Security for Mobile Agents: Issues and Requirements [*]

William M. Farmer, Joshua D. Guttman, and Vipin Swarup

The MITRE Corporation
202 Burlington Road
Bedford, MA 01730-1420
{farmer,guttman,swarup}@mitre.org

Abstract

Mobile agents are processes which can autonomously migrate to new hosts. Despite its many practical benefits, mobile agent technology results in significant new security threats from malicious agents and hosts. The primary added complication is that, as an agent traverses multiple hosts that are trusted to different degrees, its state can change in ways that adversely impact its functionality. In this paper, we investigate these new threats and develop a set of achievable security requirements for mobile agent systems.

1 Introduction

Currently, distributed systems employ models in which processes are statically attached to hosts and communicate by asynchronous messages or synchronous remote procedure calls. Mobile agent technology extends this model by including mobile processes, i.e., processes which can autonomously migrate to new hosts. This basic idea results in numerous benefits including flexible, dynamic customization of the behavior of clients and servers and robust remote interaction over unreliable networks.

Threats, vulnerabilities, and countermeasures for the currently predominating static distributed systems have been studied extensively; sophisticated distributed system security architectures have been designed and implemented [11, 14]. These architectures use the access control model, which provides a basis for secrecy and integrity security policies. In this model, objects are resources such as files, devices, processes, and the like; principals are entities that make requests to perform operations on objects. A reference monitor is a guard that decides whether or not to grant each request based on the principal making the request, the operation requested, and the access rules for the object.

The process of deducing which principal made a request is called *authentication*. In a distributed system, authentication is complicated by the fact that a request may originate on a distant host and may traverse multiple machines and network channels that are secured in different ways and are not equally trusted [11]. The process of deciding whether or not to grant a request—once its principal has been authenticated—is called *authorization*. The authentication mechanism underlies the authorization mechanism in the sense that authorization can only perform its function based on the information provided by authentication, while conversely authentication requires no information from the authorization mechanism.

Despite its many practical benefits, mobile agent technology results in significant new security threats from malicious agents and hosts. The primary added complication is that, as an agent traverses multiple machines that are trusted to different degrees, its state can change in ways that adversely impact its functionality.

In this paper, we will examine a few different ways of using mobile agents, with the aim of identifying many of the threats and security issues which a meaningful mobile agent security infrastructure must handle. We will develop a set of security requirements for mobile agent systems and will distinguish between those that appear impossible, those that are achievable with current technology, and those that might be achievable with future work. We will not, in this short paper, develop a security model which can meet the achievable requirements, though we think it can be done. See [6] for elements of such a model and [4, 9, 13, 15, 16] for related work on mobile agent security.

[*]This work was supported by the MITRE-Sponsored Research Program.

2 Mobile Agents

A mobile agent is a program that can migrate from one networked computer to another while executing. This contrasts with the client/server model where non-executable messages traverse the network, but the executable code remains permanently on the computer it was installed on. Mobile agents have numerous potential benefits. For instance, if one needs to perform a specialized search of a large free-text database, it may be more efficient to move the program to the database server rather than move large amounts of data to the client program.

In recent years, several programming languages for mobile agents have been designed. These languages make different design choices as to which components of a program's state can migrate from machine to machine. In Java [12], only program code can migrate; no state is carried with the programs. In Obliq [2], first-class function values (closures) can migrate; closures consist of program code together with an environment that binds variables to values or memory locations. In Kali Scheme [3], again, closures can migrate; however, since continuations [10, 8] are first-class values, Kali Scheme permits entire processes to migrate autonomously to new hosts. In Telescript [18], functions are not first-class values; however, Telescript provides special operations that permit processes to migrate autonomously.

The languages also differ in their approach to transporting objects other than agents. When a closure or process migrates, it can either carry along all the objects (mutable data) that it references or leave the objects behind and carry along network references to the objects. Java does not address this issue since it permits only program code to migrate. In Obliq, objects remain on the node on which they were created and mobile closures contain network references to these objects; if object migration is desired, it needs to be programmed explicitly by cloning objects remotely and then deleting the originals. In Kali Scheme, objects are copied upon migration; this results in multiple copies of the same objects; data consistency needs to be programmed explicitly if it is desired. In Telescript, objects can either migrate or stay behind when an agent that owns them migrates. However, if other agents hold references to an object that migrates, those references become invalid. Hence, programming care is required to protect against dangling pointers.

In this paper, we adopt a fairly general model of mobile agents. Agent interpreters run on individual networked computers and communicate among themselves using host-to-host communication services. An agent consists of code together with execution state. The state includes a program counter, registers, environment, recursion stack, and store. Agents execute by being interpreted by agent interpreters.

Agents communicate among themselves by message passing. In addition, agents can invoke a special asynchronous "remote apply" operation that applies a closure to arguments on a specified remote interpreter. Remote procedure calls can be implemented with this primitive operation and message passing. Agent migration and cloning can also be implemented with this primitive operation, using first-class continuation values.

3 Two Examples

In this section, we will describe two examples. We believe they are typical of many—though not of all—of the ways that mobile agents can effectively be used. We will try to draw out the most important security issues that they raise, as a concrete illustration of the problems of secure mobile agents.

Competing Airline Carriers. Consider a mobile agent that visits the Web sites of several airlines searching for a flight plan that meets a customer's requirements. We focus on four hosts: a customer host, a travel agency host, and two servers owned by competing airlines, for instance United Airlines and American Airlines, which we assume for the sake of this example do not share a common reservation system. The mobile agent is programmed by a travel agency. A customer dispatches the agent to the United Airlines server where the agent queries the flight database. With the results stored in its environment, the agent then migrates to the American Airline server where again it queries the flight database. The agent compares flight and fare information, decides on a flight plan, migrates to the appropriate airline host, and reserves the desired flights. Finally, the agent returns to the customer with the results.

The customer can expect that the individual airlines will provide true information on flight schedules and fares in an attempt to win her business, just as we assume nowadays that the reservation information the airlines provide over the telephone is accurate, although it is not always complete.

However, the airline servers are in a competitive relation with each other. The airline servers illustrates a crucial principle: *For many of the most natural and important applications of mobile agents, we cannot expect the participants to trust one another.*

There are a number of attacks they may attempt. For instance, the second airline server may be able to corrupt the flight schedule information of the first airline, as stored in the environment of the agent. It could surreptitiously raise its competitor's fares, or it could advance the agent's program counter into the preferred branch of conditional code. As we will argue in Section 4.1, cryptography does not help here either. Thus, the mobile agent cannot decide its flight plan on an airline host since the host has the ability to manipulate the decision. Instead, the agent would have to migrate to a neutral host such as the customer's host or a travel agency host, make its flight plan decision on that host, and then migrate to the selected airline to complete the transaction. This attack illustrates a principle: *An agent's critical decisions should be made on neutral (trusted) hosts.*

A second kind of attack is also possible: the first airline may hoodwink the second airline, for instance when the second airline has a cheaper fare available. The first airline's server surreptitiously increases the number of reservations to be requested, say from two to 100. The agent will then proceed to reserve 100 seats at the second airline's cheap fare. Later, legitimate customers will have to book their tickets on the first airline, as the second believes that its flight is full. This attack suggests a third principle: *Unchanging components of the state should be sealed cryptographically.*

Distributed Intrusion Detection. Consider an intrusion protection system that protects networked computer systems from electronic attacks by collecting audit data, detecting electronic attacks, and responding to suspected attacks. Mobile agents can be used to dynamically alter the data being collected, distribute the computation across the network, and dynamically respond to suspected attacks. The potential benefits of a mobile agent architecture include greater flexibility and improved performance.

In an ongoing project, we are designing a mobile agent architecture where the network is partitioned into one or more network domains. Each domain has a protected computer running an interpreter that is trusted by all agents within that domain. These interpreters trust each other to varying degrees depending on the relationships between the domains. All other interpreters run on untrusted computers that the intrusion protection system is trying to protect; hence these interpreters cannot be trusted.

The agents of this system will require special privileges to collect audit data and respond to attacks. At the same time, the agents will need to be restricted so that they cannot exceed their authority. An important aspect of this example is that the agents will execute on untrusted hosts in a hostile environment. In order to be effective, the system will require strong security controls to protect both the intrusion detection system and the underlying computer infrastructure.

Numerous attacks, both inadvertent and deliberate, are possible. Intruders can terminate or modify the behavior of interpreters. They can inject their own agents and can modify or trick legitimate agents into performing malicious tasks. They can spy on sensitive data stored within agents, within interpreters, and within communications between agents and interpreters.

Consider a data collection agent that is dispatched by a trusted interpreter, migrates to an untrusted machine, collects process information from that host (e.g., by running "`ps`" on a UNIX host), then migrates back to the original interpreter to deposit the collected information. If the network addresses of the two interpreters are stored as state variables of the agent, the second interpreter can switch the two addresses, reset the program counter, and return the agent to the first interpreter. The agent will now collect process information from the first interpreter and return it to the second interpreter, thus providing valuable information to an attacker. This attack illustrates that *a migrating agent can become malicious by virtue of its state getting corrupted.*

Ideally, we would like the interpreters to distinguish between agents of the intrusion detection system and agents of attackers. The interpreters should verify the integrity of agents and should execute legitimate agents correctly. The interpreters should provide agents with appropriate resources but prevent harmful behavior. Agents should be able to communicate privately and restrict access to sensitive code or data that they carry. Agents should execute correctly and completely; that is, agents should migrate correctly to desired hosts, execute correctly on those hosts, and should be recovered in the event of system failure.

4 Security Goals

Security is a fundamental concern for a mobile agent system. Harrison et al. [7] identify security as a "severe concern" and regard it as the primary obstacle to adopting mobile agent systems.

The operation of a mobile agent system will normally be subject to various agreements, whether declared or tacit. These agreements may be violated, accidently or intentionally, by the parties they are intended to serve. A mobile agent system can also be threatened by parties outside of the agreements: they may create rogue agents; they may hijack existing agents; or they may commandeer interpreters.

There are a variety of desirable security goals for a mobile agent system. Most of these concern the interaction between agents and interpreters. The user on behalf of whom an agent operates wants it to be protected—to the extent possible—from malicious or inept interpreters and from the intermediate hosts which are involved in its transmission. Conversely, an interpreter, and the site at which it operates, needs to be protected from malicious or harmful behavior by an agent.

Not all attractive goals can be achieved, however, except in special circumstances. In the case of mobile agents, one of the primary motivations is that they allow a broad range of users access to a broad range of services offered by different—frequently competing—organizations. Thus, in many of the most natural applications, many of the parties do not trust each other. In our opinion, some previous work (for instance [16]) is vitiated by this fact: It assumes a degree of trust among the participants which will not exist in many applications of primary interest.

Nevertheless, the special cases may be of special interest to some organizations. A large organization like the United States Department of Defense might set up a mobile agent system for inter-service use; administrative and technical constraints might ensure that the different parties can trust each other in ways that commercial organizations do not. In this paper, however, we will focus on the more generic case, in which there will be mistrust and attempts to cheat.

To emphasize the consequences of this choice, we will first discuss putative security goals that we believe cannot be achieved in realistic cases. We will then turn to the security services that can already be supported by well-known techniques for security in distributed systems. Finally, we will identify some security goals that we believe can be achieved, but not without novel additions to current distributed security mechanisms.

4.1 What is Impossible

Several apparently desirable security goals appear unachievable in the generic case we are focusing on.

Is an interpreter untampered? There appears to be no reliable way to authenticate an interpreter. For instance, suppose that one wants to determine whether the interpreter running on a particular host has been tampered with, in the sense that its text segment does not match a given executable image identically. In case the host is not running an operating system that one trusts, there appears to be no way to ensure this.[1]

In our context we can assume that many of the hosts will be purchased and maintained by adversaries, or at least competitors. Then, first, the host is unlikely to allow one to log in and inspect the memory of the running executable to do the comparison by hand, so to speak. Second, a utility program running on that host to perform such comparisons on our behalf could itself have been tampered with, leading to a regress. Third, it is infeasible in general to determine, by sending test scripts, whether an interpreter has been tampered with; the tampering has probably been designed to be unobtrusive, and to make a difference only in odd but important circumstances. Testing software is hard enough in a non-adversarial context; bugs may survive lengthy testing even if they were not designed to be hard to find.

Will an interpreter run an agent correctly? Programs are merely a special kind of data, and agents are merely itinerant programs with some additional types of data attached. Because the agent is essentially passive, there is no way to ensure that the interpreter will execute the program in accordance with the intended semantics of the program. Moreover, there is normally no way to check whether an agent has been executed faithfully: If we knew what result it would compute, we would not have needed to send the agent.

It may sometimes be possible to determine heuristically that an interpreter is cheating, by sending agents whose results we believe we can predict ahead of time. However, as we mentioned, clever cheaters are apt to escape detection for a long time.

Will a host run an agent to completion? A host may decide, for reasons of its own, to stop execution of an agent.

Will a host transmit an agent as requested? A host may decide, for reasons of its own, not to transmit an agent that requests to move, or alternatively, to transmit it to the wrong destination. However, with suitable public-key cryptographic support, it is possible to ensure that a user is not tricked into thinking that a particular host was contacted if it was not.

[1] On the other hand, if one does have some assurance about the host hardware and operating system, then one can ensure that a valid version of a program will be running [11, Section 6].

Can an agent's code and data be kept private? Since an agent's code must be executed by a potentially large group of interpreters, it must be readable by all of them. Hence, there is little point in attempting to protect it by encryption. A similar point holds for data carried by the agent that will be needed later in its travels; if an agent will need to consult data in its state at an interpreter that its sender does not trust, then that data cannot be encrypted.

By contrast, data an agent has collected may be encrypted with its sender's public key if the data will not be examined again until the agent returns home. If a host may be trusted to provide true data on a particular subject, then this method may be used to ensure no host visited later will be able to change the results meaningfully.

If a pair of interpreters trust each other at least to a limited extent, then they can choose a session key for communications between themselves [11]. In this case they can offer link security to agents: agents being transferred between those interpreters will be transmitted in encrypted form.

Can an agent carry a key? For similar reasons, an agent cannot carry its own key (or other secrets, such as credit card numbers) in a form that can be used on untrusted interpreters. Someone will peek.[2]

A secret such as a key can be carried in *encrypted* form, but an interpreter must be entrusted with a "master key" if the agent is to be able to use the decrypted secret.

However, it appears undesirable to give an agent an encrypted key even for use on trusted interpreters. It is useless until we authenticate an interpreter and distribute the master key on a secure channel, for instance using the interpreter-to-interpreter encryption mentioned above. What point does it serve then to have the agent carry an encrypted key? It seems simpler and more robust to use the interpreter-to-interpreter encryption itself, so long as the agent has a name that the sender can tag the message with. If the interpreter can be trusted with a master key, then it can surely be trusted to give the name correctly over the secure channel.

[2]For this reason we expect, in the example of the airline reservation system, that the agent will make a *reservation* rather than an actual *purchase*. The purchase itself can be handled more safely by having the sender separately engage in an electronic purchase protocol. Such protocols require the purchaser to be on-line—and to demonstrate possession of a private key—as the transaction occurs, unlike mobile agents, which can be active while their sender is off-line.

For an overview of electronic purchase protocols, see `http://www.ini.cmu.edu/NETBILL/commerce.html`.

Can agent-to-agent communication be kept private? Similar considerations apply to agent-to-agent communication. It seems pointless to give agents keys so that they can have authenticated or secret communication with other agents. That mechanism could work only while the agents are executing on trusted interpreters. And in that case, we can use the simpler and more robust interpreter-to-interpreter secure communication. The sending agent passes data to its interpreter, which sends the data through an encrypted channel to the interpreter executing the receiving agent. The interpreters are then trusted to identify the sender and recipient correctly, and to protect the message by proper encryption.

Can an agent be distinguished from a clone? Many mobile agent languages allow agents to clone themselves. However, the system cannot reliably distinguish the original agent from its clone. This is because agents do not carry keys. Thus, if the code and data of the clone are to be authenticated, they must have the same cryptographic checksum as the original agent, as the private keys of the sender and author are not available to construct new ones. Thus, the code and signed data of the clone must be identical to the original. Thus, to distinguish them at all, we must examine the unsigned portion of their state, and there is no guarantee that these components have not been tampered.

4.2 What is Easy

Some fundamental security goals can be achieved by familiar techniques for distributed security.

Can the author and the sender of an agent be authenticated? The identity of the author of the program contained in an agent can be determined if the author signs the code. Similarly, the sender of an agent may make his identity known by signing the program together with such other components of an agent as will remain fixed through its travels.

Can we check the integrity of an agent's code? Modification of an agent's code can be detected by checking the author's signature.

Can interpreters ensure agent privacy during transmission? Unauthorized parties can be prevented from reading sensitive information held by an agent while it is in transit between two interpreters if the interpreters are willing to encrypt it for transmission.

Authorization: Can interpreters protect themselves against agents? An interpreter (or a remote resource manager) can decide if an agent should have access to a resource by considering the agent's author, program, user, and state. Some of these items may be known to be worthy of a certain degree of trust.

4.3 What is Possible but not Easy

Some security goals cannot be achieved via existing approaches to security for distributed systems. Nevertheless, it appears that they can be achieved by developing special techniques for security in mobile agents. We consider these areas to be the natural context for research in mobile agent security.

We will group the issues into two classes: those which allow an interpreter to evaluate the safety of code that it is to execute, and those which allow an interpreter to evaluate the safety of an agent's state.

Can we use a language in which all programs are safe? One possibility is to develop "safe" languages, in which agents or mobile code have restricted access to operations that affect the environment; Safe-Tcl is an example [1]. In this approach, an incoming, untrusted piece of code is provided with a subset of the language primitive operations; presumably, anything that can be done with these is "safe enough." This approach is reasonable in some contexts, although its flexibility is limited.

Java [13] and Telescript [15] both use aspects of their object oriented programming languages to allow libraries to offer a secure interface to incoming code. The languages are complex, however, and widespread review is only beginning [5]. Undoubtedly piecemeal revisions will be needed, and more importantly, a comprehensive understanding of the semantics of the languages is called for. A good semantics should allow a programmer to draw confident conclusions about what possibilities are allowed by the interface he offers.

Java also offers a *byte-code verifier* [13]. This is intended to check programs at load time. Java code is compiled into an intermediate form called byte-code before it is transmitted. The byte-code verifier is intended to assure an interpreter that a newly arrived piece of byte-code—which may have been compiled by a faulty or malicious compiler—satisfies the same type-correctness properties that a correct compiler would enforce. As far as we know, there has been little independent analysis of its design or implementation.

Can a sender restrict his agents flexibly? In some applications, a sender wants his agent to run with restricted authority in most cases, but with greater authority in certain situations. For instance, in the intrusion detection tool mentioned above, a data-collection agent executing `ps` on an untrusted UNIX system needs only ordinary privilege. However, when it returns to its home interpreter, the agent must request privilege so that it can install the newly gathered information into a protected database. Thus, there must be a mechanism to allow an agent to request different levels of privilege depending on its state (including its program counter).

Can an interpreter ensure that an agent is in a safe state? Because a migrating agent can become malicious if its state is corrupted, as in the case of the intrusion detection `ps` agent, an interpreter may want to execute a procedure to test whether an agent is in a harmful state. However, the test must be application-specific, which suggests that reputable manufacturers of mobile agents may want to provide each one with an appropriate state appraisal function to be used each time an interpreter starts an agent. The code to check the agent's state may be shipped under the same cryptographic signature that protects the rest of the agent's code, so that a malicious intermediary cannot surreptitiously modify the state appraisal function.

Can a sender control which interpreters have authority to execute an agent? If executing an agent involves contacting other hosts, then an interpreter may have to authenticate that it is a legitimate representative of the agent. The sender of an agent may want to control which interpreters will be able to succeed in authenticating themselves in this role.

5 Conclusion

Many of the most important applications of mobile agents will occur in fairly uncontrolled, heterogeneous environments. As a consequence, we cannot expect that the participants will trust each other. Moreover, interpreters may disclose the secrets of visiting agents, and may attempt to manipulate their state.

Existing techniques, intended for distributed systems in general, certainly allow substantial protection within the broad outlines of these constraints. However, substantial investment in mobile agent systems may await further work on new security techniques specifically oriented toward mobile agents. These new techniques, discussed in Section 4.3, focus on two areas. One is programming language support to improve the safety of mobile code. The other is support for tracking the state carried by mobile agents. With advances in these areas, we believe that mobile agents will be an important ingredient in producing secure, flexible distributed systems.

References

[1] N. S. Borenstein. Email with a mind of its own. In *ULPAA '94*, 1994. `ftp://ftp.fv.com/pub/code/other/safe-tcl.tar.gz`.

[2] L. Cardelli. A language with distributed scope. In *Proceedings of the 22nd ACM Symposium on Principles of Programming Languages*, pages 286–298, 1995. `http://www.research.digital.com/SRC/Obliq/Obliq.html`.

[3] H. Cejtin, S. Jagannathan, and R. Kelsey. Higher-order distributed objects. *ACM Transactions on Programming Languages and Systems*, 17(5):704–739, September 1995. `http://www.neci.nj.nec.com:80/PLS/Kali.html`.

[4] D. Chess, B. Grosof, C. Harrison, D. Levine, C. Parris, and G. Tsudik. Itinerant agents for mobile computing. *IEEE Personal Communications Magazine*, 2(5):34–49, October 1995. `http://www.research.ibm.com/massive`.

[5] Drew Dean and Dan S. Wallach. Security flaws in the HotJava browser. Technical Report 95-501, Department of Computer Science, 1995. URL ftp://ftp.cs.princeton.edu/reports/1995/501.ps.Z.

[6] W. Farmer, J. Guttman, and V. Swarup. Security for mobile agents: Authentication and state appraisal. In *To appear in the Proceedings of the European Symposium on Research in Computer Security (ESORICS), Lecture Notes in Computer Science*, September 1996.

[7] C. G. Harrison, D. M. Chess, and A. Kershenbaum. Mobile agents: Are they a good idea? Technical report, IBM Research Report, IBM Research Division, T.J. Watson Research Center, Yorktown Heights, NY, March 1995. `http://www.research.ibm.com/massive`.

[8] C. Haynes and D. Friedman. Embedding continuations in procedural objects. *ACM Transactions on Programming Languages and Systems*, 9:582–598, 1987.

[9] IBM Corporation. Things that go bump in the net. Web page at `http://www.research.ibm.com/massive`, 1995.

[10] IEEE Std 1178-1990. *IEEE Standard for the Scheme Programming Language.* Institute of Electrical and Electronic Engineers, Inc., New York, NY, 1991.

[11] B. Lampson, M. Abadi, M. Burrows, and E. Wobber. Authentication in distributed systems: Theory and practice. *ACM Transactions on Computer Systems*, 10:265–310, November 1992. `http://DEC/SRC/research-reports/abstracts/src-rr-083.html`.

[12] Sun Microsystems. Java: Programming for the internet. Web page available at `http://java.sun.com/`.

[13] Sun Microsystems. HotJava: The security story. Web page available at `http://java.sun.com/doc/overviews.html`, 1995.

[14] J. G. Steiner, C. Neuman, and J. I. Schiller. Kerberos: An authentication service for open network systems. In *Proceedings of the Usenix Winter Conference*, pages 191–202, 1988.

[15] J. Tardo and L. Valente. Mobile agent security and Telescript. In *IEEE CompCon*, 1996. `http://www.cs.umbc.edu/agents/ security.html`.

[16] C. Thirunavukkarasu, T. Finin, and J. Mayfield. Secret agents — a security architecture for KQML. In *CIKM Workshop on Intelligent Information Agents*, Baltimore, December 1995.

[17] T. D. Tock. An extensible framework for authentication and delegation. Master's thesis, University of Illinois at Urbana-Champaign, Urbana, IL, 1994. `ftp://choices.cs.uiuc.edu/Papers/Theses/MS.Authentication.Delegation.ps.Z`.

[18] J. E. White. Telescript technology: Mobile agents. In *General Magic White Paper*, 1996. Will appear as a chapter of the book Software Agents, Jeffrey Bradshaw (ed.), AAAI Press/The MIT Press, Menlo Park, CA.