected from unauthorized access and modification. Measures need to be in place to prevent auditable events from being lost when the storage media reaches its capacity. These measures can include alerting the system administrator when the audit logs reach a certain capacity and depend on the system administrator to perform routine maintenance as part of their normal administrator duties.

Audit logs keep users and processes accountable for their actions. Mobile agents create audit trails across several platforms, with each platform logging separate events and possibly auditing different views of the same event (e.g., registration within a remote directory). In the case where an agent may require access to information distributed across different platforms it may be difficult to reconstruct a sequence of events. Platforms that keep distributed audit logs must be able to maintain a concept of global time or ordering of events. Auditing agent actions is very important considering the liability issues that may arise from the failure of a platform administrator to exercise due diligence in preventing known denial of service attacks. Legal issues may also come into play as rules governing the actions of agents may differ in states, countries, and regional trading blocks.

Audit logs are also necessary and valuable when the platform must recover from a security breach, or a software or hardware failure. Full recovery from a faulty or compromised system requires not only restoring the system to a safe state and performing some sort of fault diagnosis, but also sorting out which agents belonging to which organizations were affected. Audit logs are also necessary in cases of agents falsely repudiating their actions and for potential liability issues that are unique to agent societies.

Authentication mechanisms provide accountability for user actions. Agents must be able to authenticate their identity to platforms and other agents, while platforms must be able to authenticate their identity to agents and other platforms. The level of sensitivity of the transaction or data for which agents request access may require the agent to offer different degrees of authentication. In agent societies where reputation is valued and used as a means to establish trust, an agent platform must exercise due diligence to protect an agent's reputation from being harmed by other agents through masquerade.

An agent accessing publicly advertised product prices may not be required to authenticate itself at all and may only have "read access" to certain public data, but if this agent decides to purchase any of these products the agent must be able to authenticate itself before the transaction is completed. Similarly, the purchasing agent will require some proof of the selling agent's claimed identity. Both the purchasing and selling agent may require the agent platform to authenticate itself before they visit this platform. Humans must be able to authenticate themselves to agents through whatever means the agent platform makes

available to the human user, for example, voice recognition, biometrics, passwords, or smart cards.

Accountability is also essential for building trust among agents and agent platforms. An authenticated agent may comply with the security policies of the agent platform, but still exhibit malicious behavior by lying or deliberately spreading false information. Additional auditing may be helpful in proving the malicious agent's attempts at deception. For example, the communication acts of an ACL conversation could be logged, but this results in costly overhead. If it is assumed that the malicious agent does not value its reputation within an agent community, then it would be difficult to prevent malicious agents from lying.

3.4. Availability

The agent platform must be able to ensure the availability of both data and services to local and remote agents. The agent platform must be able to provide controlled concurrency, support for simultaneous access, deadlock management, and exclusive access as required. Shared data must be available in a usable form, capacity must be available to meet service needs, and provisions for the fair allocation of resources and timeliness of service must be made. The agent platform must be able to detect and recover from system software and hardware failures. While the platform can provide some level of fault-tolerance and fault-recovery, agents may be required to assume responsibility for their own fault-recovery.

The agent platform must be able to handle the requests of hundreds or thousands of visiting and remote agents or risk creating an unintentional denial of service. In the event that a platform cannot handle its computational and communication load, the platform must provide graceful degradation of services and notify agents that it can no longer provide the level and quality of service expected by the agents requesting its services.

The threat of a denial of service attack being launched by rogue or flawed agents underscores the need for the control and monitoring of platform resources. A denial of service attack against the platform is an indirect attack on all the agents that rely on this platform for agent services and platform resources. Since the platform may be hosting agents from several other organizations, a denial of service attack on a single platform may adversely impact more than one agent, platform, or organization.

Ensuring confidentiality and integrity places restraints on availability and also has performance costs. Encrypting agents and messages in transit may impose unacceptable delays in environments where near real-time response is required. Ensuring accountability may also place restraints on the availability of platform services. For example, when the audit log storage media exhausts its capacity, many security policies disable all services that create records in the audit log. In an effort to maintain accountability, this policy may inadvertently create an opportunity for a denial of service attack by simply producing an

inordinate amount of auditable events, such as repeated failed login attempts, and consume the remaining capacity of the audit log storage media.

3.5. Anonymity

The agent platform may need to balance an agent's need for privacy with the platform's need to hold the agent accountable for its actions. The platform may be able to keep the agent's identity secret from other agents and still maintain a form of reversible anonymity where it can determine the agent's identity if necessary and legal. Anonymity in human communities may help foster certain types of commerce or promote the freedom of expression by eliminating the fear of retribution for voicing unpopular viewpoints. There are many situations, however, in which the participants are reluctant or unwilling to engage in a transaction with an anonymous counterpart. Purchasers of goods and services may want to protect their privacy by remaining anonymous, but credit agencies would not extend credit to anonymous consumers without being able to verify their credit history and credit worthiness. The anonymity issues affecting human communities also apply to agent communities. The security policies of agent platforms and their auditing requirements must be carefully balanced with agent privacy expectations. The collection and use of audit information for uses other than security must also be understood by the agents visiting the agent platform. Moreover, what information belongs in public agent directories and under what conditions can the information be accessed from these directories must also be carefully controlled.

4. Countermeasures

Many conventional security techniques used in contemporary distributed applications (e.g., client-server) also have utility as countermeasures within the mobile agent paradigm. Moreover, there are a number of extensions to conventional techniques and techniques devised specifically for controlling mobile code and executable content (e.g., Java applets) that are applicable to mobile agent security. We review these countermeasures by considering those techniques that can be used to protect agent platforms, separately from those used to protect the agents that run on them.

Most agent systems rely on a common set of baseline assumptions regarding security. The first is that an agent trusts the home platform where it is instantiated and begins execution. The second is that the home platform and other equally trusted platforms are implemented securely, with no flaws or trapdoors that can be exploited, and behave non-maliciously. The third is that public key cryptography, primarily in the form of digital signature, is utilized through certificates and revocation lists managed through a public key infrastructure.

4.1. Protecting the Agent Platform

One of the main concerns with an agent system implementation is ensuring that agents are not able to interfere with one another or with the underlying agent platform. One common

approach for accomplishing this is to establish separate isolated domains for each agent and the platform, and control all inter-domain access. In traditional terms this concept is referred to as a reference monitor [22]. An implementation of a reference monitor has the following characteristics:

- It is always invoked and non-bypassable, mediating all accesses;
- It is tamper-proof; and
- It is small enough to be analyzed and tested.

Implementations of the reference monitor concept have been around since the early 1980's and employ a number of conventional security techniques, which are applicable to the agent environment. Such conventional techniques include the following:

- Mechanisms to isolate processes from one another and from the control process,
- Mechanisms to control access to computational resources,
- Cryptographic methods to encipher information exchanges,
- Cryptographic methods to identify and authenticate users, agents, and platforms, and
- Mechanisms to audit security-relevant events occurring at the agent platform.

More recently developed techniques aimed at mobile code and mobile agent security have for the most part evolved along these traditional lines. Techniques devised for protecting the agent platform include the following:

- Software-Based Fault Isolation,
- Safe Code Interpretation,
- Signed Code,
- Authorization and Attribute Certificates,
- State Appraisal,
- Path Histories, and
- Proof Carrying Code.

4.1.1. Software-Based Fault Isolation

Software-Based Fault Isolation [23], as its name implies, is a method of isolating application modules into distinct fault domains enforced by software. The technique allows untrusted programs written in an unsafe language, such as C, to be executed safely within the single virtual address space of an application. Untrusted machine interpretable code modules are transformed so that all memory accesses are confined to code and data segments within their fault domain. Access to system resources can also be controlled through a unique identifier associated with each domain. The technique, commonly referred to as *sandboxing*, is highly efficient compared with using hardware page tables to maintain separate address spaces for modules, when the modules communicate frequently among fault domains. It is also ideally suited for situations where most of the code falls into one domain that is trusted, since modules in trusted domains incur no execution overhead.

4.1.2. Safe Code Interpretation

Agent systems are often developed using an interpreted script or programming language. The main motivation for doing this is to support agent platforms on heterogeneous computer systems. Moreover, the higher conceptual level of abstraction provided by an interpretative environment can facilitate the development of the agent's code [24]. The idea behind Safe Code Interpretation is that commands considered harmful can be either made safe for or denied to an agent. For example, a good candidate for denial would be the command to execute an arbitrary string of data as a program segment.

One of the most widely used interpretative languages today is Java. The Java programming language and runtime environment [2] enforces security primarily through strong type safety. Java follows a so-called sandbox security model, used to isolate memory and method access, and maintain mutually exclusive execution domains. Security is enforced through a variety of mechanisms. Static type checking in the form of byte code verification is used to check the safety of downloaded code. Some dynamic checking is also performed during runtime. A distinct name space is maintained for untrusted downloaded code, and linking of references between modules in different name spaces is restricted to public methods. A security manager mediates all accesses to system resources, serving in effect as a reference monitor. In addition, Java inherently supports code mobility, dynamic code downloading, digitally signed code, remote method invocation, object serialization, platform heterogeneity, and other features that make it an ideal foundation for agent development. There are many agent systems based on Java, including Aglets [3, 14], Mole [8], Ajanta [25], and Voyager [9]. However, limitations of Java to account for memory, CPU, and network resources consumed by individual threads [40] and to support thread mobility [2] have been noted.

Probably the best known of the safe interpreters for script-based languages is Safe Tcl [12], which was used in the early development of the Agent Tcl [11] system. Safe Tcl employs a *padded cell* concept, whereby a second "safe" interpreter pre-screens any harmful commands from being executed by the main Tcl interpreter. The term padded cell refers to this isolation and access control technique, which provides the foundation for implementing the reference monitor concept. More than one safe interpreter can be used to implement different security policies, if needed. Constructing policy-based interpreters requires skill to avoid overly restrictive or unduly protected computing environments.

In general, current script-based languages do not incorporate an explicit security model in their design, and rely mainly on decisions taken during implementation. One recent exception is the implementation of secure JavaScript in Mozilla [39]. The implementation uses a padded cell to control access to resources and external interfaces, prevents residual information from being retained and accessible among different contexts operating simultaneously or sequentially, and allows policy, which partitions the name space for access control purposes, to be specified independently of mechanism.

4.1.3. Signed Code

A fundamental technique for protecting an agent system is signing code or other objects with a digital signature. A digital signature serves as a means of confirming the authenticity of an object, its origin, and its integrity. Typically the code signer is either the creator of the agent, the user of the agent, or some entity that has reviewed the agent. Because an agent operates on behalf of an end-user or organization, mobile agent systems [10, 11, 14, 25] commonly use the signature of the user as an indication of the authority under which the agent operates.

Code signing involves public key cryptography, which relies on a pair of keys associated with an entity. One key is kept private by the entity and the other is made publicly available. Digital signatures benefit greatly from the availability of a public key infrastructure, since certificates containing the identity of an entity and its public key (i.e., a public key certificate) can be readily located and verified. Passing the agent's code through a non-reversible hash function, which provides a fingerprint or unique message digest of the code, and then encrypting the result with the private key of the signer forms a digital signature. Because the message digest is unique, and thus bound to the code, the resulting signature also serves as an integrity mechanism. The agent code, signature, and public key certificate can then be forwarded to a recipient, who can easily verify the source and authenticity of the code.

Note that the meaning of a signature may be different depending on the policy associated with the signature scheme and the party who signs. For example, the author of the agent, either an individual or organization, may use a digital signature to indicate who produced the code, but not to guarantee that the agent performs without fault or error. In fact, author-oriented signature schemes were originally intended to serve as digital shrink wrap, whereby the original product warranty limitations stated in the license remain in effect (e.g., manufacturer makes no warranties as to the fitness of the product for any particular purpose).

Microsoft's Authenticode, a common form of code signing, enables Java applets or Active X controls to be signed, ensuring users that the software has not been tampered with or modified and that the identity of the author is verified. For many users, however, the signature has gone beyond establishing authenticity and become a form of trust in the software, which could ultimately have disastrous consequences.

Rather than relying solely on the reputation of a code producer, it would be prudent to have an independent review and verification of code performed by a trusted party or rating service [14, 26]. For example, the decision to execute an agent could be taken only with the endorsement of a site security administrator, given in the form of a digital signature applied to the code. While such an approach may be desirable, experience with trusted computer evaluations indicates that obtaining quality security reviews in a timely fashion is difficult to achieve. Nevertheless, some checking is better that none, provided that the limitations are known by everyone.

4.1.4. State Appraisal

The goal of State Appraisal [16] is to ensure that an agent has not been somehow subverted due to alterations of its state information. The success of the technique relies on the extent to which harmful alterations to an agent's state can be predicted, and countermeasures, in the form of appraisal functions, can be prepared before using the agent. Appraisal functions are used to determine what privileges to grant an agent, based both on conditional factors and whether identified state invariants hold. An agent whose state violates an invariant can be granted no privileges, while an agent whose state fails to meet some conditional factors may be granted a restricted set of privileges.

Both the author and owner of an agent produce appraisal functions that become part of an agent's code. An owner typically applies state constraints to reduce liability and/or control costs. When the author and owner each digitally sign the agent, their respective appraisal functions are protected from undetectable modification. An agent platform uses the functions to verify the correct state of an incoming agent and to determine what privileges the agent can possess during execution. Privileges are issued by a platform based on the results of the appraisal function and the platform's security policy. It is not clear how well the theory will hold up in practice, since the state space for an agent could be quite large, and while appraisal functions for obvious attacks may be easily formulated, more subtle attacks may be significantly harder to foresee and detect. The developers of the technique, in fact, indicate it may not always be possible to distinguish normal results from deceptive alternatives.

4.1.5. Path Histories

The basic idea behind Path Histories [20, 27] is to maintain an authenticatable record of the prior platforms visited by an agent, so that a newly visited platform can determine whether to process the agent and what resource constraints to apply. Computing a path history requires each agent platform to add a signed entry to the path, indicating its identity and the identity of the next platform to be visited, and to supply the complete path history to the next platform. To prevent tampering, the signature of the new path entry must include the previous entry in the computation of the message digest. Upon receipt, the next platform can then determine whether it trusts the previous agent platforms that the agent visited, either by simply reviewing the list of identities provided or by individually authenticating the signatures of each entry in the path history. While the technique does not prevent a platform from behaving maliciously, it serves as strong deterrent, since the platform's signed path entry is non-repudiatable. One obvious drawback is that path verification becomes more costly as the path history increases.

4.1.6. Proof Carrying Code

The approach taken by Proof Carrying Code [17] obligates the code producer (e.g., the author of an agent) to formally prove that the program possesses safety properties

previously stipulated by the code consumer (e.g., security policy of the agent platform). Proof Carrying Code is a prevention technique, while code signing is an authenticity and identification technique used to deter, but not prevent the execution of unsafe code. The code and proof are sent together to the code consumer where the safety properties can be verified. A safety predicate, representing the semantics of the program, is generated directly from the native code to ensure that the companion proof does in fact correspond to the code. The proof is structured in a way that makes it straightforward to verify without using cryptographic techniques or external assistance. Once verified, the code can run without further checking. Any attempts to tamper with either the code or the safety proof result in either a verification error or, if the verification succeeds, a safe code transformation.

Initial research has demonstrated the applicability of Proof Carrying Code for fine-grained memory safety, and shown the potential for other types of safety policies, such as controlling resource use. Thus, the technique could serve as a reasonable alternative to Software-Based Fault Isolation in some applications. The theoretical underpinnings of Proof Carrying Code are based on well-established principles from logic, type theory, and formal verification. There are, however, some potentially difficult problems to solve before the approach is considered practical. They include a standard formalism for establishing security policy, automated assistance for the generation of proofs, and techniques for limiting the potentially large size of proofs that in theory can arise. In addition, the technique is tied to the hardware and operating environment of the code consumer, which may limit its applicability.

4.2. Protecting Agents

While countermeasures directed towards platform protection are a direct evolution of traditional mechanisms employed by trusted hosts, and emphasize active prevention measures, countermeasures directed toward agent protection tend more toward detection measures as a deterrent. This is due to the fact that an agent is completely susceptible to an agent platform and cannot prevent malicious behavior from occurring, but may be able to detect it.

The problem stems from the inability to effectively extend the trusted environment of an agent's home platform to other agent platforms. While a user may digitally sign an agent on its home platform before it moves onto a second platform, that protection is limited. The second platform receiving the agent can rely on the signature to verify the source and integrity of the agent's code, data, and state information provided that the private key of the user has not been compromised. On the agent's subsequent hop to a third platform, the initial signature from the first platform remains valid for the original code, data, and state information, but not for any state or data generated on the second platform.

For some applications, such minimal protection may be adequate. For example, agents that do not accumulate state or convey their results back to the home platform after each hop have less exposure to certain attacks. For other applications, simple schemes may prove

adequate. For example, the Jumping Beans [43] agent system addresses some security issues by implementing a client-server architecture, whereby an agent always returns to a secure central host first before moving onto any other platform. Agent systems that allow for more decentralized mobility, such as IBM Aglets, prevent the receiving platform from accepting agents from any agent platform that is not defined as a trusted peer within the receiving platform's security policy. Alternatively, the originator can restrict an agent's itinerary to only a trusted set of platforms known in advance.

While these simple schemes have value, they do not support the loose roaming itineraries envisioned in many agent applications. Some more general-purpose techniques for protecting an agent include the following:

- Partial Result Encapsulation,
- Mutual Itinerary Recording,
- Itinerary Recording with Replication and Voting,
- Execution Tracing,
- Environmental Key Generation,
- Computing with Encrypted Functions, and
- Obfuscated Code (Time Limited Blackbox).

4.2.1. Partial Result Encapsulation

One approach used to detect tampering by malicious hosts is to encapsulate the results of an agent's actions, at each platform visited, for subsequent verification, either when the agent returns to the point of origin or possibly at intermediate points as well. Encapsulation may be done for different purposes with different mechanisms, such as providing confidentiality using encryption, or for integrity and accountability using digital signature. The information encapsulated depends somewhat on the goals of the agent, but typically include responses to inquiries made or transactions issued at the platform. In general, there are three alternative ways to encapsulate partial results:

- Provide the agent with a means for encapsulating the information,
- Rely on the encapsulation capabilities of the agent platform, or
- Rely on a trusted third party to timestamp a digital fingerprint of the results.

While none of the alternatives prevents malicious behavior by the platform, they do allow certain types of tampering to be detected. One difference with agent controlled encapsulation is that it can be applied independently in the design of an appropriate agent application, regardless of the capabilities of the agent platform or supporting infrastructure.

Often the amount of information gathered by an agent is rather small in comparison to the size of the encryption keys involved and the resulting ciphertext. A solution called sliding encryption [29] allows small amounts of data to be encrypted and yield efficient sized results. The scenario envisioned for use of sliding encryption is one in which the agent,

using a public key it carries, encrypts the information as it is accumulated at each platform visited. Later, when the agent returns to the point of origination, the information is decrypted using the private key maintained there. While the purpose of sliding encryption is confidentiality, an additional integrity measure could be applied as well, before encryption occurs.

Another method for an agent to encapsulate result information is to use Partial Result Authentication Codes (PRAC) [30], which are cryptographic checksums formed using secret key cryptography (i.e., message authentication codes). This technique requires the agent and its originator to maintain or incrementally generate a list of secret keys used in the PRAC computation. Once a key is applied to encapsulate the information collected, the agent destroys it before moving onto the next platform, guaranteeing forward integrity. The forward integrity property ensures that if one of the servers visited is malicious, the previous set of partial results remains valid. However, only the originator can verify the results, since no other copies of the secret key remain. As an alternative, public key cryptography and digital signatures can be used in lieu of secret key techniques. One benefit is that authentication of the results can be made a publicly verifiable process at any platform along the way, while maintaining forward integrity.

The PRAC technique has a number of limitations. The most serious occurs when a malicious platform retains copies of the original keys or key generating functions of an agent. If the agent revisits the platform or visits another platform conspiring with it, a previous partial result entry or series of entries could be modified without the possibility of detection. Since the PRAC is oriented towards integrity and not confidentiality, the accumulated set of partial results can also be viewed by any platform visited, although this is easily resolved by applying sliding key or other forms of encryption.

Rather than relying on the agent to encapsulate the information, each platform can be required to encapsulate partial results along the way [27, 32]. The distinction is not only one of where the encapsulation mechanisms are retained, either with the agent or at a platform, but also one of responsibility and associated liabilities. While it appears rather straightforward to change from one orientation to the other, the nuances can be significant. For example, an agent, which used to sign partial results with the public of its originator, is revised to have all platforms visited sign the results with their private key. If the agent triggers a digital signature encapsulation via a platform programming interface, the implementation must ensure that enough context is included (e.g., the time and query issued) so that agent cannot induce the platform to sign arbitrary information.

Karjoth and his associates [38] devised a platform oriented technique for encapsulating partial results, which reformulated and improved on the PRAC technique. The approach is to construct a chain of encapsulated results that binds each result entry to all previous entries and to the identity of the subsequent platform to be visited. Each platform digitally signs its entry using its private key, and uses a secure hash function to link results and identities within an entry. Besides forward integrity, the encapsulation technique also provides confidentiality by encrypting each piece of accumulated information with the

public key of the originator of the agent. Moreover, the forward integrity is stronger than that achieved with PRAC, since a platform is unable to modify its entry in the chain, should it be revisited by the agent, or to collude with another platform to modify entries, without invalidating the chain. A variant of this technique, which uses message authentication codes in lieu of digital signatures, is also described.

Yee, who proposed the PRAC technique, noted that forward integrity could also be achieved using a trusted third party that performs digital time-stamping. A digital timestamp [31] allows one to verify that the contents of a file or document existed, as such, at a particular point in time. Yee raises the concern that the granularity of the timestamps may limit an agent's maximum rate of travel, since it must reside at one platform until the next time period. Another possible concern is the general availability of a trusted time-stamping infrastructure.

4.2.2. Mutual Itinerary Recording

One interesting variation of Path Histories is a general scheme for allowing an agent's itinerary to be recorded and tracked by another cooperating agent and vice versa [19], in a mutually supportive arrangement. When moving between agent platforms, an agent conveys the last platform, current platform, and next platform information to the cooperating peer through an authenticated channel. The peer maintains a record of the itinerary and takes appropriate action when inconsistencies are noted. Attention is paid so that an agent avoids platforms already visited by its peer. The rationale behind this scheme is founded on the assumption that only a few agent platforms are malicious, and even if an agent encounters one, the platform is not likely to collaborate with another malicious platform being visited by the peer. Therefore, by dividing up the operations of the application between two agents, certain malicious behavior of an agent platform can be detected.

The scheme can be generalized to more than two cooperating agents. For some applications it is also possible for one of the agents to remain static at the home platform. Because the path records are maintained at the agent level, this technique can be incorporated into any appropriate application. Some drawbacks mentioned include the cost of setting up the authenticated channel and the inability of the peer to determine which of the two platforms is responsible if the agent is killed.

4.2.3. Itinerary Recording with Replication and Voting

A faulty agent platform can behave similar to a malicious one. Therefore, applying fault tolerant capabilities to this environment should help counter the effects of malicious platforms. One such technique for ensuring that a mobile agent arrives safely at its destination is through the use of replication and voting [32]. The idea is that rather than a single copy of an agent performing a computation, multiple copies of the agent are used. Although a malicious platform may corrupt a few copies of the agent, enough replicates avoid the encounter to successfully complete the computation.

For each stage of the computation, the platform ensures that arriving agents are intact, carrying valid credentials. Platforms involved in a particular stage of a computation are expected to know the set of acceptable platforms for the previous stage. The platform propagates onto the next stage only a subset of the replica agents it considers valid, based on the inputs it receives. Underlying assumptions are that the majority of the platforms are well behaved, and most agents arrived unscathed with equivalent results at each stage. One of the protocols used in this technique requires forwarded agents to convey the accumulated signature entries of all forwarding nodes onto each stage. The approach taken is similar to Path Histories, but extended with fault tolerant capabilities. The technique seems appropriate for specialized applications where agents can be duplicated without problems, the task can be formulated as a multi-staged computation, and survivability is a major concern. One obvious drawback is the additional resources consumed by replicate agents.

4.2.4. Execution Tracing

Execution tracing [18] is a technique for detecting unauthorized modifications of an agent through the faithful recording of the agent's behavior during its execution on each agent platform. The technique requires each platform involved to create and retain a non-repudiatable log or trace of the operations performed by the agent while resident there, and to submit a cryptographic hash of the trace upon conclusion as a trace summary or fingerprint. A trace is composed of a sequence of statement identifiers and platform signature information. The signature of the platform is needed only for those instructions that depend on interactions with the computational environment maintained by the platform. For instructions that rely only on the values of internal variables, a signature is not required and, therefore, is omitted. The technique also defines a secure protocol to convey agents and associated security related information among the various parties involved, which may include a trusted third party to retain the sequence of trace summaries for the agent's entire itinerary. If any suspicious results occur, the appropriate traces and trace summaries can be obtained and verified, and a malicious host identified.

The approach has a number of drawbacks, the most obvious being the size and number of logs to be retained, and the fact that the detection process is triggered occasionally, based on suspicious results or other factors. Other more subtle problems identified include the lack of accommodating multi-threaded agents and dynamic optimization techniques. While

the goal of the technique is to protect an agent, the technique does afford some protection for the agent platform, providing that the platform can also obtain the relevant trace summaries and traces from the various parties involved.

4.2.5. Environmental Key Generation

Environmental Key Generation [33] describes a scheme for allowing an agent to take predefined action when some environmental condition is true. The approach centers on constructing agents in such a way that upon encountering an environmental condition (e.g., string match in search), a key is generated, which is used to unlock some executable code cryptographically. The environmental condition is hidden through either a one-way hash or public key encryption of the environmental trigger. The technique ensures that a platform or an observer of the agent cannot uncover the triggering message or response action by directly reading the agent's code.

The procedure is somewhat akin to the way in which passwords are maintained in modern operating systems (e.g., the hash of a password is stored) and used to determine whether login attempts are valid. One weakness of this approach is that a platform that completely controls the agent could simply modify the agent to print out the executable code upon receipt of the trigger, instead of executing it. Another drawback is that an agent platform typically limits the capability of an agent to execute code created dynamically, since it is considered an unsafe operation. An author of an agent can apply the technique, however, in conjunction with other protection mechanisms for specific applications on appropriate platforms.

4.2.6. Computing with Encrypted Functions

The goal of Computing with Encrypting Functions [21] is to determine a method whereby mobile code can safely compute cryptographic primitives, such as a digital signature, even though the code is executed in untrusted computing environments and operates autonomously without interactions with the home platform. The approach is to have the agent platform execute a program embodying an enciphered function without being able to discern the original function; the approach requires differentiation between a function and a program that implements the function.

For example, Alex has an algorithm to compute a function *f*. Barb has input x and wants to compute *f*(x) for Alex, but he doesn't want Barb to learn anything about *f*. If *f* can be encrypted in a way that results in another function E(*f*), then Alex can create a program P(E(*f*)), which implements E(*f*), and send it to Barb, embedded within his agent. Barb then runs the agent, which executes P(E(*f*)) on x, and returns the result to Alex who decrypts it to obtain *f*(x). If *f* is a signature algorithm with an embedded key, the agent has an effective means to sign information without the platform discovering the key. Similarly, if it is an encryption algorithm containing an embedded key, the agent has an effective means to encrypt information at the platform.

Although the idea is straightforward, the trick is to find appropriate encryption schemes that can transform arbitrary functions as intended. Sander and Tschudin propose investigating algebraic homomorphic encryption schemes as one possible candidate. Their initial results look promising, and hopefully will form the basis for discovering other classes of functions. The technique, while very powerful, does not prevent denial of service, replay, experimental extraction, and other forms of attack against the agent.

4.2.7. Obfuscated Code

Hohl [34] gives a detailed overview of the threats stemming from an agent encountering a malicious host as motivation for Blackbox Security. The strategy behind this technique is simple -- scramble the code in such a way that no one is able to gain a complete understanding of its function (i.e., specification and data), or to modify the resulting code without detection. A serious problem with the general technique is that there is no known algorithm or approach for providing Blackbox protection. However, the paper cites Computing with Encrypted Functions as an example of an approach that falls into the Blackbox category, with certain reservations concerning the limited range of input specifications that apply.

A time limited variation of Blackbox protection is introduced as a reasonable alternative, whereby the strength of the scrambling does not necessarily imply encryption as with the unqualified one, but relies mainly on obfuscation algorithms. Since an agent can become invalid before completing its computation, obfuscated code is suitable for applications that do not convey information intended for long-lived concealment. The examples given for pure obfuscation algorithms seem rather trivial and potentially ineffective. One promising method relies on a trusted host to trigger the execution of an agent's code segment. It is not strictly speaking a pure obfuscation algorithm, however, since code is redistributed to a trusted host, which is not part of the originally proposed scheme. The method does, however, indicate a possible relationship between Environmentally Generated Keys and Obfuscated Code techniques. One serious drawback to this technique is the lack of an approach for quantifying the protection interval provided by the obfuscation algorithm, thus making it difficult to apply in practice. Furthermore, no techniques are currently known for establishing the lower bounds on the complexity for an attacker to reverse engineer an agent's code.

5. Mobile Agent Applications and Security Scenarios

Mobile agent technology is beginning to make its way out of research labs and is finding its way into many commercial applications areas. The following section takes a look at these application areas and discusses relevant security issues for typical scenarios.

5.1. Electronic Commerce

Mobile agent-based electronic commerce applications have been proposed and are being developed for a number of diverse business areas, including contract negotiations, service

brokering, auctions, and stock trading [1, 4, 27, 41]. For example, manufacturers can negotiate the delivery of goods and services with suppliers through their respective agents. The agents may need to access the supplier's database, transfer money, and negotiate terms of delivery, warranties, and service contracts. Mobile agents representing bidders may meet on an auction house's platform to engage in blind, straight, or Dutch auctions, each employing different strategies and having different financial constraints.

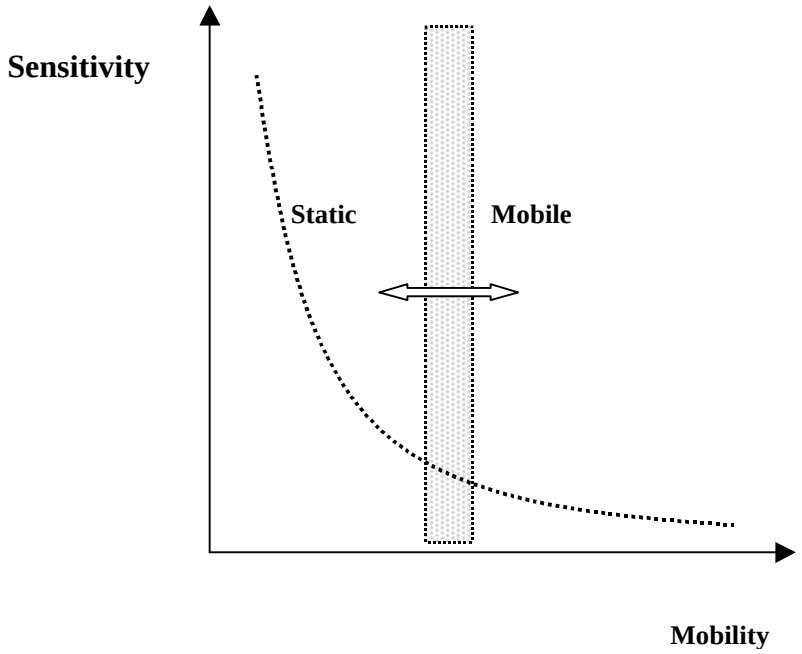

Figure 2 Degree of Mobility vs. Sensitivity of Agent

The level of security required for the application and the sensitivity of the mobile agent's code and data directly influences the degree of mobility of a mobile agent. As shown in Figure 2, as the sensitivity of the agent's task increases, the designer of the agent may decrease the degree of mobility of the agent. The shaded vertical bar represents a cut-off point for the designer to decide which agents will be static and which agents will be mobile. This decision will be made based on the available security mechanisms, performance requirements, sensitivity of the agent's code and data, maximum acceptable risk, and the level of functionality required.

A mobile agent’s tasks could be divided between static and mobile agents so that the more security the transaction required, the less mobile the agent would be. For example, a "window shopping" mobile agent could visit vendor sites searching for the price and availability of goods and services. When the mobile agent finds the goods and services that meet its criteria, a static agent at the home platform or at a trusted platform could complete the sale and sign the receipt with its private key. The loss of the window shopping agent which has no authority to transfer money or sign receipts could be

tolerated, but the loss or compromise of an agent that is authorized to sign documents or transfer money could not. The vendor would be more likely to allow an agent to visit its platform that would only need to read public information, and the mobile agent would want any evidence used for non-repudiation to be stored and guarded on its home platform or a trusted third party platform and not generated or stored on the vendor's platform.

5.2. Network Management

Mobile agents are also well suited for network management applications such as remote network management, software distribution, and adaptive response to network events. Most of the current network management software is based on the Simple Network Management Protocol (SNMP). The developers of SNMP designed it in the mid-1980s as a temporary measure with known problems and inherent limitations until industry could provide a better solution. Mobile agents allow for greater control of network devices and aren't limited by whatever parameters the manufacturer of the device decides to make available via SNMP. Mobile agents can also provide adaptive responses to network events. Without mobile agents, all of the software required to support and respond to all possible scenarios must be kept loaded and available on a device at all times. The network manager must take the time to respond to each event, and interact with each device individually. In contrast, mobile agents can be dispatched to respond to network events, and only the software required for normal operation needs to be loaded onto the device. Upgrading network software to add new functionality involves a significant roll-out effort on behalf of network administrators. Installing new software, upgrading existing software, or reconfiguring network hardware can be accomplished by simply dispatching mobile agents to the appropriate machines [28].

Network management security policies are unlikely to allow code from outside the organization onto the network. These policies are more likely to allow only in-house code or code signed by a trusted vendor onto the network. Only authorized network administrators would be allowed to introduce agents onto the network, strict access control to mobile agent platforms would be in place, mobile agent design and development would conform to mature software engineering principles and quality control methods, and audit logs of security-related events would be maintained and analyzed.

Insider crime, where a disgruntled employee tries to launch malicious agents, seems to be one of the greatest threats in an environment in which few or no outside agents are allowed onto the network. Accountability mechanisms can be put in place to deter insiders from launching malicious agents. These mechanisms, however, can only act as a deterrent and cannot prevent this from actually occurring. This risk would seem to be equivalent to an insider in a non-mobile agent computing environment introducing Trojan horses, backdoors, or other malicious code, but in the case of mobile agents the damage could be far greater. Mobile agents that are developed in-house or purchased from trusted vendors are likely to undergo the same software engineering methods as their non-mobile counterparts in order to assure the quality of code. Given the complexity of developing

secure and reliable code in non-mobile computing paradigms, it stands to reason that design and programming errors will continue to create headaches for system administrators and software developers of mobile agent systems. Mobile agents' inherent autonomy, ability to clone themselves, and mobility adds more complexity to the design and development process. Clearly, mobile agent design, development, and management tools are still in their infancy and will be needed before any large-scale deployment of agents will become feasible in an operational environment. Agent developers and administrators could also benefit from better resource control mechanisms in mobile agent platforms.

In contrast to the network management domain, active networking and the use of intelligent packets represents the other end of the spectrum where packets from almost anywhere can request computing and communication services from routers or other networking elements on the Internet. Since the intelligent packets would be coming from outside the security domain of the host platform, their code would require stronger security mechanisms to establish identity and trust and would have fewer privileges than the mobile code originating from within the security domain of the host platform.

5.3. Personal Digital Assistants (PDA)

Manufacturers of cell phones, personal organizers, car radios, and other consumer electronic devices are introducing more and more functionality into their products and are becoming the focus of agent developers. One can already take advantage of voice recognition technology to speak to a car radio, have it map out directions, arrange to send a fax, or consult a personal calendar to schedule an important meeting. These devices are not on-line continuously and can benefit from a mobile agent's ability to operate autonomously from the host that launched it. Agent developers often cite the example of a user launching an agent to make travel, hotel, and dinner reservations by negotiating with other agents, as an illustrative scenario for mobile agent technology [1, 4, 27].

Most agent users are unlikely to be developers of agents and will either download agents from a reputable vendor or use agents already installed on their electronic devices. These scenarios might be characterized by transactions where the user only sends out agents and receives the same agent back, but hosts few, if any, agents from any other source. Moreover, the human user’s digital assistant will likely have a smaller platform footprint than commercial platforms and naturally offer fewer services. In this type of environment, the primary security concerns are confidentiality, non-repudiation, mutual authentication.

A number of conventional security mechanisms are being applied to mobile agent systems. The PKI serves as a foundation for mobile agent security services and makes authentication, non-repudiation, and encryption readily available to agent developers and users. Although they fall short of addressing all the security threats of the mobile agent computing paradigm, there are a number of scenarios in which they may offer adequate protection. For example, assigning sensitive tasks only to static agents and restricting mobility between trusted parties adds an additional level of assurance that may

complement existing security mechanisms.

Although conventional encryption techniques can be applied to both the agent and its messages, even if the messages from a PDA to another platform are encrypted, eavesdroppers may be able to infer information about the user's intentions based on the destination of messages. For example, if the agent is sending messages or visiting computer hardware vendors, a software vendor may begin sending unsolicited advertisements for software or computer accessories. Since disconnected operation places limits on the mobile agent's ability to use private keys stored on the human user's PDA, and carrying its private key with it is not safe, time critical transactions could take place on a third party trusted platform that is continuously available, unlike a personal digital assistant.

Just as banks, credit card companies, and insurance companies offer a number of complementary financial and legal services, similar services may be provided for agent users. A human user may use a portfolio analysis and stock broker agent developed by a financial institution for which the user is provided with a level of insurance similar to the Federal Deposit Insurance Corporation (FDIC) and the liability to fraud is limited in a similar way as today's consumer credit cards. These institutions would be responsible for the safety of the agent systems, and in return could differentiate themselves from their competitors and generate new sources of income from these new agent-based services.

6. Security, Design, and Performance Issues

A number of advantages of using mobile code and mobile agent computing paradigms have been proposed [4, 36, 42]. These advantages include: overcoming network latency, reducing network load, executing asynchronously and autonomously, adapting dynamically, operating in heterogeneous environments, and having robust and fault-tolerant behavior.

One of the main obstacles to the widespread adoption of mobile agents is the legitimate security concerns of system developers, network administrators, and information officers. It has been argued that once the security issues have been resolved and a collection of security mechanisms have been developed to counter the associated risks, then the users of mobile agent technology will be free to develop useful and innovative solutions to existing problems and find a wide array of application areas that will benefit from this technology. Using this collection of security mechanisms to mitigate agent-to-agent, agent-to-platform, and platform-to-agent security risks may, however, introduce performance constraints that could dictate design decisions or negate the benefit of using mobile agents for certain applications.

The selection of security mechanisms to employ for a particular application must, therefore, be carefully considered during the design phase of an agent system development cycle and not be added at the end as an additional feature. Security issues may determine which agents are mobile and which agents remain stationary. Security issues may also

determine the functions that a mobile agent is designed to perform and those that it should never perform. The following subsections briefly discuss the impact of various security issues on the design and performance of mobile agent systems.

6.1. Overcoming Network Latency

Mobile agent solutions have been proposed for critical systems that need to respond to changes in their environments in real time. An example of such an application is the use of mobile agents to control robots employed in distributed manufacturing processes. Mobile agents have been offered as a solution, since they can be dispatched from a central controller to act locally and directly execute the controller's instructions. These distributed manufacturing processes may allow different companies to use their special-purpose machining tools and run proprietary data-analysis algorithms. Since the mobile agent resides on the robot's host platform, the control process is not subject to network latencies in control messages sent across slow, unreliable, or unpredictable networks. Although this approach may resolve network latency problems it also raises serious network security issues. For example, the mobile agent may want to keep trade secrets hidden from the platform, while the platform wants to protect itself from unauthorized agents issuing instructions to the machining tools or accessing data belonging to other agents.

Virtual machines and interpreters make mobility in a heterogeneous environment possible, but an interpreted program typically runs slower than an equivalent program that uses native code instructions. Making matters worse, the security mechanisms needed to provide resource control, to protect the platform from the agent, and to conceal agent data from the platform, may impose unacceptable performance costs that prevents the agent from delivering the performance necessary for real-time applications. Thus, the designer of the mobile agents used in distributed manufacturing applications must make important tradeoffs between security and performance. Developers and researchers have made increasing virtual machine and interpreter performance a top priority, and faster hardware is continuously being introduced to the market, but the quest for better performance will always remain.

Straßer and Schwem have contrasted the performance of mobile agent solutions with Remote Procedure Calls (RPC) [37]. An RPC allows a procedure to be executed on a remote server, transfer the control flow and some arguments, from the client to the server, until the request is executed and the results are returned. The authors developed a performance model for mobile agent systems in which agents can alternatively use RPC or agent migration to interact with applications on other platforms. The authors concluded that the selection of RPC or agent migration depended on several factors, including network delay, throughput, migration overhead, number of messages, number of platforms involved, and code caching. The authors did not evaluate the impact of adding security to the performance model. The addition of encryption adds an additional overhead to both the RPC and the mobile agent communication cost. In RPC, as the number of messages increases, so does the overhead cost of encrypting them. In contrast, a mobile agent would only need to be encrypted twice, once on the way to the host, and once on the way back.

In the cases of lightweight agents and frequent RPC message passing, lightweight mobile agents may be cast in a better light as the overhead of encrypting a single lightweight agent should be lower than the overhead of encrypting several RPC messages. As the agent becomes more mobile and migrates more frequently, however, the agent migration overhead related to security increases, since the agent is encrypted/decrypted at each hop.

6.2. Reducing Network Load

Mobile agents are well suited for search and analysis problems involving multiple distributed resources that require specialized tasks that aren't supported by the data server [27, 42]. A mobile agent-based search and data analysis approach can help decrease network traffic resulting from the transfer of large amounts of data across a network for local processing. Instead of transferring the data across the network, mobile agents can be dispatched to the machine on which the data resides, essentially moving the computation to the data, instead of moving the data to the computation, thus reducing the network load for such a scenario. Clearly, transferring an agent that is smaller in size than the data to be transferred reduces the network load. These benefits hold when the comparison is made between encrypted lightweight mobile agents and the relatively larger data to be transferred.

These benefits, however, cannot be realized unless the appropriate security mechanisms are in place. The risks associated with allowing access to public data through well-defined interfaces are relatively low, but the same cannot be said of allowing mobile code to access to local resources. For example, any web site on the internet can submit a remote query to portal search engines such as Yahoo! or get stock quotes from NASDAQ's web site, although the practice is discouraged for commercial uses, without any security mechanisms restricting their accesses. This is possible because the parameters of the query are well defined and understood by the server, and the query poses a low risk to a secure computing environment or to the information provider’s business operations. If someone wanted to perform a proprietary analysis of the voluminous stock information stored on the NASDAQ server, however, the system administrator would have good reason to be reluctant to allow someone else's code to execute on their machines. Thus, the benefits of using mobile agents to reduce the network load and augment the server functionality depends on the availability of secure mobile agent frameworks before they can be realized.

6.3. Asynchronous Execution and Autonomy

A lot of attention is being focused on the use of mobile agents with mobile devices such as cellular phones, personal digital assistants, automotive electronics, and military equipment [4]. Their asynchronous execution and autonomy makes them well-suited for applications that use fragile or expensive network connections. A mobile agent can be launched and continue to operate even after the machine that launched it is no longer connected to the network.

Although a mobile agent may have autonomy of movement and is not bound to the host from where it began its execution, it may rely on that host for security services. For

example, a mobile agent may be engaged in electronic commerce and be required to digitally sign documents related to a particular transaction. Mobile agents cannot safely carry their private key with them, and must rely on their home platform or another trusted platform to provide this service. If the mobile agent is engaged in a transaction that requires a digital signature to be provided, it must be able to communicate with its home platform or another trusted platform. For example, if the agent is unable to provide a digitally signed auction bid, the other agents cannot be expected to wait for the agent to resolve its differences with the electronic auction house before they make their next bids. In this example, although the agent has autonomy of movement, its reliance on the home platform for security services limits the types of tasks it can perform autonomously.

Mobile agents typically load their class files dynamically, as needed, from their home platform. The ability to dynamically load classes also has security implications. If the home platform is not available, these class files may be provided by the local host or must be found and transferred from a remote trusted host. Class loading from a remote platform or the local host platform raises a number of security issues. The class files may have been modified in such a way as to alter the functionality of the agent or even to allow for eavesdropping of the agent’s transactions. Class versioning problems may also yield unexpected results which the mobile agent may not be able to repudiate.

6.4. Adapting Dynamically

Mobile agents have the ability to sense their execution environment and autonomously react to changes [4]. For example, if the computational load of the host platform is too high and the host's performance doesn't meet the agent's service expectations, the agent can move to another machine that can better satisfy its computational needs. Mobile agents can distribute themselves among the hosts in the network in such a way as to maintain the optimal configuration for solving a particular problem. As long as the agents move to hosts within the same security domain and each security domain places the agents under the same security policy, granting or denying the same privileges it had on its previous platform, security concerns do not adversely affect mobile agents' ability to dynamically adapt to the execution environment.

If the mobile agents begin moving outside their existing security domain and become subject to new security policies, then the way in which they adapt will be influenced, and potentially restricted, by the security policies of other platforms. Mobile agent platforms must, therefore, be able to communicate their security policies to mobile agents, and the mobile agents must be able to evaluate different security policies in addition to the available resources before deciding on their next destination. Communicating, negotiating, and managing mobile agent security policies also introduces new security administration costs for the mobile agent platform.

6.5. Operating in Heterogeneous Environments

Since mobile agents are generally computer- and transport-layer-independent, and dependent only on their execution environment, they offer an attractive approach for

heterogeneous system integration. Mobile agents' ability to operate in heterogeneous computing environments is made possible by virtual machines or interpreters on the host platform. The benefits of heterogeneity, however, introduce several new security concerns. The current implementations of virtual machines or interpreters that make heterogeneous computing environments possible, however, do not provide adequate support for resource control. For example, Java currently provides no way for the host to limit the processor and memory resources allocated to a given object or thread and is, therefore, susceptible to denial of service attacks. A related issue is the ability of the agent to allocate resources external to the program, for example, by opening files and sockets, and creating windows. In addition, the current Java VM offers no protection from references to an object's public methods. A Java object's public methods and the data to which they may provide access are available to any other object that has a reference to them. There is currently no way for an agent to directly monitor or control which agents are accessing its methods.

6.6. Robust and Fault-Tolerant Behavior

The ability of mobile agents to react dynamically to unfavorable situations and events makes it easier to build robust and fault-tolerant distributed systems. For example, if a host is being shut down, all agents executing on that machine are warned, whenever possible, and given time to dispatch and continue their operation on another host in the network.

The ability of the mobile agents to move from one platform to another in a heterogeneous environment has been made possible by the use of virtual machines and interpreters. Virtual machines and interpreters, however, can offer only limited support for preservation and resumption of the execution state in heterogeneous environments, because of differing representations in the underlying hardware. For example, although a number of research efforts are underway to address this issue, the full execution state of an object cannot currently be retrieved in Java[2]. Information such as the status of the program counter and frame stack is currently off limits for Java programs.

Conventional fault-recovery and check-pointing techniques are challenged even further in the mobile agent computing paradigm. Even though an arsenal of techniques exist to provide security and fault-tolerance, the designer must be careful in selecting which mechanisms to use and how they impact the overall system performance and functionality. For example, check-pointing before and after an agent's arrival, and upon completion of certain transactions or events may be necessary to ensure for acceptable fault-recovery. With each check-pointing procedure and non-repudiation mechanism invoked, however, more overhead is introduced. The fault-recovery techniques may also involve more than one machine, further complicating the recovery.

Although mobile agents possess a great deal of autonomy and perform well in

[2] A number of research projects, such as Nomads at the University of West Florida, are using modified Java Virtual Machines to capture the frame stack.

disconnected operations, the failure of the home platform or other platforms that the agents rely on to provide security services can seriously reduce their intended functionality. Even though a mobile agent can become more fault-tolerant by moving to another machine, the mobile agent's reliance on the safe operation of a safe home or trusted platform places restrictions on its functionality. Designers of mobile agent platforms are also faced with tradeoffs between security and fault-tolerance. For example, in order to address the security risks involved in "multi-hop" agent mobility, some agent architectures have been built on centralized client-server models requiring agents to return to a central server before moving on to another host machine [28]. Clearly, addressing the security risks in this manner renders all the mobile agents vulnerable to a failure of the central server and raises scalability issues.

7. Areas for Future Research

The area of mobile agent security is still in a somewhat immature state. The traditional host orientation toward security persists, and the focus of protection mechanisms within the mobile agent paradigm remains on protecting the agent platform. However, emphasis is moving toward developing techniques that are directed towards protecting the agent, a much more difficult problem. Fortunately, there are a number of applications for agents where conventional and recently introduced security techniques should prove adequate, until further progress can be made.

The next wave of security improvements for agent systems is likely to emerge from the present baseline of protection techniques, either through incremental refinements that reduce processing and storage overhead or simplify the use of the mechanism, or clever combination of complementary mechanisms to form a more effective composite protection scheme. Other peripheral topics currently neglected by researchers are also potential candidates. From the threats and countermeasures reviewed earlier and in the ensuing discussion, there appears to be an opportunity for research along the following lines:

- Agent Security Framework,
- Security Design Tools,
- PKI Privilege Management Extensions, and
- Anonymity.

7.1. Agent Security Framework

In the past, as teams of individuals have developed agent systems, pragmatics prevailed and emphasis was placed on functionality over security. While some agent system implementations incorporate appropriate security techniques, often little regard is given to interoperability among agent systems. What is needed is an overall framework that integrates compatible techniques into an effective security model and provides an umbrella under which interoperability can exist.

The Foundation for Intelligent Physical Agents' (FIPA) '97 and '98 standards and Object Management Group's MASIF standards both fall short in providing the desired framework. The FIPA work is focused mainly on standardizing the agent communication language used among cooperating agents. Many of the details regarding the architecture of the agent platform require significant work before any substantive progress can be made on security. The MASIF standards on the other hand do make a clear and definitive statement on security, relying on the CORBA security services architecture. Unfortunately, although the CORBA model adequately addresses security services for an agent platform, it largely ignores any independent security services needed by an agent.

7.2. Mobile Agent Security Design Tools

Mobile agent application developers currently face a number of obstacles before they can efficiently design and develop large-scale mobile agent systems. These obstacles include: the lack of advanced development and modeling tools, the lack of mature agent standards, and the difficulty in optimizing performance under varying computational and communication loads. The limitations of agent and agent platform security mechanisms must also be overcome before agent developers can realize the full benefits of mobile agent technology. The selection of security mechanisms has a direct impact on agent migration, autonomy, disconnected operation, network latency, performance, and agent messaging. Mobile agent security design tools can help agent system developers determine the effects of employing various security mechanisms and make better decisions about functionality and performance tradeoffs.

7.3. PKI Privilege Management Extensions

Attribute certificates have long been discussed as a means of extending a public key infrastructure to allow users or other issuers to control how their authority is delegated to software that operates on their behalf. The idea is that individuals, whose identity is established through an existing PKI (e.g., PGP, MISSI, PEM, etc.), could delegate their authority by using their private key to sign a specially designed certificate, an attribute certificate. An attribute certificate contains no key material, such as the public key for an entity, but incorporates a message digest of the software along with the privilege and policy delegations. Both ANSI X9 and the IETF have made some initial attempts at standardization in this area, however, the topic has not received much attention to date from the agent community.

The main area needing resolution is how to express privilege and policy within the certificate. The syntax must be able to be processed by a machine, rich enough to capture real-world privileges and policy, and simple enough for people to use. While privilege can be represented easily using a simple "privilege = attribute set" notation often employed in present day agent systems, policy is more difficult, since it must express the protection the agent must receive in conducting its activities.

7.4. Anonymity

This topic refers to protection mechanisms for preserving the anonymity of a user on whose behalf an agent operates. Most of the work on this topic has been done in the context of messaging systems. Anonymity of not only an initiating agent, but also of any recipient agent or intermediaries may apply.

8. Summary

A wide variety of techniques for implementing security in agent systems is available. Not all are compatible with one another, nor are they all suitable for most applications. Many of these techniques must be implemented within the framework of the agent system, while a number of them can be applied independently within the context of the application.

While elementary security techniques should prove adequate for a number of agent-based applications, many applications are expected to require a more comprehensive set of mechanisms. Moreover, to meet the needs of a specific application, a flexible framework must exist in which a subset of mechanisms can be selected and applied. The trick, of course, is to select a comprehensive baseline of countermeasures which meets the philosophy of protection guiding the design of the agent system, fulfills the needs of most applications, includes compatible mechanisms, and can be extended to include other advanced mechanisms that may be invented. Clearly, this is a period where establishing such a baseline requires more experimentation and experience with alternative design choices, including those involving tradeoffs in performance, scalability, and compatibility.

9. References

[1] "Mobile Agents White Paper," General Magic, 1998.
<URL: http://www.genmagic.com/technology/techwhitepaper.html>

[2] A. Fuggetta, G.P. Picco, and G. Vigna, "Understanding Code Mobility," IEEE Transactions on Software Engineering, 24(5), May 1998.
<URL: http://www.cs.ucsb.edu/~vigna/listpub.html>

[3] James Gosling and Henry McGilton, "The Java Language Environment: A White Paper," Sun Microsystems, May 1996.
<URL: http://java.sun.cm/docs/white/langenv/>

[4] Danny Lange and Mitsuru Oshima, Programming and Deploying Java Mobile Agents with Aglets, Addison-Wesley, 1998.

[5] Anurag Acharya, M. Ranganathan, Joel Salz, "Sumatra: A Language for Resource-aware Mobile Programs," in J. Vitek and C. Tschudin (Eds.), Mobile Object Systems: Towards the Programmable Internet, Springer-Verlag, Lecture Notes in Computer Science No. 1222, pp. 111-130, April 1997.
<URL: http://www.cs.umd.edu/~acha/papers/lncs97-1.html>

[6] "Agent Management," FIPA 1997 Specification, part 1, version 2.0, Foundation for Intelligent Physical Agents, October 1998.
<URL: http://www.fipa.ord/spec/fipa97/fipa97.html>

[7] "Mobile Agent System Interoperability Facilities Specification," Object Management Group (OMG) Technical Committee (TC) Document orbos/97-10-05, November 1997.
<URL: http://www.omg.org/techprocess/meetings/schedule/Technology_Adoptions.html#tbl_MOF_Specification>

[8] Markus Straßer, Joachim Baumann, Fritz Hohl, "Mole - A Java Based Mobile Agent System," in M. Mühlhäuser (ed.), Special Issues in Object Oriented Programming, Verlag, 1997, pp. 301-308.
<URL: http://www.informatik.uni-stuttgart.de/ipvr/vs/projekte/mole/ECOOP96.ps.gz>

[9] "ObjectSpace Voyager Core Package Technical Overview," version 1.0, ObjectSpace Inc., December 1997.
<URL: http://www.objectspace.com/developers/voyager/white/index.html>

[10] Joseph Tardo and Luis Valente, "Mobile Agent Security and Telescript," Proceedings of IEEE COMPCON '96, Santa Clara, California, pp. 58-63, February 1996, IEEE Computer Society Press.

[11] Robert S. Gray, "Agent Tcl: A Flexible and Secure Mobile-Agent System," Proceedings of the Fourth Annual Tcl/Tk Workshop (TCL 96), pp. 9-23, July 1996.
<URL: http://actcomm.dartmouth.edu/papers/#security>

[12] John K. Ousterhout, Jacob Y. Levy, and Brent B. Welch, "The Safe-Tcl Security Model," Technical Report SMLI TR-97-60, Sun Microsystems, 1997.

[13] Li Gong, "Java Security Architecture (JDK 1.2)," Draft Document, revision 0.8, Sun Microsystems, March 1998.
<URL:http://pcba10.ba.infn.it/api/jdk1.2beta3/docs/guide/security/spec/security-spec.doc.html>

[14] Günter Karjoth, Danny B. Lange, and Mitsuru Oshima, "A Security Model For Aglets," IEEE Internet Computing, August 1997, pp. 68-77.

[15] "Odyssey Information," General Magic Inc., 1998.
<URL: http://www.genmagic.com/technology/odyssey.html>

[16] William Farmer, Joshua Guttman, and Vipin Swarup, "Security for Mobile Agents: Authentication and State Appraisal," Proceedings of the 4th European Symposium on Research in Computer Security (ESORICS '96), September 1996, pp. 118-130.

[17] G. Necula and P. Lee, "Safe Kernel Extensions Without Run-Time Checking," Proceedings of the 2nd Symposium on Operating System Design and Implementation (OSDI '96), Seattle, Washington, October 1996, pp. 229-243.
<URL: http://www.cs.cmu.edu/~necula/papers.html>

[18] Giovanni Vigna, "Protecting Mobile Agents Through Tracing," Proceedings of the 3rd ECOOP Workshop on Mobile Object Systems, Jyvälskylä, Finland, June 1997.
<URL: http://www.cs.ucsb.edu/~vigna/listpub.html>

[19] Volker Roth, "Secure Recording of Itineraries Through Cooperating Agents," Proceedings of the ECOOP Workshop on Distributed Object Security and 4th Workshop on Mobile Object Systems: Secure Internet Mobile Computations, pp. 147-154, INRIA, France, 1998.
<URL: http://www.igd.fhg.de/www/igd-a8/pub/#Mobile Agents>

[20] Joann J. Ordille, "When Agents Roam, Who Can You Trust?" Proceedings of the First Conference on Emerging Technologies and Applications in Communications, Portland, Oregon, May 1996.
<URL: http://cm.bell-labs.com/cm/cs/doc/96/5-09.ps.gz>

[21] Thomas Sander and Christian Tschudin, "Protecting Mobile Agents Against Malicious Hosts," in G. Vinga (Ed.), Mobile Agents and Security, Springer-Verlag, Lecture Notes in Computer Science No. 1419, 1998.
<URL: http//www.icsi.berkeley.edu/~tschudin/>

[22] "Trusted Computer System Evaluation Criteria," Department of Defense, CSC-STD-001-83, Library No. S225 711, August 1983.
<URL: http://www.cru.fr/securite/Documents-generaux/orange.book>

[23] R. Wahbe, S. Lucco, T. Anderson, "Efficient Software-Based Fault Isolation," Proceedings of the Fourteenth ACM Symposium on Operating Systems Principles, ACM SIGOPS Operating Systems Review, pp. 203-216, December 1993.
<URL: http://www.cs.duke.edu/~chase/vmsem/readings.html>

[24] John K. Ousterhout, "Scripting: Higher-Level Programming for the 21st Century," IEEE Computer, March 1998, pp. 23-30.

[25] Neeran Karnik, "Security in Mobile Agent Systems," Ph.D. Dissertation, Department of Computer Science, University of Minnesota, October 1998.
<URL: http://www.cs.umn.edu/Ajanta/>

[26] Nayeem Islam, et al., "A Flexible Security System for Using Internet Content, IEEE Software, September 1997, pp. 52-59.

[27] David Chess, Benjamin Grosof, Colin Harrison, David Levine, Colin Parris, and Gene Tsudik, "Itinerant Agents for Mobile Computing," IEEE Personal Communications, vol. 2, no. 5, October 1995, pp. 34-49.

[28] "Jumping Beans Security," Ad Astra Engineering.
<URL: http://www.jumpingbeans.com/Security.html>

[29] A. Young and M. Yung, "Sliding Encryption: A Cryptographic Tool for Mobile Agents," Proceedings of the 4th International Workshop on Fast Software Encryption, FSE '97, January 1997.

[30] Bennet S. Yee, "A Sanctuary for Mobile Agents," Technical Report CS97-537, University of California in San Diego, April 28, 1997.
<URL: http://www-cse.ucsd.edu/users/bsy/index.html>

[31] Stuart Haber and Scott Stornetta, "How to Time-Stamp a Digital Document," Journal of Cryptology, vol. 3, pp. 99-111, 1991.

[32] F.B. Schneider, "Towards Fault-Tolerant and Secure Agentry," Proceedings 11th International Workshop on Distributed Algorithms, Saarbucken, Germany, September 1997.

[33] James Riordan and Bruce Schneier, "Environmental Key Generation Towards Clueless Agents," G. Vinga (Ed.), Mobile Agents and Security, Springer-Verlag, Lecture Notes in Computer Science No. 1419, 1998.

[34] Fritz Hohl, "Time Limited Blackbox Security: Protecting Mobile Agents From Malicious Hosts," G. Vinga (Ed.), Mobile Agents and Security, pp. 92-113, Springer-Verlag, Lecture Notes in Computer Science No. 1419, 1998.

[35] Anderson, J. "Computer Security Technology Planning Study," U.S. Air Force Electronic Systems Division Technical Report, pp. 73-51, October 1972.

[36] Jonathan Smith, “A Survey of Process Migration Mechanisms,” Operating Systems Review, 22(3), ACM Special Interest Group on Operating Systems, pp. 28-40, July 1988.

[37] M. Straßer and M. Schwem, “A Performance Model for Mobile Agent Systems,” H. Arabnia (Ed.), Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'97), Vol. II, CSREA, pp. 1132-1140, 1997.

[38] G. Karjoth, N. Asokan, and C. Gülcü, “Protecting the Computation Results of Free-Roaming Agents,” Second International Workshop on Mobile Agents, Stuttgart, Germany, September 1998.

[39] Vinod Anupam and Alain Mayer, “Secure Web Scripting,” IEEE Internet Computing, vol. 2, no. 7, November/December 1998.

[40] Grzegorz Czajkowski and Thorsten von Eicken, “JRes: A Resource Accounting Interface for Java,” ACM Conference on Object Oriented Languages and Systems (OOPSLA), Vancouver, Canada, October 1998.

[41] Alper Caglayan and Colin Harrison, Agent Sourcebook: A Complete Guide to Desktop, Internet, and Intranet Agents, Wiley Computer Publishing, 1997.

[42] David Chess, Colin Harrison, and Aaron Kershenbaum, “Mobile Agents: Are They a Good Idea?” IBM Research Report RC 19887 (88465), December 1994.

[43] “Jumping Beans White Paper,” Ad Astra Engineering Inc., Sunnyvale CA, December 1998.