MOBILE AGENTS AND SECURITY

W. A. Jansen
National Institute of Standards and Technology
Gaithersburg, MD 20899, USA
jansen@nist.gov

Abstract

Mobile agent technology offers a new computing paradigm in which 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 security vulnerabilities. This paper gives an overview of mobile agents and their general architecture, and reviews inherent vulnerabilities of this technology and current approaches for addressing them.

MOBILE AGENTS AND SECURITY

Introduction

Over the years computer systems have successfully 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 occurred: 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 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. Whether this evolutionary branch successfully propagates forward or eventually dies out, remains to be seen. Nevertheless, the technology is intriguing and offers a new paradigm for computing that cannot be summarily dismissed.

The catalysts for this evolutionary path are mobile software agents – programs that are goal-directed and capable of suspending themselves 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 [0] 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. [1]. For example, a simple agent written as a Java applet [2] has mobility of code through the movement of class files from a web server to a browser. However, no associated state information is conveyed. In contrast, Aglets [3], 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 [4], 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.

Agent Model

A number of models exist for describing agent systems for comparative [1] or standardization purposes [5, 6]. 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 comprises the code and state information needed to carry out some computation. As mentioned earlier, the computation is typically a goal-directed task, performed autonomously by the agent on behalf of some individual, with the cooperation of other agents. An agent's code tends to be static or unchanging, while its state information may vary dynamically, reflecting the results of the actions taken by the agent. Mobility allows an agent to move or hop among agent platforms. A mobile agent can be portrayed as a guest that temporarily visits various agent platforms, which

host its activities. One benefit of mobility is that the knowledge and know-how embodied in the agent can be positioned where it is most advantageous to the task at hand.

The agent platform provides the computational environment in which an agent operates. The platform where an agent originates is referred to as the home platform, and normally is the most trusted environment for an agent. The platform assigns a newly originated or incoming agent to a requested location or place, where it can compute and interact with other agents. Besides furnishing the engine on which an agent executes its code, typical services offered by an agent platform include the capability for an agent to clone itself, spawn or create new agents, terminate any spawned agents, locate other agents at the platform or a platform elsewhere, send messages to other agents, and relocate itself on another platform. One or more hosts may comprise an agent platform, and an agent platform may support more than one meeting place. However, these more detailed aspects do not advance the discussion of security issues and, therefore, are not reflected in the agent model depicted in Figure 1.

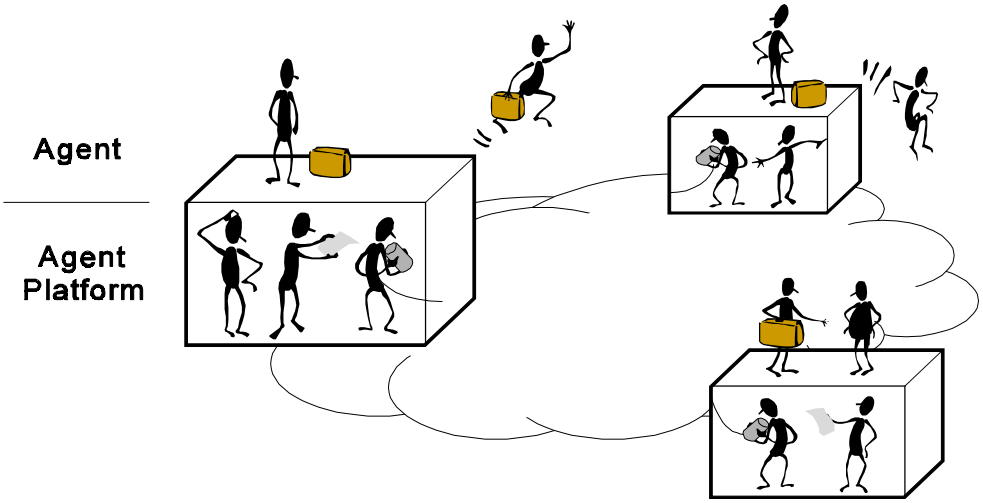

Figure 1: Simple Agent Model

For pedagogical reasons, the agent platform itself can be thought of as a collection of special purpose agents working together to provide the services and computational environment for other, normal run-of-the-mill agents. This perspective is similar to how an operating system kernel is composed of trusted processes that enable user processes to operate by performing basic functions such as scheduling and memory management. To relocate itself, for example, an agent conceptually invokes a service (i.e., calls upon a trusted agent within the platform), which captures and packs the state information of the calling agent, serializes the calling agent's code, and dispatches the archived representation to the requested destination. While this perspective is useful in discussing the behavior and organization of the internals of the agent platform, it is only illustrative, and in no way implies that an agent platform must be implemented accordingly.

Threat Overview

Threats to security generally fall into four comprehensive classes: disclosure of information, denial of service, corruption of information, and interference or nuisance. There are a variety of ways to examine in greater detail these classes of threats as they apply to agent systems. Here, we use the components of the agent model to categorize the threats, as a way to identify the possible source and target of an attack. It is important to note that many of the threats that are discussed have counterparts in classical 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.

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 threat categories are illustrated in Figure 2. The cases of an agent attacking an agent on another agent platform and of an agent platform attacking another platform are covered within the last category, since these attacks are focussed primarily on the communications capability of the platform to exploit potential vulnerabilities.

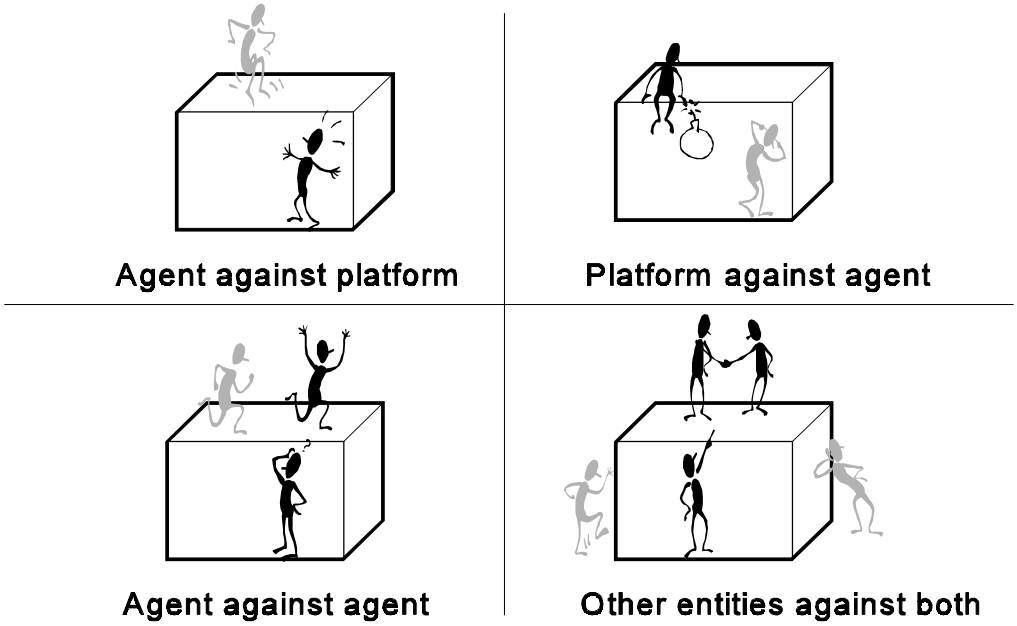

Figure 2: Threat Categories

Agent Against Agent Platform

An incoming agent has two main lines of attack. The first is to gain unauthorized access to information residing at the agent platform; the second is to use its authorized access in an unexpected and disruptive fashion. Unauthorized access may occur simply through a lack of adequate access control mechanisms at the platform or through masquerading as an agent trusted by the platform. Once access is gained, information can be disclosed or information residing at the platform, including instruction codes, can be altered. Depending on the level of access, the agent may be able to completely shut down or terminate the agent platform. Even without gaining unauthorized access to resources, an agent can deny platform services to other agents by exhausting computational resources, if resource constraints are not established or not set tightly. Otherwise, the agent can merely interfere with the platform by issuing meaningless service requests wherever possible.

Agent Platform Against Agent

A receiving agent platform can easily isolate and capture an agent and may attack it by extracting information, corrupting or modifying its code or state, denying requested services, or simply reinitializing or terminating it completely. An agent is very susceptible to the agent platform and may be corrupted merely by the platform responding falsely to requests for information or service, or delaying the agent until its task is no longer relevant. Extreme measures include the complete analysis and reversing engineering of the agent's design so that subtle changes can be introduced. Modification of the agent by the platform 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) or the accuracy of the computation (e.g., changing collected information to yield incorrect results).

Agent Against Other Agents

An agent can target another agent using several general approaches. These include actions to falsify transactions, eavesdrop upon conversations, or interfere with an agent's activity. For example, an attacking agent can respond falsely to direct requests it receives from a target or even deny that a legitimate transaction occurred. An agent can gain information by serving as an intermediary to the target agent (e.g., through masquerade) or by using platform services to eavesdrop on intra-platform messages. If the agent platform has weak or no control mechanisms in place, an agent can even directly interfere with another agent by invoking its public methods (e.g., attempt buffer overflow, reset to initial state, etc.), or access and modify the agent's data or code.

Other Entities Against Both

Even assuming the agents and agent platform are well behaved, other entities both outside and inside the agent framework may attempt actions to disrupt, harm, or subvert the agent framework. The obvious methods involve attacking the inter-agent and inter-platform communications through masquerade, (e.g., through forgery or replay) or intercept. For example, at a level of protocol below the agent-to-agent or platform-to-platform protocol, an entity may eavesdrop on messages in transit to and from a target agent or agent platform to gain information. An attacking entity may also intercept agents or messages in transit and modify their contents, substitute other contents, or simply replay the transmission dialogue at a later time in an attempt to disrupt the synchronization or integrity of the agent framework.

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 specially 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. In [7], Farmer et al. provide an alternate perspective by categorizing, from easy to impossible, commonly sought security objectives and associated conventional techniques that can be applied.

Protecting an Agent Platform

Without adequate defenses, an agent platform is vulnerable to attack from many sources including malicious agents, malicious platforms, and other malicious entities. Fortunately most conventional protection techniques, traditionally employed in trusted systems and communication security, can be used to provide analogous protection mechanisms for the agent platform. This is due in large part to the traditional role hardware plays as the foundation upon which software protection mechanisms are built. That is, within the mobile agent paradigm, the agent platform is a counterpart to a trusted host within the traditional framework. Conventional security 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, agent, and platforms, and
- Mechanisms to audit security relevant events occurring at the agent platform.

Similarly, more recently developed techniques aimed at mobile code and applicable to mobile agent security have evolved along those same traditional lines. They include the following:

- Developing agents using an interpreted script or programming language,
- Limiting the capabilities of agent languages so that they are considered "safe,"
- Applying digital signatures to agents and other information to indicate authenticity, and
- Restricting an agent's capabilities at an agent platform by:
  - constraining resources (e.g., lifetime, storage),
  - controlling service access (e.g., network destinations, directory segment), and
  - making capabilities location dependent.

Java programming language and runtime environment [2] illustrate the nature of the recently developed techniques listed above. There are many agent systems based on Java, including Aglets [3], Mole [8], and Voyager [9]. The Java environment includes built-in security controls such as sandboxing for isolation of code into mutually exclusive execution domains, and byte code verification to check the safety of class file downloads. It also inherently supports code mobility, dynamic class loading, digitally signed code, object serialization, platform heterogeneity, and other features that make it an ideal foundation for agent development. The Java security model for version 1.2 [13] contains a new permission-based mechanism for constraining the computational capabilities of mobile code, which also can benefit agent systems based on Java. Each permission specifies the authorized access to a particular resource, such as a connect permission to allow access to a given host and port. The Aglet security model [14], for example, to a large degree reflects Java's underlying protection mechanisms.

Other notable interpreted systems include Telescript [10] and Agent TCL [11]. The latter is based on Safe TCL [12], which employs a padded cell concept as a counterpart to Java sandboxing. The term padded cell denotes an isolation technique whereby a second interpreter pre-screens any harmful commands from being executed by the main interpreter. Similar mechanisms, to those in Java for constraining agents, have been built into Telescript, Agent TCL, and Telescript's successor, Odyssey [15].

Protecting an Agent

Because an agent is completely a software entity, the traditional view that hardware protects software applies only when the set of platforms an agent visits can be trusted to some degree. Assuming an agent trusts its home platform to provide the required support services and not subvert its activities, countermeasures in the form of conventional security techniques can be applied on behalf of the agent via the platform. These measures rely primarily on identifying and authenticating trusted parties with whom to interoperate and include the following:

- Issue users and agent platforms public key certificates for strong authentication,
- Convey information (e.g., agents and messages) securely (i.e., with confidentially, integrity, source authentication, and non-repudiation) among agent platforms,
- Detect and ignore replay attacks against agent platforms, and
- Enable an agent to audit platform services and other security related events for post processing analysis and detection.

Limits of Conventional Countermeasures

Conventional security controls generally work fine for static agents, but eventually break down as mobility becomes increasingly unrestricted. The basic dilemma is that agents need to travel and work autonomously, but protecting agents from unknown agent platforms is very difficult to achieve. This vulnerability results in a double-edged problem. When an agent hops between

platforms, the receiving platform cannot determine whether tampering has occurred, and the agent, in turn, cannot determine whether the agent platform is malicious.

Single Hop Solution

An agent is the most secure when residing on its home platform, since it is the place where the agent is instantiated and commences activity. While an agent and home platform are not impervious to attack, conventional countermeasures can be used to build adequate defenses, provided agents remain static. When introducing mobility, a way is needed to extend the trusted computing environment offered by the home platform to other platforms being visited. Limiting the agent's itinerary to only those agent platforms trusted by the home platform helps to accomplish this objective.

A single hop solution is an analogue to client-server security arrangements, whereby an agent's home platform (i.e., the client side) authenticates a second platform (i.e., the server side) on behalf of the agent before any transactions occur. After the authentication exchange, instead of an exchange of messages over the network between client and server counterparts, the agent simply moves to the other platform, issues its transactions locally, and returns with or sends back the results. The solution maintains security by restricting an agent's itinerary to trusted agent platforms, authenticated firsthand by its home platform (i.e., reached via a single hop), and by having the home platform digitally sign and/or encrypt the agent before it moves. The receiving platform is also protected, because the agent comes directly from an authenticated source, which vouches for the code and state information through a digital signature and, if encryption is applied, protects the contents from disclosure.

Figure 3 gives an illustration of the single hop solution that employs some of the conventional countermeasures mentioned earlier in the paper and referred to above. Each agent platform is assumed to hold its own public-private key pair and the public key certificate of the other platform, which it has verified during the mutual authentication protocol exchange. A secret session key, shared by both platforms, is established and used to conceal the serialized agent and its state information during transmission. A shroud is used to depict an encrypted object. Both the code, represented by the agent's body, and state information, represented by the suitcase, are also signed by the sending platform, the agents home platform. Once an agent arrives at the receiving platform, a set of constraints, represented by a shackle, is applied.

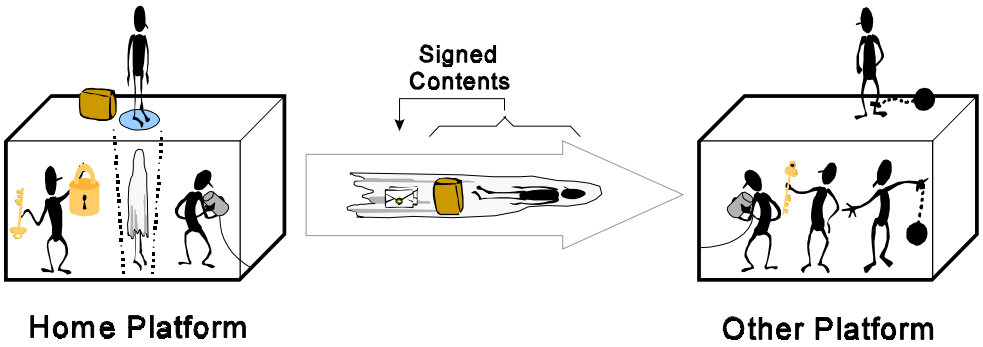

Figure 3: Single Hop Illustration

Bilateral trust does not prevent either party from behaving maliciously. However, using conventional security techniques, the home platform can readily assess whether another agent platform is illegitimate or suspect. For example, if the credentials of the second platform are invalid or the signature on the results of the computation is invalid, the home platform can take appropriate action. Moreover, if the results are signed by the other platform to indicate they are

authentic, and eventually shown to be incorrect or fraudulent, then compensation may be attainable through legal means. Similarly, if the agent behaves maliciously after being designated as a legitimate agent by the home platform, compensation may be attainable from the owners of the agent platform or the offending agent.

The Multi-Hop Problem

The main weakness in applying conventional techniques beyond a single hop is that bilateral trust relationships are not, in general, transitive relationships. If Ray trusts Sally and Sally trusts Tim, it doesn't necessarily mean that Ray trusts Tim. Furthermore, trust relationships may not necessarily be reciprocal. If Ray trusts Sally, it doesn't mean that Sally trusts Ray. Therefore, even if an agent's initial code and state information is signed by its home platform, an intermediate agent platform in a multi-hop itinerary may alter the agent's accumulated state information, making the agent suspect after the initial hop. Moreover, intermediate results accumulated in an agent's state may be easily viewed and responses to inquiry transactions made at an agent platform may be colored accordingly. For example, an agent searching for the lowest price of a specific music CD may find that after the first quote, subsequent quotes are each a penny less than that of the previous low.

Fortunately, a number of specialized applications exist whose characteristics are amenable to using conventional techniques with multi-hop itineraries. For example, consider those applications where an agent may be trusted regardless of where it has been, provided it doesn't change state and an acceptable authority signs it. Conventional techniques can also prove satisfactory in a narrow range of applications where the itinerary can be restricted to platforms that are members of a closed trusted domain. In more general situations, conventional techniques become limited quickly.

Some New Approaches

Besides the obvious approaches of restricting agents to fixed itineraries within a network of trusted platforms, or restricting agents to travel only one authenticated hop away from home, a number of novel approaches have been proposed. They include the following ideas:

- Subject agents to state appraisal as a compliment to signed code,
- Require agents to convey proof of safety properties of its code,
- Require agents to maintain a record of the agent platforms visited,
- Require agent platforms to maintain execution traces of an agent's code, and
- Enable agents to execute encrypted functions safely at an agent platform.

As with the traditional techniques, the focus of the new approaches is primarily on protecting the agent platform from malicious agents, rather than the reverse. The last two items, however, on execution traces and computing with encrypted functions, offer some hope for an eventual solution that is effective.

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 in advance of 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.

Proof Carrying Code

This approach taken by Proof Carrying Code [17], obligates the code producer (e.g., the author of an agent) to prove formally that the program possess safety properties previously stipulated by the code consumer (e.g., security policy of the agent platform). 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. 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.

Path Histories

The basic idea behind Path Histories [18] is to have an agent maintain an authenticable record of the prior platforms visited, 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. 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.

Path Histories is a protection measure proposed by several individuals, with slightly different variations. One interesting variation of the general scheme [19] is allowing an agent's path history to be held by another cooperating agent and vice-versa, 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; have the peer

maintain a record of the itinerary; and have the peer take appropriate action when inconsistencies are noted. Some drawbacks mentioned include the cost of setting up the authenticated channel and the inability for the peer to determine which of two platforms is responsible if the agent is killed.

Execution Tracing

Execution tracing [20] is a technique for detecting unauthorized modifications of an agent, including to its state, through faithful recording of the agent's behavior during execution at 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 identifier 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.

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.

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 inside 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, Alice has an algorithm to compute a function f. Bob has input x and wants to compute f(x) for Alice, but she doesn't want Bob to learn anything about f. If f can be encrypted in a way that results in another function E(f), then Alice can create a program P(E(f)), which implements E(f), and send it to Bob, embedded within her agent. Bob then runs the agent, which executes P(E(f)) on x, and returns the result to Alice 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 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. Moreover, should the technique become practical for widespread application, the degree of commercial acceptance for its use at agent platforms remains an open question.

Summary

The area of mobile agent security is in a state of immaturity. While numerous techniques exist to provide security for mobile agents, there is not at present an overall framework that integrates compatible techniques into an effective security model. 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 slowly moving toward developing techniques that are oriented toward protecting the agent, a much more difficult problem. Fortunately, there are a number of applications where conventional and emerging security techniques should prove adequate, if applied judiciously.

References

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

[1] A. Fuggetta, G.P. Picco, and G. Vigna, "Understanding Code Mobility," IEEE Transactions on Software Engineering, 24(5), May 1988
<URL: http://www.cs.ucsb.edu/~vigna/listpub.html>

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

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

[4] 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>

[5] "Agent Management," FIPA '97 Specification, part 1, version 2.0, Foundation for Intelligent Physical Agents, October 1998
<URL: http://www.fipa.ord/spec/fipa97/fipa97.html>

[6] "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

[7] William Farmer, Joshua Guttman, and Vipin Swarup, "Security for Mobile Agents: Issues and Requirements," Proceedings of the 19th National Information Systems Security Conference, Baltimore, MD, pp. 591-597, October 1996
<URL: http://csrc.nist.gov/nissc/1996/papers/NISSC96/paper033/>

[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, OR, May 1996
<URL: http://cm.bell-labs.com/cm/cs/doc/96/5-09.ps.gz>

[21] Thomas Sander and Christian Tshudin, ”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/>