Security for Mobile Agents: Privileges and State Appraisal Mechanism

Theodore A. Tsiligiridis
Athens University of Agriculture,
Infolab, Department of General Science,
75 Iera Odos, 11855 Athens, Greece.
tsili@aua.gr

Abstract
Mobile agents are autonomous software entities that can halt their execution, transport themselves to another agent-enabled host on the network, and continue their execution on the new host, deciding where to go and what to do along the way. They are goal-oriented, adaptive, can communicate with other agents, and can continue to operate even after the machine that launched them has been removed from the network. In this sense, mobile agent technology presents a distributed computing paradigm, which poses significant challenges and important concerns to the security of the agents that form the widespread deployment of an application, as well as, of the hosts on which these agents execute. Mobile agent security issues include: authentication, identification, secure messaging, certification, resource control, non-repudiation, trusted third parties, and denial of service. Moreover, the mobile agent frameworks must be able to counter new threats as agent hosts must be protected from malicious agents, agents must be protected from malicious hosts, and agents must be protected from malicious agents.
State Appraisal defines a security mechanism for protection of mobile agents. Both the author and owner of an agent produce appraisal functions that become part of an agent's code. They are used to determine what privileges to grant to an agent, based on conditional factors and whether the identified state invariants hold. An agent platform uses the functions to verify the correct state of an incoming agent and to determine what privileges the agent can process during execution. Note that the privileges are issued by a platform based on the results of the appraisal function and the platform's security policy. In this work, the concern is to explore the platform processing environment using the capabilities present in most mobile agent systems, and applying some forms of privilege management certificates. This combination offers the ability for most agent systems to extend the agent platform with a policy engine, and to provide a framework for security policy of an application.

Keywords: *Mobile agent systems, mobile agent security, privileges, state appraisal function.*

1. Introduction

Mobile agents are agents that can physically travel across a network, and perform tasks on machines that provide agent-hosting capability. The goal is to provide a distributed computing infrastructure supporting applications whose components can move between different execution environments. This allows processes to migrate from computer to computer, for processes to split into multiple instances that execute on different machines, and to return to their point of origin. Unlike remote procedure calls, where a process

invokes procedures of a remote host, process migration allows executable code to travel and interact with databases, file systems, information services and other agents.

Generally, the factors need to be addressed before agents migrate across a network are the common execution language, the process persistence, the communication mechanism between agent hosts, and finally, the security to protect agents and agent hosts. Thus, if a process is to migrate from one host to another, then both hosts must share a common execution language. In a homogenous networking environment, it's conceivable that assembly language or machine code could be sent across the network for execution. However, such a system would be extremely limited. A more likely scenario for mobile agency is a heterogeneous environment, where many different system architectures are connected. In this case, an interpreted scripting language or emulation of a system that is capable of executing machine code solves the problem of a common execution language. Persistence is a property showing that the processes ready to migrate to remote machines, they must be capable of saving their execution state, or spawning a new process whose execution state will be saved. Persistence involves converting the object's state (variables, stack, and possibly even the point of execution) and converting it into a data form suitable for transmission over a network. Agents should not have to be responsible for achieving this themselves, and process persistence would likely be built into the mobile agent language or architecture. Communication mechanism between agent hosts must exist to transfer agents across networks. An agent might be transferred using TCP/IP, or by using a higher level of communication such as RMI, SMTP, HTTP etc. Mobile agent architectures may even use a variety of transport mechanisms, giving greater flexibility. An agent's executable code must be transferred, which may consume a large amount of network bandwidth, unless shared code is located at the agent host. Techniques such as shared libraries of code, or caching, may be of benefit. In addition, the persistent state of the agent must be transferred.

Security is critical when executable code is transferred across a network. Malicious or badly written could wreak havoc when unleashed upon an unsuspecting host, and agents themselves need protection against hostile hosts that would seek to dissect or modify them. There is no obvious solution that will solve all the security problems of mobile agents, but precautions can be taken to minimise risk. When an agent leaves for a new host, extreme care must be taken to prevent unauthorised modification or analysis of the agent. Agents may carry with them confidential or sensitive information and logic, which shouldn't be accessible to the agent host. Encryption may be of benefit, but the data and code must be decrypted at some point in time for the agent to execute. Once this occurs, the agent becomes vulnerable, and is at the mercy of the agent host. In a scripting language, the internal logic of the agent is exposed, but even compiled languages can be decompiled with a disturbing degree of success. Other than using trusted hosts, there is little that can be done to protect the agent from snooping eyes.

Threats to the security of mobile agents generally fall into four main classes: disclosure of information, denial of service, corruption of information, and interference or nuisance.

There are a variety of ways to examine these classes of threats as they apply to agent systems. The usual approach is to use the components of an agent system to categorise the threats by identifying 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. This is the case, for example, of executing any code from an unknown source, either downloaded from a network supplied on floppy disk. Mobile agents simply offer a greater opportunity for abuse and misuse, broadening the threats of scale significantly. New threats arising from the mobile agent paradigm are due to the fact that, contrary to the usual situation computer security where the owner of the application and the operator of the computer system are the same, the agent's owner and system's operator are different.

Four threat categories are identified: threats streaming 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 cases on 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 focused primarily 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.

The design and implementation of mechanisms to relocate computations requires a careful assessment of security issues. If these issues are not addressed properly, mobile agent technology cannot be used to implement real-world applications. Much work in mobile agent security has focused on the problem of trust appraisal that is, judging whether mobile agents comply with some security policy. For instance, techniques involving authentication (Microsoft Corporation 1996, Farmer W. et al. 1996a), state appraisal mechanisms (Farmer W. et al. 1996a), and proof-carrying code (Necula G., 1997) have been proposed to appraise trust in the integrity and safety of mobile agents. The problem of appraising trust in servers has been largely unexplored. Much work has also been done on the problem of executing agents securely, the goal being to protect servers and agents from malicious agents. Sandbox techniques (Fritzinger J., Mueller M., 1996, Wahbe R. et al., 1993) protect servers by constraining the run-time behaviour of un-trusted agents. Other techniques protect mutually untrusting agents from each other by enforcing strong separation of their resources. Mobile agents' applications are currently being developed by industry, government, and academia for use in such areas as telecommunications systems, personal digital assistants, information management, on-line auctions, service brokering, contract negotiation, air traffic control, parallel processing, and computer simulation.

2. Computational model

There are many ways to implement mobile agent systems. Interpreted scripting languages or virtual machine-based interpretative language compilers are frequently used for their inherent flexibility in adapting heterogeneous platforms to support agents uniformly.

Depending on the agent system, individual agents may be represented as independent processes or lightweight threads. Similarly, the computational environment for an agent may involve a single host computer or multiple hosts. The privilege management scheme applies to a variety of agent systems, despite these kinds of implementation differences. The approach also provides a means to work independently of or, if available, in conjunction with a Public Key Infrastructure (PKI), including one built in compliance with the X.509 public key certificate framework (ITU-T Recommendation X.509 1997).

Privilege management issues can be discussed using a simple computational model of mobile agents, consisting of only two main components: the agent and the agent platform. Figure 1 illustrates the agent and agent platform components along with other components needed for the privilege management. An agent comprised of the code together with execution state information needed to curry out some computation. Agents execute on servers within the context of global environments provided by the servers. The role of servers may play individual networked computers, servers that run on computers, etc. These servers communicate among themselves using host-to-host communication services The execution state of information includes a program counter, registers, local environment, control stack, and store. 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. When the author and owner each digitally sign an agent, their respective appraisal functions are protected from undetectable modification. One way of looking at this in comparison with attribute certificates is that state appraisal conveys both the policy engine and the prescribed policy internal to agent.

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. Mobility allows agents to move, or hop, among agent platforms. The global environment provides agents with restricted access to services such as communication services and access to computational or data resources of the underline hosts. 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 server. 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.

An agent platform uses the functions to verify the correct state of an incoming agent and to determine what privileges the agent can process during execution. Note that the privileges are issued by a platform based on the results of the appraisal function and the platform's security policy. In this work, the concern is to explore the platform-processing environment using the capabilities present in most mobile agent systems, and applying some forms of privilege management certificates, like the attribute and policy certificates. This

combination offers the ability for most agent systems to extend the agent platform with a policy engine, and to provide a framework for security policy of an application. The approach also needs an overview of shortcomings of agent systems in the way in which they manage the privileges of agents.

3. Protection of an 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 underline 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. More recently developed techniques aimed at mobile code and mobile agent security have for the most part evolved along these traditional lines. In the sequel we will describe the main characteristics of the techniques devised for protecting the agent platform.

Software-based fault isolation

This 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 fault domain. Access to system resources can also be controlled through a unique identifier associated with each domain. The technique, commonly referred to sandboxing, is highly efficient compared with using hardware page tables to maintain separate address spaces for modules, when 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.

Safe code interpretation

Agents 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 (Ousterhout J., 1998). 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. Mobile code systems permit computational entities to be distributed, installed, and executed on remote servers. Such systems have been developed based on numerous programming languages including Java, Telescript, Obliq, Tcl, Scheme, ML, and Python. Note that the two primary advantages provided by mobility are the software distribution across wide-area networks and the adaptability, which enables distributed systems to reconfigure themselves dynamically based on quality of service requirements.

One of the most widely used interpretative languages today is Java. The Java programming language and runtime environment 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. A security manager mediates all accesses to system resources, serving in effect as a reference monitor. Java inherently supports code mobility, dynamic code downloading, digitally signed code, remote method invocation, object serialisation, platform heterogeneity, and other features that make it an ideal foundation for agent development. There are many agent systems based on Java, including CARDAS (Boggavarapu K., Boggavarapu M., 2000), Ajanta (Karnink N., Tripathi A. 2000, Tripathi A. et al., 1999, Karnink N., Tripathi A., 1998), Aglets (Karjoth G. et al. 1997, Lange D.B. et al., 1998), Mole (Young A. and Young M., 1997), and Voyager (ObjectSpace Inc., 1997). However, limitations of Java to account for memory, CPU, and network resources consumed by individual threats (Czajkoowski G., von Eicken T., 1998) and to support threat mobility (Fuggetta A. et al., 1988) have been noted.

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 organisation, mobile agent systems 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 entity. One key is kept private by the entity and the other is made publicity 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 a code, encrypting the result with the private key of the signer forms a digital signature. Because the message digest is unique, and thus bound to 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.

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 organisation, 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 indented 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).

State appraisal

The goal of state appraisal (Farmer et al., 1996a) is to ensure that an agent has not been somehow subverted due to alterations of its state information. This means that the agent has

not become malicious as a consequence of alterations to its state. Checking the integrity of an agent's state is difficult since the state can change during execution and hence cannot be signed by the agent's sender. 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 grand 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.

Path histories

The basic idea behind path histories is to maintain an authenticatable record of the prior platforms visited by an agent, so that 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 determine whether it trusts the previous agent platforms that the agent visited, either by simply reviewing the list of identifies provided or by individually authenticating the signatures of each entry in the path history to confirm identity. While the technique does not prevent a platform from behaving maliciously, it serves as a 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. The technique is also dependent on the ability of a platform to judge correctly whether to trust the visited platforms identified.

4. Privileges and state appraisal function

Requirements for secure mobile agent

Security is a fundamental concept. By building security from the ground-up we gain efficiency by identifying and dealing with potential bottlenecks at the design stage. To

provide an efficient architecture and to ensure security it will be necessary to eliminate unnecessary and/or insecure communication among agents and interpreters. The classification of requirements for secure mobile agents is as follows (Farmer W. et al., 1996b):

- The author and sender of an agent must be authenticated. Authentication is the process of deducing which principal has made a specific request. In a distributed system, authentication is complicated by the fact that a request may originate on a distant host may traverse multiple machines and network channels that are secure in different ways and are not equally trusted.
- The correctness of an agent's code must be checked.
- Interpreters must ensure that agent privacy is maintained during transmission.
- Authentication and authorization: Interpreters must protect themselves against malicious agents by first authenticating that its proposed activities are authorized.
- Agents must be created in a language that supports the development of safe programs. The language used must ensure agent safety, whereas all the programs must guarantee that they have no unbounded loops, violations of array index bounds, etc. This will make attacks and malicious code propagation much harder.
- A sender must have control over an agent's flexibility; e.g., restrict or increase an agent's authorization in particular situations.
- An interpreter must ensure that an agent is in a safe state. Because a migrating agent can become malicious, each agent must be equipped with an appropriate state appraisal function to be used each time an interpreter starts an agent. This will ensure that an agent will perform as required and has not been tampered with a malicious way. Agents' creators will be provided with appropriate static analysis tools that will ensure that the state appraisal function satisfies key safety and security properties.
- A sender must have control over which interpreters have the authority execute an agent.

Security agents protect a system against information operations attacks by implementing key security features such as encryption, authorization, policy enforcement, virus checking, survivability, and intrusion detection. Since security agents have more privileges than other agents, we need higher assurance during development and deployment to ensure the safe and secure behaviour of security agents

Agent privileges and restrictions

Many times an application creating and launching an agent needs to specify the privileges to be given to an agent at a remote host. It may also need to specifically prescribe certain restrictions on its capabilities. Typically an application executes under the authority of a human user, who is designated as owner of the agents launched by that application. The privileges for an agent are defined for the following categories of operations:

- The list of servers that the agent is permitted to visit.
- The number of times the agent is allowed to visit a server. This restriction is important to eliminate the possibility of any replay attacks.
- At each server, the names of the resources that the agent should be allowed to access, and the resource methods that it are permitted to invoke.

- Names of remote hosts with whom the agent be allowed to establish network connections when the agent is resident at a given server.
- Names of the agents who should be permitted to invoke this agent's methods using RMI facility.

An agent's creator to inhibit certain actions by the agent when executing at remote hosts can also specify restrictions. These can be viewed as negative privileges. For example an agent's creator may wish to specify the hosts that the agent should never visit, or the names of the resources/services that should never be allowed to be accessed by the agent when executing at some given server. Sometimes a server may need to forward an agent to another server, which may note be on the agent's original travel plan prescribed by its creator. This kind of situation arises when a server needs to obtain certain services from another server in order for the agent to complete its designated task. In such a case, the second server would perform the requested service under the authorization given by the forwarding server. The forwarding server would need to grand the agent additional privileges.

Conclusions

Currently, protecting agents from malicious hosts is an area of ongoing research. This paper describes the initial steps of a research effort to design and implement security mobile agent systems. This initial work focused on understanding the security mechanisms, particularly those related with those controlling the behaviour of mobile agent system entities through the allocation of privileges. The unique aspect of the architecture is a state appraisal mechanism. When an agent arrives at a new site of execution, it must decide what privileges it will need at that site. After state appraisal has determined which permissions to request, the authorization mechanism on the new execution site may determine which of the requested permissions it is willing to grant. The long term goal for this research is to develop guidelines for the security analysis of mobile agent systems and to determine if existing systems provide the security abstractions and mechanisms needed to develop real-world applications.

References

Boggavarapu K., Boggavarapu M., (2000). Reliable Agents and Security Management System, URL: http://www.lehigh.edu/~kcb2/documents/Reliable_agents_full.pdf .

Czajkoowski G., von Eicken T., (1998). JRes: A resource Accounting Interface for Java, ACM Conference on Object Oriented Languages and Systems (OOPSLA), Vancouver, Canada.

Farmer W., Guttman J., Swarup V., (1996a). Security for mobile agents: Authentication and state appraisal. In: Proceedings of the 4th European Symposium on research in Computer Security (ESORICS '96), pp. 118-130.

Farmer W., Guttman J., Swarup V., (1996b). Security for mobile agents: Issues and requirements. In: National Information Systems Security Conference. National Institute of Standards and Technology.

Fritzinger J. and Mueller M., (1996). Java security. URL: http://java.sun.com/security.

Fuggetta A., Picco G., Vigna G., (1988). Understanding Code mobility, IEEE Transaction on Software engineering, 24(5), pp.342-361
ITU-T Recommendation X.509 (1997). Information Technology – Open Systems Interconnection – The Directory: Authentication Framework.
Karnink N., Tripathi A. (2000). A Security Architecture for mobile Agents in Ajanta. In: Proceedings of the 20th International Conference on Distributed Computing Systems.
Karnink N., Tripathi A. (1998). Design Issues in Mobile Agent Programming Systems. IEEE Concurrency, 6(6), pp: 52-61.
Karjoth G., Lange D.B., Oshima M. (1997). A security model for Aglets, IEEE Internet Computing, pp. 68-77.
Microsoft Corporation (1996). Authenticode. http://www.microsoft.com/intdev/security.
Necula G., (1997). Proof-carrying code. Proceedings of the 24th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Langauges (POPL '97).
ObjectSpace Inc., (1997). ObjectSpace Voyager Core Package Technical Overview, ObjectSpace Inc. URL: http://www.obgectspace.com.
Ousterhout J., (1998). Scripting: Higher-Level programming for the 21st Century, IEEE Computer, pp. 23-30.
Tripathi A., Karnink N., Vora M., Tanvir A., Singh R. (1999). Mobile Agent Programming in Ajanta. In: Proceedings of the 19th International Conference on Distributed Computing Systems.
Wahbe R., Lucco S., Anderson T.E. and Graham S.L. (1993). Efficient software-based fault isolation. Proceedings of the 14th ACM Symposium on Operating Systems Principles, pp.203-216.
Young A. and Young M., (1997). Sliding Encryption: A Cryptographic Tool for Mobile Agents. Proceedings of the 4th International Workshop on Fast Software Encryption, FSE'97.

FIGURES

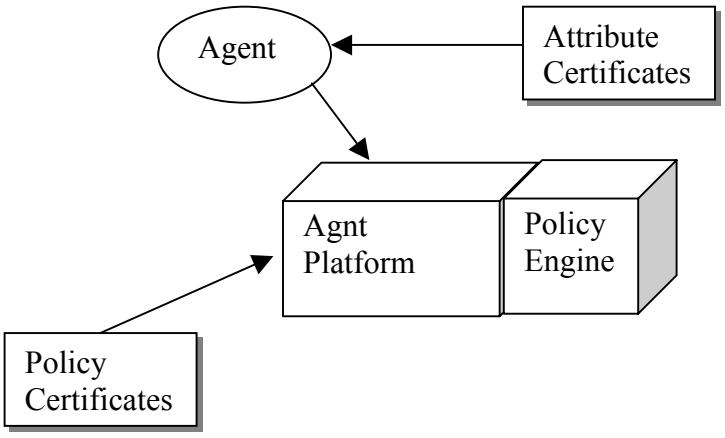

Figure1: Agent System Model