Authentication in mobile-agent system: D'Agents

Haiying Tan

Department of Computer Science

University of Auckland

htan048@ec.auckland.ac.nz

Abstract. D'Agents is a mutiple-language, mobile-agent system. We address the authentication services in this system. D'Agents uses the external encryption tool PGP, which in turn relies on RSA for authentication and encryption. These tools allow the D'Agents server to verify the identity of an incoming agent and the identity of the sending machine. Both agents and messages can be encrypted to avoid interception, and digitally signed to reliably identify their owner. Based on the authentication information, two kinds of agents are distinguished by D'Agents: owned and anonymous. We also compare the authentication services of D'Agents with similar services available in other mobile-agent systems: Concordia, Grasshopper and SOMA.

1. Introduction

Mobile agent is a program, which represents a user in a heterogeneous network[1], moves autonomously from machine to machine, and functions on behalf of the user. With the advent of the web, the potential for mobile agent systems is exploding. However the mobile agent systems have problems as well as promise. Security threat is one of the challenges preventing mobile agent systems from being more widely deployed, due to the autonomous behavior of the mobile agents and the heterogeneous network. The security issues in mobile agent systems can be classified into two broad areas: host security (protecting the host platform from a malicious agent) and code security (protecting the mobile agent from a malicious host platform). This paper's main object is authentication, one of the tasks involved in host security.

D'Agents, formerly named Agent Tcl, is a mobile-agent system developed in Dartmouth College[1]. As one of the earliest mobile agent systems, it was intended to address the weaknesses of existing mobile agent systems, such as insufficient security mechanisms. The architecture of D'Agents is based on the server model of Telescript,

and it supports agents written in Tcl, Java and Scheme, as well as stationary agents written in C and C++[2].

The architecture of D'Agents consists of four levels shown in Fig. 1[1]. The lowest level contains each supported transport mechanism, the next level is the main component in D'Agents, a server that runs on each machine, which manages local and incoming agents. The interpreter level provides the execution environments for each supported language. The last level is the agent level that contains the agents themselves. The agents execute in the interpreters and use the facilities provided by the server to migrate from machine to machine and to communicate with other agents[1].

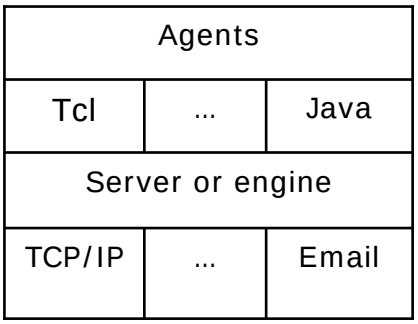

Fig. 1 D'Agents architecture

2. D'Agents security issue

Security in D'Agents is provided in various capacities internally or externally. Its internal security mechanisms use security policies with resource management and its external security mechanism is based on PGP. The agents can decide if external security is required or not. D'Agents handles host security using public key cryptography for authentication and secure execution environment for authorization. It has three components in its security architecture (shown in Fig 2): encryption subsystem, a language-dependent enforcement module, and a language-independent policy module[1].

When an incoming agent arrives, the server of the receiving machine verifies the agent's digital signature, and then either accepts or rejects the agent after checking against the server's current access list. If the server accepts the agent, it records the identity of the agent's owner, starts up an execution environment for the agent, and resumes agent execution[1]. When an agent requests access to a resource, the enforcement module forwards the request to the appropriate resource manager. The resource manager, which is just a stationary agent, implements a security policy that determines whether the access request should be approved or denied. The security

[1] *All the figures in this paper were originally from Reference 1, I made some modifications to all of them*

module then enforces the decision. This approach provides a clean separation between security policy and mechanism, with the same resource managers making security decisions for all agents, regardless of their implementation language[10].

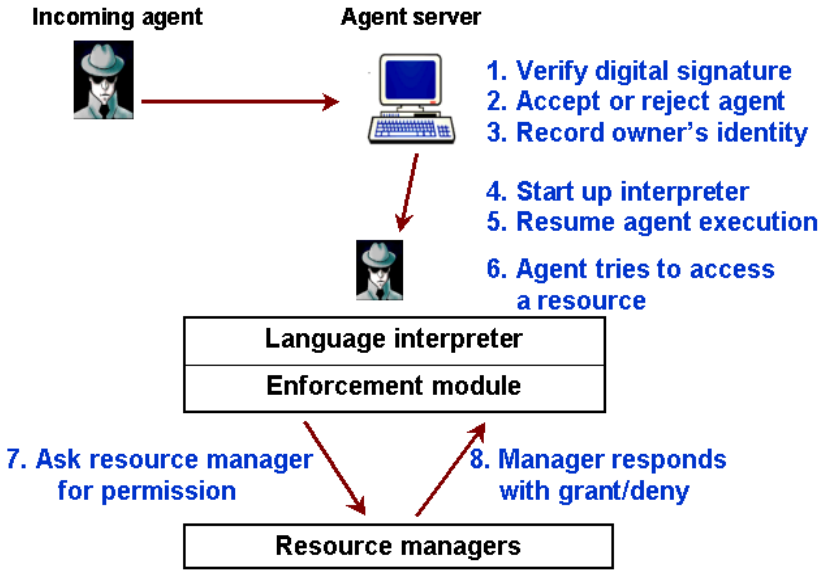

Fig. 2 The components of the D'Agents security architecture

3. D'Agents authentication schemes

"Authentication is the process of deducing which principal has made a specific request"[3]. The crucial security goal here in D'Agents is for the server to verify that the incoming agent is a legitimate representative of the agent. This encompasses a variety of security mechanisms including encryption, public key infrastructure (PKI), and support for executing signed code. PKI serves as a foundation for mobile agent security services and makes authentication, non-repudiation, and encryption readily available to agent developers and users. Most agent systems either use Pretty Good Privacy (PGP) or the Secure Socket Layer (SSL) protocols for authentication. D'Agents employs PGP to realize key distribution and encryption functionalities, which is based on RSA for authentication and IDEA for encryption.

3.1 Authentication terminology

Owned agent & anonymous agent D'Agents server distinguishes between an owned agent whose owner could be authenticated and is on the server's list of

authorized users, and an anonymous agent whose owner could not be authenticated or the server's list of authorized users doesn't contain this owner[1]. Each server can decide whether to allow anonymous agents to execute in a restricted environment or terminate immediately. Here we assume that only those machines in the server's list of authorized users are trustworthy.

IDEA & RSA Single-key encryption as IDEA (International Data Encryption Algorithm) can only be used as part of a security solution, as messages cannot be exchanged securely between two hosts, which never met before. Any eavesdropper might be able to figure out the "secret" key. But once the secure key got exchanged securely it runs much faster in practice than the implementations of public key algorithms[4].

RSA is one of the widely used public-key algorithms, which uses asymmetric cryptography. It consists a pair of keys. One key is used to encrypt the message (public key) and the other is used to decrypt it (private key). As their name suggests, the private key is kept secret and the public key is available to the public.

Digital signature By using a public-key cryptography entity, message can be sent securely, but the question remains: how can the receiving machine make sure that the agent is really from the sending machine, and not an impersonator? Digital signature can solve this problem. It serves as a means of confirming the authenticity of an agent. Typically the code signer is either the creator of the agent, or the user of the agent. Digital signatures benefit greatly from the availability of a public key Infrastructure.

For example, if machine A wants to send an agent to machine B, the state image is signed with A's private key, and encrypted with B's public key, and sent to machine B, when the agent arrives at B, B uses its private key to decrypt the state image, and uses A's public key to decrypt the result. If B can successfully finish the two steps then B can ensure that the agent is from A, because only A can use A's private key to sign a message. Here, we assume B knows the correct public key of A.

Pretty Good Privacy (PGP) PGP is based on a referral model where referral means that the certificate depends on the integrity of a chain of authenticators. The authenticators are the users themselves. The users and their keys are referred from one user to the other, forming an authentication ring[4]. There is no central control on the certificates, so their maintenance is also performed by the users themselves. This is

due to the fact that PGP is an Internet phenomenon, developed largely by one person, Phil Zimmermann[4]. PGP handles the tasks that an agent realm would require.

3.2 Authentication process

Because RSA is used in D'Agents, a package of keys has to be created for each user (owner). This package contains both the public key and the private key of that user, and the public keys of other known users. Additionally a package for the server has to be created which includes its private and public key and the public keys from all know servers.

PGP allows the D'Agents server to verify the identity of an incoming agent and the identity of the sending machine. Additionally, it allows each agent to verify the identity of those agents with which it is communicating by examining their security vectors. The security vector specifies the owner of the sending agent, whether the owner could be authenticated, the sending machine, whether the sending machine could be authenticated, whether the message was encrypted, and whether the sending agent is on the same machine[1]. The recipient agent, which might be controlling access to some resource such as database, can employ its own security decisions based on this security vector. An agent can choose whether to use encryption and signatures when it migrates or sends a message to another agent. The following cases describe the process if it uses both. D'Agents allows the authentication of both agents and platforms by the use of digital signatures. Based on this information, the agent platform and the agent can made decisions about the request of a foreign agent[12].

3.2.1 One-hop Authenticator

When an agent registers with its home server, the registration request is digitally signed with the owner's private key (authentication), optionally encrypted with the destination server's public key (confidentiality), and sent to the server. The server verifies the digital signature, checks its list of authorized users and then accepts or rejects the request(see Fig. 3)[1].

When an agent migrates for the first time, the state image is digitally signed with the owner's private key, optionally encrypted with the destination server's public key, and sent to the destination server. The server verifies the digital signature, checks whether the owner is allowed to send agents to its machine, either accepts or rejects

the incoming agent(see Fig. 3). Of course, once the agent has migrated, the owner's private key is no longer available[1].

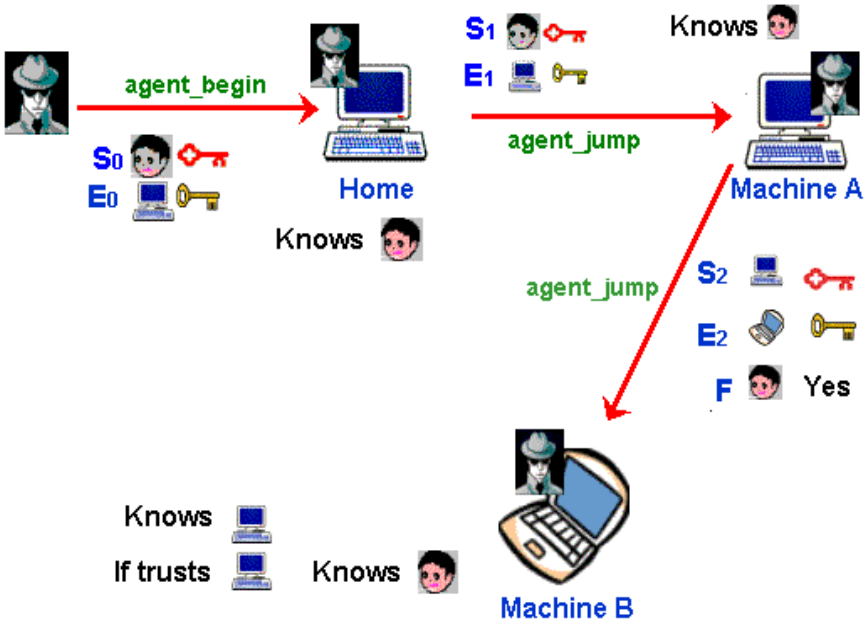

Fig. 3 One-hop and multi-hop authenticators

3.2.2 Multi-hop Authenticator

In multi-hop systems, there is a need to establish trust in the next hop. The agent needs to trust its current host to securely transmit it to next hop and be signed by this host. D'Agents resolve multi-hop authentication by maintaining that if the sender is not the owner of the agent, then it must be able to authenticate the owner before sending it to a third server. So a server is able to authenticate an agent if: (1) the agent was signed by the owner and the destination server is able to authenticate the owner, or (2) the agent was signed by the sender who is not the owner, but the sender was able to authenticate the server[5].

For all subsequent migrations after the first migration, the agent is digitally signed with the private key of the sending machine. If the sending machine was able to authenticate the owner itself, the destination machine considers the owner authenticated and gives the agent the full set of resource limits for that owner. If the destination machine does not trust the sending machine, or the sending machine could not authenticate the owner, the destination machine considers the agent anonymous (see Fig. 3)[1].

3.2.3 Authentication of newly created Agents

When a new child is created on a different machine, the same strategy is used as with the case sending message to an agent on a different machine. The child agent is signed with the owner's private key if the agent is still on its home machine, and with the sending machine's key if the agent has already migrated. The recipient will believe the owner's identity if it trusts the sending server(see Fig. 4). When a new agent is created on the same machine, neither encryption nor digital signatures are required. The new agent inherits the security vector of its parent[1].

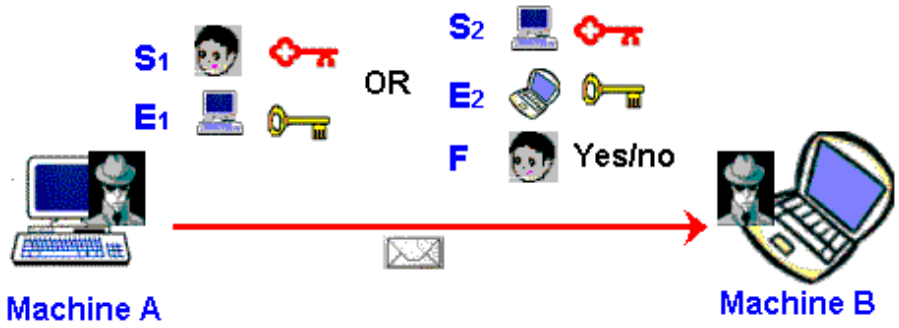

Fig. 4 Authentication of newly created agents

3.3 Weaknesses

If an agent migrates among a set of mutually trusting machines, each machine will be able to (directly or indirectly) authenticate the owner, and will give the agent the full set of access permissions for that owner. Once the agent leaves this set of machines, however, it becomes anonymous, and remains anonymous even when it comes back, since the non-trusted machines might have modified the agent in a malicious way. This leads to the system's most serious multi-hop problem. Under this circumstance, an application that needs the full access rights of its owner to finish its task cannot just send out a single agent to migrates through the machines, instead it must send an agent to the first machine, wait for the results, send a new agent to the second machine, and so on. Obviously network traffic is inevitable, which mobile agent system are meant to avoid. If we can detect any malicious modification to the agent, this problem can be solved.

There are some other problems. First, all public keys must be known in advance, as D'Agents doesn't include an automatic distribution mechanism for the public

keys[1]. A modest-key distribution or certification mechanism must be added to D'Agents to reduce the burden on the system administrator[1]. Next problem is related with slow PGP, which also makes it impossible to generate session keys for ongoing communication. Replacing PGP with a better encryption library can solve this. Finally, the system is vulnerable to replay attacks in which an attacker replays a migrating agent or a message sent to an agent on a different machine. Here a server could have a distinct series of sequence numbers for each server with which it is in contact[1].

4. Comparison

Over the last few years a large number of mobile agent systems have been developed, both in the academic field and in the industrial one. Some of the systems did not address security problems at all, such as Messenger. Odyssey and Voyager don't support external security, even though they benefit from Java Security Manager. Aglets Software Developer Kit (ASDK) includes no agent authentication mechanisms except server domain authentication[11]. Three most influential mobile agent systems with authentication services are selected and discussed here. Unlike D'Agents, they are all based in Java and take advantage of the Java Security Model.

4.1 Three mobile agent systems

Concordia Concordia[13] by Mitsubishi Electric ITCA (MEITCA) is a mobile agent system that has a strong focus on security and reliability [2]. Agent hosts are protected from malicious agents through cryptographic authentication of the agent's owner[2]. The Security Manager authenticates each agent by verifying its identity, if the identity matches, the agent is able to access the resource[6]. The security level can be adjusted from the weak identity check to the strong authentication and security provided from external authorities [7].

Secure communications are implemented using SSL. Agent data is encrypted during transfer and storage, and security permissions for an agent depend on the user who launched the agent[9]. As in D'Agents, agents can either be owned or anonymous. Each agent associated with a particular user is assigned an identity, and carries a hash code of the user's password. The user is authenticated by a password that the agent carries, not by a certificate with a secure hash of agent code. The user's

passwords are stored in a global password file, which makes Concordia hard to scale up [4]. Also user identification does not guarantee that the agent contains the same code as when the user launched it[9].

Grasshopper Grasshopper[14] is developed by GMD FOKUS and distributed by IKV++, which is compliant to OMG' MASIF standard [8]. It supports any CORBA 2.0 compliant ORB. For internal security role based authentication and for external security PKI is used. Grasshopper makes use of SSL and X.509 public key certificates for secure communication and authentication. RSA is used for authentication and exchange of the symmetric session key. The actual communication is encrypted with DES. In order to take part in a SSL communication, each user must have a personal private key certificate from a trusted third party and the corresponding private key.

During the SSL negotiation the certificates including the public keys are exchanged and the protocol data are signed with the private keys. The combination of the right private keys and certificates authenticates both parties. Grasshopper allows the evaluation of chains of certificates, which occurs if both parties do not trust the same trusted third party directly. In the SSL handshake, the chain is verified step-by-step, until a top-level, self-signed certificate is found. If this certificate belongs to a trusted party, the authentication is successful. Otherwise the agency administrator is asked via the GUI, if he wants to trust the top-level certificate. If not, authentication fails. This also happens if one of the certificates in the chain is expired, or if one of the verification steps fails, e.g. when the chain is corrupt or inconsistent[12].

SOMA Secure and Open Mobile Agent (SOMA)[15] is designed to be an open platform at the University of Bologna, Italy. It is based on a thorough security model. And it interoperates with CORBA and conforms to emerging mobile agent standards MASIF. Each user in SOMA is authenticated by the security component (using X.509 certificates) and provided the access roles they are allowed. Each agent is defined by its owner identity and role. The owner digitally signs the agent's initial state, unique identifier and code and the roles are embedded in agent's state[5].

When a host receives an agent, it performs authentication of the agent using X.509 certificates and roles. In multi-hop systems, each host performs authentication checks before allowing the agent to execute in its environment. After authentication, the role certificate can be used to decide the authorization of the agent. So agents are

authenticated on the basis of their credentials[10]. SOMA protects the agents moving in an un-trusted environment in terms of both secrecy and integrity by using traditional encrypted and authenticated channels. Before being transferred from the current to the destination place, the agent is first encrypted and digitally signed by the current site[10].

4.2 Discussion

A comparison of the discussed systems is summarized in Table 1, which compares the following system security features.

- Internal Security Mechanism: Describes security mechanisms of the platform to prevent internal attacks from malicious agents[12].
- External Security Mechanism: Describes security mechanisms of the platform to prevent attacks from outside the agent system, e.g. cryptographic protocols[12].
- Authentication of Agent And Platform: Is the agent itself authenticated towards the agent system. Does the platform provide authentication among agent systems?
- Identify Agent: How does the system identify an agent?
- Public Key Distribution: How does the system distribute the public key?

Table 1 Security feature comparison

Criterion	D'Agents	Concordia	Grasshopper	SOMA
Internal Security Mechanism	policies with resource management	Based on Java Security Manager	Based on Java Security Manager	Based on Java Security Manager
External Security Mechanism	PGP (RSA)	SSL	X.509 certificates, SSL	X.509 Certificates
Authentication of Agent & Platform	Yes	Yes	Yes	Yes
Identify Agent	Digital signature using PGP	Global password	Their own strings	Digital signature. Only agents from untrusted domains needed
Public Key Distribution	No global distribution method	N/A	X.509 Certificates	X.509 Certificates

The above comparison gives us a general idea of what approaches can be taken to provide authentication services. All these systems support Java, but Java Security Manager only supports security for access control, not encryption and authentication. So in order to accomplish authentication, a unique identity needs to be assigned to each host in the system, the agents can be assigned an extended identity based on their creator, which can also define the access rights of the agents. Signed passport can be used to establish agents identity. To detect any tempering with agent's data, its static code can also be signed [5]. Secure transmission of agents over an untrusted network can be achieved using cryptography. Standard techniques of public/private keys provide a sufficiently secure implementation.

X.509 and PGP(or SSL) represent two different approaches to the distribution of trust on the Internet. X.509 is based on the concept of Certification Authority (CA) server, which is a centralized control of trust to certify and manage all the certificates. It is in natural opposition to the concept of the open network[4]. While PGP is totally decentralized, but very difficult to scale up the whole system. To ensure the quality of the secure services, a trade-off between uncontrolled distribution and central control must be considered in the authentication system.

Concordia and Grasshopper both employ SSL. In Concordia, each agent is associated with the same password after it is created, so there is no guarantee that the mobile code will be the same. In Grassopper, SSL uses RSA for authentication and session key exchange and the faster DES algorithm for the encryption of the data. In Grasshopper, agents are authenticated by their owner strings, and platforms authenticate themselves with cryptographic methods during the SSL handshake. Both Grasshopper and SOMA support X.509 certificates, so external certificates can be easily imported. They are unique from other Java based mobile agent systems as they also interoperates with CORBA, it can be integrated into CORBA-compliant environments. They both are compliant to the MASIF standard as well.

5. Conclusion

D'Agents is an open, academic system, which is available now on D'Agents web page[16]. It is a simple but powerful mobile-agent system, which supports multiple languages, even though it is still Tcl-centerical and Java's security advantage is not taken of. It provides reasonably good authentication services for the mobile agents.

D'Agents has been used in several information-retrieval applications including the technical-report searcher and 3Dbase[2] both at Dartmouth and in external research labs. As the developer planned to replace PGP with faster and more flexible encryption library supporting both public-key and secret-key cryptography and also add a modest key-distribution or certification mechanism, better authentication services and performance can be expected for D'Agents in the near future.

References:

[1] R. Gary, D Kotz, G. Cybenko, D. Rus, "D'Agents: Security in a multiple-language, mobile-agent system". In G. Vigna, editor, Mobile agent and security, volume 1419 of LNCS. Springer 1998. http://www.cs.dartmouth.edu/~rgray/

[2] R. S. Gray, G. Cybenko, D. Kotz, D. Rus, "Mobile agents: Motivations and State of the Art", in Jeffrey Bradshaw (Eds.), Handbook of Agent Technology, AAAI/MIT Press, 2000. www.cs.dartmouth.edu/reports/abstracts/TR2000-365/

[3] S. Berkovits, J. Guttman, V. Swarup, "Authentication for mobile agents", In G. Vigna, editor, Mobile agent and security, volume 1419 of LNCS. Springer 1998.

[4] P. Fu, "A security architecture for mobile agent system", University of British Columbia, Oct, 2001.

[5] G. Gupta, "Security issues in mobile agent-based systems", University of Southern California, 2001, http://www-scf.usc.edu/~gauravgu/resume/resume.pdf

[6] S. Adnan, J. Datuin, P. Yalamanchili, "A Survey of Mobile Agent Systems", CSE 221 Final Project, University of South California, June 13, 2000. http://www.cs.ucsd.edu/classes/sp00/cse221/reports/dat-yal-adn.pdf

[7] D. Horvat, D. Cvetkovic, V. Milutinovic, P. Kocovic, V. Kovacevic, "Mobile agents and Java mobile agents toolkits", Proceedings of the HICSS-2000, Maui, Hawaii, USA, Jan, 2000.

[8] S. Fischmeister, G. Vigna, R. A. Kemmerer. "Evaluating the Security Of Three Java-Based Mobile Agent Systems", in Proceedings of the IEEE IC on Mobile Agents 2001 Atlanta. http://www.softwareresearch.net/publications/C041.pdf

[9] O. Koskimies, "A survey on agent system supporting Java", Deparment of computer science, University of Helsinki, Sep, 1998.

[10] R. Broos, et al, “Mobile agent platform assessment report”, edited for the CLIMATE Sub-cluster Agent Platforms, http://www.fokus.gmd.de/research/cc/ecco/climate/ap-documents/miami-agplatf.pdf

[11] J. Altmann, F. Grabner, M. Gruber, L. Klug, W. Stockner, “Evaluation of agent platforms, Technical Report”, Software Competence Center Hagenberg, 2000.

[12] W. F. Farmer, J. Guttman, V. Swarup, “Security for Mobile Agents: authentication and State Appraisal”, Proceedings of the 4th European Symposium on Research in Computer Security (ESORICS '96), pp. 118-130. Rome, Italy, September 1996.

[13] Concordia Page. http://www.meitca.com/HSL/Projects/Concordia/

[14] Grasshopper Page. http://www.grasshopper.de or http://www.ikv.de/

[15] SOMA Page. http://lia.deis.unibo.it/Research/SOMA/

[16] D’Agents Page. http://www.cs.dartmouth.deu/~agent/