이동 에이전트 시스템의 보안모델 설계를 위한 요구사항 분석

박 진 호* 정 진 욱**
*대덕대학 컴퓨터인터넷정보계열
**성균관대학교 정보통신공학부

요 약

이동에이전트 시스템은 이동에이전트 영역을 지원하기 위한 내부 구조이다. 본 논문에서는 이러한 내부 구조를 설계하면서 만나는 주요 요구사항을 분석하고 설명하고자 한다. 일반적인 이동에이전트 시스템을 설명한 후, 이동에이전트 시스템 설계의 요구사항을 시스템 수준과 프로그래밍 언어 수준으로 구분하여 설명하고자 한다. 이동에이전트의 실행환경 개발에서 주로 요구되는 에이전트의 이동성과 보안성의 제공 등이 시스템 수준의 요구사항이다. 주로 라이브러리 수준에서의 이동에이전트의 프로그래밍을 위해 제공되는 에이전트 프로그래밍 모델과 특징들과 같은 것이 프로그래밍 언어 수준의 요구사항이다. 본 논문에서는 이와 같은 시스템 및 프로그래밍 언어 수준의 요구사항을 명확히 하고 개발자들이 이러한 요구사항을 만족시키기 위한 방법들을 설명하며, 특히 이동에이전트의 보안을 위한 요구사항들에 대하여 분석하고자 한다.

Requirements Analysis in Security Model Design of Mobile Agent Systems

Jin-Ho Park* Jin-Wook Chung**

ABSTRACT

A mobile agent system is an infrastructure that supports the mobile agent paradigm. The main challenges encountered in designing this infrastructure are discussed in this paper. After introducing a generic mobile agent system, we discuss the design issues at two levels. System level issues like the provision of agent mobility and security, are mainly encountered in developing the runtime environments for agent execution. Language level issues, such as agent programming models and primitives, arise in providing support for mobile agent programming, mainly at the library level. This paper identifies such system and language-level issues, and illustrates the different ways developers are addressing them. We outline the specific challenges addressed by this dissertation, primarily in the area of mobile agent security.

1. Introduction

Interest in network centric programming and applications has surged in recent months due to various factors, such as the exponential growth of the Internet user base, and the widespread adoption of the Worldwide Web as a platform for information dissemination, electronic commerce and entertainment. The increasing popularity of Java, a language that facilitates mobility of code across a network, is another driving force. The use of Internet-based technologies in private networks (thus creating intranets and extranets) has further fuelled the demand for network-based applications. In response to this, new techniques, languages and paradigms have evolved which facilitate the creation of such applications. Perhaps the most promising among the new paradigms is the use of mobile agents. In this chapter, we describe the mobile agent paradigm and review its development from a historical perspective. We consider its characteristics and how they can be exploited by certain classes of applications, and consider the drawbacks of using mobile agents on open networks such as the Internet.

In a broad sense, a software agent is any program that acts on behalf of a (human) user, just as different types of agents (e.g., travel agents, insurance agents, secretaries) represent other people in day-to-day transactions in the real world. A mobile agent then is a program which represents a user in a computer network, and is capable of migrating autonomously (under its own control) from node to node in the network, to perform some computation on behalf of the user. Its tasks are determined by the agent application, and can range from online shopping to real-time device control to distributed scientific computing. Applications can inject mobile agents into a network, allowing them to roam the network either on a predetermined path, or one that the agents themselves determine based on dynamically gathered information. Having accomplished their goals, the agents may either terminate or return to their "home site" in order to report their results to the user.

Harrison[1] identified several advantages of the mobile agent paradigm, in comparison with RPC and message-passing. These stem from the characteristics of the paradigm which allow it to:

- reduce network usage
- increase asynchrony between clients and servers
- add client-specified functionality to servers
- dynamically update server interfaces
- introduce concurrency

Mobile agents can facilitate the programming of several types of network applications. However, the mobile agent paradigm also adds significant problems, primarily in the area of security and robustness. The use of mobile agents requires that each cooperating host in the distributed system provide a facility for executing them. These hosts are exposed to the risk of

system penetration by malicious agents, similar to viruses and Trojan horses. Unless some countermeasures are taken, such agents can potentially leak or destroy sensitive data and disrupt the normal functioning of the host. Malicious (or just buggy) agents can cause inordinate consumption of resources such as CPU time, disk space, network connections or even screen space, thereby denying their use to other agents and legitimate users of the host. Security mechanisms are thus necessary to safeguard hosts' resources from the agents executing on them. Similarly, agents themselves may need to be protected from the hosts they visit. An agent is likely to carry as part of its state, sensitive information about the user it represents - e.g. credit card numbers, personal preferences to customize an intelligent search, or electronic cash to be used for purchasing goods at merchants' servers. Such data must not be revealed to unauthorized hosts or agents, nor must they be allowed to modify it arbitrarily. Thus, security is usually the major concern in the design of mobile agent systems.

As larger and more complex systems of roving agents are deployed, the problem of managing a user's agents becomes more pronounced. Scalable mechanisms are needed for naming and locating agents, even as they migrate in the course of their tasks. Application programmers need reliable control primitives for starting, stopping and issuing application-specific commands to these agents. The agent system itself must incorporate robustness and fault tolerance mechanisms to allow such applications to operate over unreliable networks.

2. Reference Model of Mobile Agent System

We introduce a reference model for a mobile agent system (along with the associated terminology), which will serve as a framework for the discussion that follows. Figure 1. shows a simple model of a generic mobile agent system, and an example of the interactions involved. In the figure, Host-A and Host-B are two machines connected by a network. In order to participate in the mobile agent system, each of them runs an agent server process, which is a protected agent execution environment. These processes are responsible for hosting and executing any agents that arrive over the network, and for providing primitive operations to agent programmers. Examples of such operations include those that allow agents to migrate, communicate, access host resources, etc. Agent servers can communicate with each other using a server-server protocol. A logical network of agent servers in effect implements the mobile agent system.

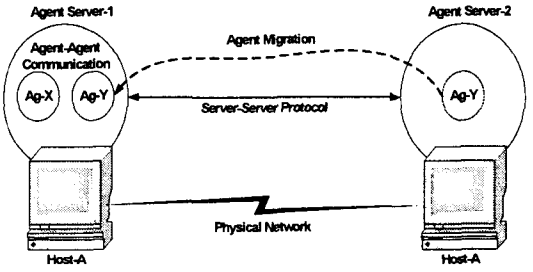

Figure 1. A mobile agent system

In the figure, the agent server on Host-A is currently hosting an agent, labelled Ag-X. Agent Ag-Y, which was running on Host-B, decided to migrate to Host-A, possibly because it needs to communicate with Ag-X, or because it requires some resource which is available on Host-A. In response to its request for migration, Agent Server-2 contacts Agent Server-1 and transmits Ag-Y across the network. Agent Server-1 accepts the incoming agent code and assigns a thread to execute it. Thus, Ag-Y resumes its execution at Host-A. It can now interact locally with Ag-X and/or Agent Server-1. In general, agents can migrate between agent servers and execute within the environments provided by these servers. A single network host may have more than one agent server running, each of which may have multiple agents executing simultaneously. Agent servers can be specialized to provide application-specific services. For example, in an electronic marketplace application, each vendor runs an agent server that provides a shop-front interface to customers' agents. The shop-front includes product descriptions, price lists, etc. and mechanisms for agents to look up such catalogs and order products.

3. Requirements of System Level Design

Agents can execute on many different hosts during their lifetimes. In general, we cannot assume that these hosts have identical architectures or even that they run the same operating system. Thus, agents must be programmed in a language that is machine-independent and widely available. Many useful agent applications will require Internet-wide access to resources. Users will need to dispatch agents from their laptops, irrespective of their physical location. These agents may utilize services or repositories of data anywhere on the network. Hence, the mechanisms used in the agent infrastructure itself should scale up to wide-area networks. Portability and scalability are therefore, common demands made of the agent system. We now discuss several other system-level design issues.

3.1 Agent mobility

The primary identifying characteristic of mobile agents is their ability to autonomously migrate from host to host. Thus, support for agent mobility is a fundamental requirement of the agent infrastructure. An agent can request its host server to transport it to some remote destination. The agent server must then deactivate the agent, capture its state, and transmit it to the server at the remote host. The destination server must restore the agent state and reactivate it at the remote host, thus completing the migration.

The state of an agent includes all its data, as well as the execution state of its thread, also referred to as its thread-level context. At the lowest level, this is represented by its execution context and call stack. If this can be captured and transmitted along with the agent, the destination server can reactivate the thread at precisely the point where it

requested the migration. This can be useful for transparent load-balancing, since it allows the system to migrate processes at any time in order to equalize the load on different servers. It may also be useful in some types of fault-tolerant programs, allowing checkpoint-restart schemes for recovering from crashes. An alternative is to capture execution state at a higher level, in terms of application-defined agent data. The agent code can then direct the control ow appropriately when the state is restored at the destination. However, this only captures execution state at a coarse granularity (e.g. function-level), in contrast to the machine instruction-level state provided by the thread context.

Most current agent systems execute agents using commonly available virtual machines or language environments, which do not usually support thread-level state capture1. The agent system developer could modify the virtual machines for this purpose, but this renders the system incompatible with standard installations of those virtual machines. Since mobile agents are autonomous, migration only occurs under explicit programmer control, and thus state capture at arbitrary points is usually unnecessary. Most current systems therefore rely on coarse-grained execution state capture to maintain portability.

Another issue in implementing agent mobility is the transfer of agent code. One possibility is for the agent to carry all its code as it migrates. This allows the agent to run on any server which can execute the code. Some systems do not transfer any code at all, and require that the agent's code be pre-installed on the destination server. This is advantageous from a security perspective, since no foreign code is allowed to execute. However, it suffers from poor flexibility and limits its use to closed, local networks. In a third approach, the agent does not carry any code but contains a reference to its code base - a server that provides its code upon request. During the agent's execution, if it needs to use some code that is not already installed on its current server, the server can contact the code base and download the required code. This is sometimes referred to as code-on-demand.

3.2 Naming and name resolution

Various entities in the system, such as agents, agent servers, resources, and users need to be assigned names which can uniquely identify them. An agent should be uniquely named, so that its owner can communicate with or control it while it travels on its itinerary. For example, a user may need to contact his/her shopper agent to update some preferences it is carrying. Agent servers need names so that an agent can specify its desired destination when it migrates. Some namespaces may be common to different entities - e.g., agents and agent servers may share a namespace. This allows agents to uniformly request either migration to a particular server or co-location with another agent with which it needs to communicate.

Next, the system must provide a mechanism to find the current location of an entity, given its name. This process is called

name resolution. The names assigned to entities may be location dependent, which allows easier implementation of name resolution.

3.3 Security

The introduction of mobile code in a network raises several security issues. In a completely closed local area network - contained entirely within one organization - it is possible to trust all machines and the software installed on them. Users may be willing to allow arbitrary agent programs to execute on their machines, and their agents to execute on arbitrary machines. However in an open network such as the Internet, it is entirely possible that the agent and server belong to different administrative domains. In such cases, they will have much lower levels of mutual trust. Several types of security problems may arise:

- Servers are exposed to the risk of system penetration by malicious agents, which may leak sensitive information.
- Sensitive data contained within an agent dispatched by a user may be compromised, due to eavesdropping on insecure networks, or if the agent executes on a malicious server.
- The agent's code, control ow and results could be altered by servers for malicious purposes.
- Agents may mount "denial of service" attacks on servers, whereby they hog server resources and prevent other agents from progressing.

The mobile agent system must provide several types of security mechanisms for detecting and foiling such attacks. These include privacy mechanisms (to protect secret data and code), authentication mechanisms (to establish the identities of communicating parties) and authorization mechanisms (to provide agents with controlled access to server resources).

4. Requirements of Language Level Design

Programming language support for mobile agents can be discussed under two distinct heads - programming languages/models, and primitives for agent programming.

4.1 Agent programming languages and models

Since an agent may execute on heterogeneous machines with varying operating system environments, the portability of agent code is a prime requirement. Therefore, most agent systems are based on interpreted programming languages[2], which provide portable virtual machines for executing agent code. Another important criterion in selecting an agent language is safety. Languages that support type checking, encapsulation, and restricted memory access are particularly suitable for implementing protected server environments for executing agents.

Several systems use scripting languages such as Tcl, Python, and Perl for coding

agents. These are typically high-level, macro-like languages often used to glue together operating system-supplied utilities. They are relatively simple to learn and use, and allow rapid prototyping for small to moderate-sized agent programs. They have mature interpreter environments which permit efficient, high-level access to local resources and operating system facilities. However, script programs often suffer from poor modularization, encapsulation, and performance. It is also di_cult to create and maintain large applications using script languages. Some agent systems therefore use object-oriented languages such as Java, Telescript or Obliq [2]. Agents are defined as objects which encapsulate their state as well as code, and the system provides support for object migration in the network. Such systems offer the natural advantages of object-orientation in building agent-based applications. Complex agent programs are easier to write and maintain using object-oriented languages. Some systems have also used interpreted versions of traditional procedural languages like C, for agent programming [3].

Mobile agent systems can differ significantly in the programming model used for coding agents. In some cases, the agent program is merely a script, often with little or no need for control ow. In others, the script language (e.g. Python) borrows features from object-oriented programming and provides extensive support for procedural ow control. Some systems model the agent-based application as a set of distributed interacting objects acting as agents, each having its own thread of control and thus able to autonomously migrate across the network. Others use a callback-based programming model in which the system signals certain events at different times in the agent's life-cycle. The agent is then programmed as a reactive entity using a set of event-handling procedures.

4.2 Programming primitives

In this section, we identify the primitive language-level operations required by programmers implementing agent-based applications. We also illustrate the alternatives supported by the mobile agent systems we surveyed. Agent programming primitives can be categorized into:

- Basic agent management : agent creation, dispatching, cloning and migration.
- Agent-to-agent communication and synchronization.
- Agent monitoring and control: status queries, recall and termination of agents.
- Fault tolerance: check pointing, exception handling, audit trails,. . .
- Agent protection and security: encryption, signing, data sealing,. . .

5. Mobile Agent Security Issues

This section defines the mobile agent security issues addressed by this dissertation.

The major challenges were as follows:

- Designing secure agent transfer and communications protocols
- Protection of host resources
- Protection of agent code and state
- Secure control of remote agents

5.1 Secure communication and agent transfer

Messages sent across an open network like the Internet are inherently insecure. As a mobile agent traverses the network, its code and data are vulnerable to various types of security threats. We consider the following types of attacks on communication links, that the system needs to protect against[4]:

(1) Passive attacks

In passive attacks, the adversary does not interfere with the message traffic, but only attempts to extract useful information from it. The simplest such attack is eavesdropping, which can result in the leakage of sensitive information stored in the message (agent) being transmitted. Even if the adversary is unable to decipher the message contents (because of encryption, for example), useful information may be gleaned from the sizes and frequency of messages exchanged, or merely the fact that two principals are in communication. This type of passive attack is usually called traffic analysis in the security literature. To counter passive attacks, a confidentiality (i.e. privacy) mechanism is therefore necessary.

(2) Active attacks

In the case of open networks like the Internet, we must assume a very general threat model in which the adversary can arbitrarily intercept and modify network-level messages, or even delete them altogether and insert forged ones. These are termed as active attacks, since they involve active interference by the adversary. Another type of attack in this category involves impersonation. The adversary impersonates one of the legitimate principals in the system and can attempt to intercept messages intended for that principal. Active attacks require greater sophistication on the part of the adversary, but can also be more dangerous than passive attacks. While we cannot always prevent all such attacks, the damage caused by them can be minimized if the communications link provides assurances of data integrity (i.e. data is either delivered unmodified or a ag is raised signalling that it has been tampered with) and authentication (i.e. the source and destination of the message is unambiguously identified).

Passive attacks are difficult to detect, but can usually be protected against using cryptographic mechanisms[4]. In contrast, active attacks are relatively easy to detect cryptographically, but given our general threat model, they are di_cult to prevent altogether. In an agent system, the server-server protocol messages often contain sensitive data, such as agent code (which may be proprietary and therefore needs to be kept secret) and data (which, as illustrated above, needs to be protected as well). The agent

servers must also be provided with mechanisms for detecting tampering and impersonation. In other words, confidentiality, integrity and authentication mechanisms must be an integral part of the secure agent transfer protocol. The mobile agent paradigm itself imposes some additional constraints on the possible solutions to these security issues. For example, we cannot use certain cryptographic techniques because agents cannot carry secrets with them, for fear of exposure to malicious servers. Also, since agents usually operate autonomously, there is minimal interaction with their human owners. Thus, protocols that require interaction with the user are often inappropriate in mobile agent systems.

5.2 Protection of host resources

A host participating in a mobile agent system runs an agent server process. The agent server in turn allows the execution of agent programs. Treating these programs on par with local (trusted) software can expose the host to various types of attacks, launched by malicious agents. These attacks can be categorized as under:

- pilfering of sensitive information
- damage to host resources
- denial of service to other agents
- annoyance attacks

We now illustrate these with examples. A malicious agent may visit a server and proceed to open files containing, say, company secrets or financial data. It can transmit this information back to its owner, who can use it to gain a competitive advantage. This is an example of the pilfering of information by agents. An anti-social agent could damage its host's resources by simply deleting files or erasing the hard disk, much like various strains of viruses and Trojan horses. It might also attempt to mount a denial of service attack - by using up the server's resources (such as disk space, network ports or file handles), it can effectively prevent the server from doing business with other agents. Other types of resources are also vulnerable - e.g., the agent can open up hundreds of windows on the server's console, rendering it unusable. It can make the computer beep repeatedly. While these annoyance attacks may not cause tangible damage to the host, they nevertheless have to be prevented.

It is clear therefore that the various resources of the host system need to be protected from malicious agents. At the same time, legitimate agents must be given access to the resources they need. In this context, the primary system-level problems that need to be addressed in any mobile agent architecture are:

- Binding of agents to the local environment
- Authorization
- Enforcement of access controls

The mobile agent system must provide mechanisms to agent servers for specifying restricted access rights for agents - this is usually termed as authorization. The rights assigned usually depend on the agent's

identity (implying that a secure authentication facility is necessary), and are determined by consulting a pre-defined security policy. In addition, we also need mechanisms for enforcing the specified rights - this is the problem of access control. The underlying system-level problem is that of providing a safe binding between the visiting agent code and the local environment - enabling the agent to access the resources it needs (in the ways it is authorized to), but ensuring at the same time that it cannot breach system security by accessing resources it is not authorized to use.

5.3 Protection of agents

When an agent executes on a host's agent server, it is in e_ect completely exposed to that host. The act of migrating to a server implies a certain level of trust in that server and its host. If the server happens to be malicious, it can affect the agent in many different ways:

- It can simply destroy the agent and thus impede the functioning of its parent application.
- It can steal useful information stored in the agent, such as intermediate results gathered by the agent during its travels.
- It can modify the data carried by the agent, for example changing the price quoted by a competitor in a shopping mall, to fool the parent application into favouring the malicious server.
- It can attempt to alter the agent's code and have it perform malicious actions when it returns to its home site. This is especially dangerous, since the home site could treat its own agents as trusted entities, and possibly allow them to bypass access controls to its own resources.

An agent server must of necessity have access to the agent's code and state in order to execute it. Parts of state in fact must usually change, in order to store the results of computations or queries. Thus it is not possible to provide a general guarantee that the agent will not be maliciously modified[5]. However, the parent application must have some mechanism for detecting such modifications. If it determines that the agent has been "attacked", it can take appropriate measures, such as executing it in a restricted environment (with stricter access controls than it would use otherwise), or even discarding it altogether.

When an agent is dispatched, it has an initial itinerary of hosts to visit. Different parts of the agent may be intended for different hosts, and some parts may need to be kept secret until the agent arrives at the intended host. Since these hosts are usually not trusted equally, agent applications need a mechanism for selectively hiding and exposing parts of the agent's state and code to the different agent servers it visits. Also, a secure, verifiable audit trail which records the actual path followed by the agent can be a useful mechanism. This allows the application to ensure that the agent followed the intended itinerary.

5.4 Secure control of remote agents

A mobile agent application may dispatch large numbers of agents to remote sites. It may be necessary to periodically monitor their progress and issue control commands to them. The agent infrastructure must therefore provide some means for the application to query the status of its agents. The application may decide to recall its agents back to their home site, or terminate them midway through their tasks if appropriate. Agent servers must provide remotely invocable primitive operations for this purpose. However, these operations are liable to be misused by malicious users. Therefore, only certain authorized entities must be allowed to invoke them. Thus, authentication of the caller is imperative, and the server must establish and enforce some rules about which entities can, for example, terminate an agent.

6. Conclusions

The preceding sections identified the security issues inherent in mobile agent systems. These issues translate into a set of security mechanisms that the system needs to provide, which can be summarized as follows:

(1) Privacy and integrity of the agent

The system must provide mechanisms for secure communications, and secure transfer of agent code and state as it migrates around an insecure network. Tampering of the agent should be detectable.

(2) Authentication of entities in the system

The entities participating in a mobile agent application, such as servers and agents, must be unambiguously identified.

(3) Authorization and access control

Servers must be provided with a mechanism for protecting their resources, by specifying their access control policies and enforcing them.

In this paper, we identified the major research issues inherent in the development of mobile agent programming systems. These include system-level issues such as agent mobility, global naming and security, as well as language-level support for agent programming, in the form of programming languages, models and primitives. We identified the variety of approaches that system designers have taken to address these issues. The specific research challenges related to security that this dissertation addresses include the design and implementation of a research infrastructure for mobile agent programming, the design of secure agent transfer and communication protocols, the protection of host resources from malicious agents, the protection of agents from tampering by other agents or their hosts, etc.

Reference

[1] Colin G. Harrison, David M. Chess, and

Aaron Kershenbaum. Mobile Agents : Are they a good idea ? Technical report, IBM Research Division, T. J. Watson Research Center, March 1995.

[2] Tommy Thorn. Programming Languages for Mobile Code. ACM Computing Surveys, 29(3) : 213~239, September 1997.

[3] Lubomir F. Bic, Munehiro Fukuda, and Michael B. Dillencourt. Distributed Computing using Autonomous Objects. IEEE Computer, pages 55~61, August 1996.

[4] Warwick Ford. Computer Communications Security - Principles, Standard Protocols and Techniques. Prentice Hall, 1994.

[5] William M. Farmer, Joshua D. Guttman, and Vipin Swarup. Security for Mobile Agents: Issues and Requirements. In Proceedings of the 19th National Information Systems Security Conference, pages 591~597, October 1996.

박 진 호

1995년 대전대학교 전자계산학과(공학사)

1997년 대전대학교 대학원 컴퓨터공학과(공학석사)

1997년 ~ 현재 성균관대학교 대학원 전기전자 및 컴퓨터공학부(박사수료)

2000년 ~ 2002 송호대학 정보산업계열 전임강사

2002년 ~ 현재 대덕대학 인터넷정보기술계열 전임강사

정 진 욱

1979년 성균관 대학교 대학원(공학석사)

1991년 서울 대학교 대학원 계산통계학과(이학박사)

1973-1985 한국과학기술연구소 실장

현재 성균관 대학교 전기전자 및 컴퓨터 공학부 교수