Access Control and Key Management for Mobile Agents

Volker Roth [1] and Mehrdad Jalali-Sohi [2]

Fraunhofer Institute for Computer Graphics, Rundeturmstr. 6, 64283 Darmstadt, Germany

Abstract

Security is a fundamental precondition for the acceptance of mobile agent systems. In this paper we present a mobile agent structure which supports authentication, security management and access control for mobile agents.

Key words: mobile agent security, agent authentication, key management, access control, access groups, itinerary

1 Motivation

According to the *Agent Society*, "Intelligent, and frequently autonomous and mobile, computer code known as agents represent the next great wave of innovation and development across the Infosphere comprised of the Internet, Intranets, Extranets, World Wide Web, and countless other networked computer systems. This arena has increasingly become very active, rapidly evolving, and expanding in scope and importance. The technology is expected to eventually have an effect as profound as the World Wide Web" [1].

However, a crucial factor for the actual success of this technology depends on our ability to *secure* it against attacks [4]. This poses great challenges in particular for *mobile* agents. While the processes involved in ordinary client/server systems profit from being executed in separate trusted computing bases, a mobile agent crosses the borders of different security domains and might be executed in an environment which cannot be controlled by the agent's sender. This poses a potential conflict of interests since the mobile agent might not be interested in keeping the integrity and

[1] vroth@igd.fhg.de
[2] jalali@igd.fhg.de

following the rules of its host or other agents hosted by it. Neither might the server be interested in honouring the agent's confidentiality and integrity.

Quoting from Karjoth *et. al.*: "It is our belief that applications based on aglets [†] will be widely accepted only if users are convinced that security services can cope with both kinds of threats" [7]. This might well be extended to other mobile agent systems as well. In the same article, the authors criticise that "the security frameworks of Java and other script languages for 'remote' programming such as Safe Tcl have allowed developers to make some progress toward one issue of mobile agent security – namely, the safe execution of untrusted code – through restricted environments based on sandboxing or a separated execution environment. . . However, no system at present provides a general security model".

Research in mobile agent security is catching momentum though. Starting with papers by Harrison, Chess, Farmer, Guttman, Swarup *et. al.* (see [5,4,12,11]) research now diversifies while authors concentrate on the various aspects of mobile agent security.

Lee, Gunter, Feigenbaum *et. al.* investigate the application of *proof-carrying code* to the malicious agent problem (see their position papers at the *DARPA Workshop on Foundations for Secure Mobile Code, March 1997, Monterey, Clifornia* [8,3,6]). Meadows proposes to lay out baits called "protection objects" in order to detect tampering on mobile agents [9]. Furthermore code obfuscation is investigated as a possible means to protect agents against analysis and manipulation by malicious hosts. Both approaches are criticised (in our opinion rightfully) by Sander and Tschudin due to a "lack of cryptographic strength" [10]. In the very same paper both authors promise to shake at the widespread belief that certain operations such as digitally signing a document offline in a secure way cannot be performed by a mobile agent on a malicious host [10]. The solution is based on *homomorphic encryption schemes*. Subsequent discussion and analysis of this approach might show wether this is feasible.

Yee describes an approach using two agents which is illustrated using the air-fare booking agent scenario which is frequently used to discuss mobile agent security related problems [2]. The protocol described requires a fixed itinerary for both agents, tolerates at most one malicious host and only "provided a solution for a special case", though. However, distributing security–critical operations over multiple agents seems to be promising and is also pursued by the authors of this paper.

Trusted hardware environments or *secure coprocessors* are sometimes considered as an alternative for software-only approaches with regard to protecting agents against malicious hosts. This approach is taken for instance in the *Sanctuary project* [2].

[†] aglet → IBM *Aglet Workbench* mobile agent

However, so far there seems to exists neither a framework which supports the integration of the numerous techniques proposed, nor an agent structure which supports cryptographic handling of agents, agent access control or key management for agents. In the remainder of this paper we intend to fill this gap.

2 Structuring Mobile Agents

The basic mechanisms for protecting the contents of a mobile agent are encryption and digital signatures. The former for ensuring the confidentiality and the latter for ensuring the integrity and authenticity of parts of the agent.

While integrity protection and authentication of an agent's *static* parts is easily done by applying digital signatures this is not straightforward for its *state* – or in other words – its *mutable* parts. Since "a migrating agent can become malicious by virtue of its state getting corrupted" [12], this poses a severe problem. A server either needs some kind of state appraisal mechanism in order to verify the correctness of all the agent's state transitions, or it has to form trust relationships (again using digital signatures) with other servers (see e. g. [11]). Detection and non-repudiation of such trust violations are in principle possible for mobile agents with a fixed itinerary [4] and consequently also for those whose loose itinerary can be securely determined a-posteriori.

Confidentiality of an agent's contents might be subject to complex access control policies. The sender of an agent might want to conceal data the agent collects on one server from other servers to which the agent might migrate subsequently. The hosting server might also have substantial interest that such data is not disclosed to other servers on the agent's itinerary. On the other hand a host might want to pass the agent credentials or other data which should be readable for and only for particular other servers. Regarding the various types of data required for the management and execution of mobile agents (e. g. class code, serialized streams of class instances, security policies, encrypted keys, state information, information payloads, certificates, etc.) a mobile agent is probably best associated with a compact database or filesystem which supports easy searching, extraction, insertion, grouping and protection of individual records.

For this reason we propose a tree-based agent structure which is illustrated in Figure 1. This structure is comparable to an ordinary hierarchical filesystem; it supports hierarchical signing and encryption of agents, key management, agent security policies, and data mangement. Nodes of the tree might either be *folders* or *files*. A folder's *domain* is the set of its descendants. A file's domain is its contents. Each node has an identifier (or name) which is unique among its siblings. If a node is labeled *signed* then a digital signature is applied to its domain. If a node is labeled *encrypted* then its domain is encrypted and integrity protected. An encrypted do-

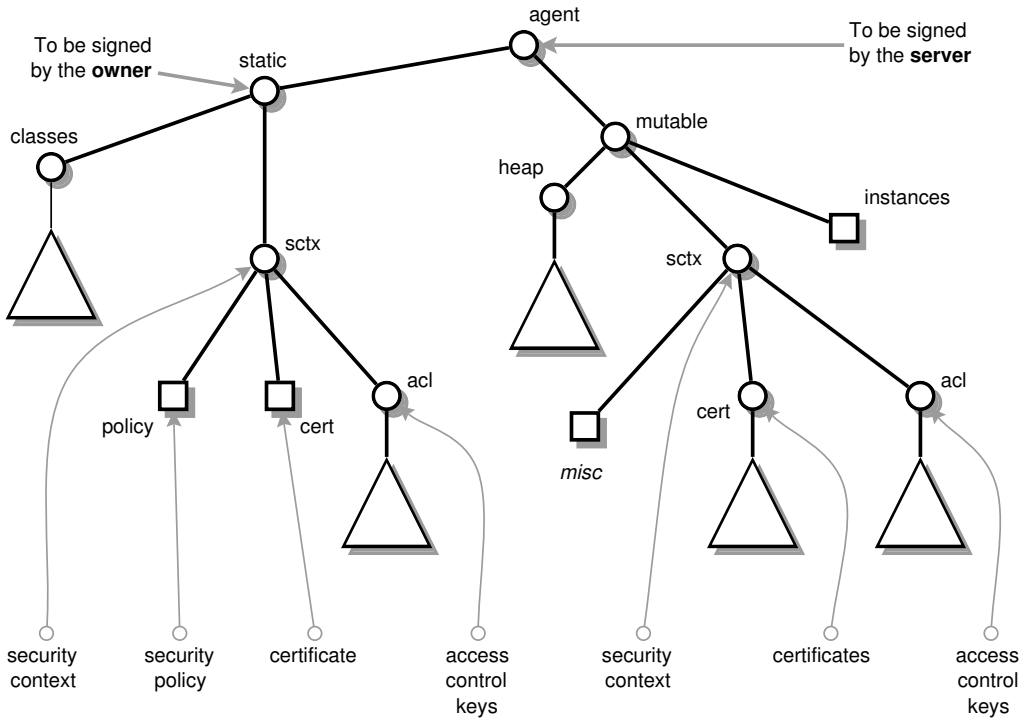

Fig. 1. The basic structure.

main is tagged with the name of the decryption key and a signed node is tagged with the signer's identity. As a matter of consequence, domains of nodes which are signed are readable but not writable unless the writer can generate valid signatures, and domains of nodes which are encrypted are neither readable nor writable unless the reader or writer has the correct key. Signing node *static* thus protects the complete left branch of the tree shown in Figure 1 against tampering.

So far it is still possible to simply erase certain nodes (and consequently its descendants also) or to modify and re-sign nodes using a different identity. Hence, a standard or a security policy has to specify the existence of a minimal set of mandatory nodes, as well as a mapping from signers of these nodes to roles in the agent model.

In our model, the signer of node *static* is interpreted as the rightful owner of the agent. The domain of *static* contains all data of an agent that does not change during its life time. The digital signature of the agent's owner renders it read-only. The agent server which sponsored the last state change of the agent has to sign the root node *agent* thus binding the state to the agent. Quoting from Chess *et. al.*: the state information "can be signed only by the previously visited AMP [†], where the agent reached its current state. In deciding how much to trust a stateful agent, AMP must consider how much it trusts not only the originator, but also all preceeding AMPS" [4].

[†] AMP → *Agent Meeting Point*

The *static* branch of the agent also contains the bytecode of its classes. For obvious practical reasons we concentrate on support for object-oriented agent programming languages with Java in particular in mind. Numerous disclaimers for software products tell us that no software company is willing to assume responsibility for their products as is obligatory for almost any other industry. However, we feel that developers of agent class packages should digitally sign their packages such that the incopration of bombs, back doors and viruses can not be repudiated later. This protects users as well as developers from incrimination and tightens the web of trust which is required for a mobile agent infrastructure. The signing of class packages is well supported by Sun with their JAR tools (see also Section 4).

Instantiations of classes are stored in node *instances* of the *mutable* branch. This branch also contains a *heap* folder whose intention is to serve as a general storage space for the agent. Both, the *static* and the *mutable* branch contain a security context *sctx*. This context stores the access control keys (see Section 3) and security management information, e. g. certificates required to authenticate the signer of some agent part. The agent's security policy is stored in the security context of the *static* branch and might thus not be altered by entities other than the agent's sender.

3 Access Control and Key Management

Each domain $i \in \{1, \ldots, n\}$ of the agent's structure might be encrypted with a particular key k_i. A node whose domain is encrypted is tagged with a key identifier $\mathrm{N}(k_i)$. Access groups to this domain are defined by access to the encryption key. If a domain's access group contains only a single server then in principle the encryption key might be the public key of that server (which is not recommended, though). Otherwise, the symmetric key k_i is chosen randomly and sealed for each server $s \in \{1, \ldots, m\}$ in the domain's access group using the servers' public keys. Hence, $\mathrm{seal}_s \leftarrow \mathrm{E}_s(k_i)$. The seals are stored in a *sctx/acl* folder.

Since a node might be tagged with at most one key identifier, this leads to a potential name conflict. Servers s and $\tilde{s}$ both need to locate different seals using the same identifier.

There are two immediate solutions for this problem. The first solution is to create a folder for each server (which is in at least one access group) within the *sctx/acl* folder using a unique distinguished server name. For each domain a server should have access to, the corresponding encryption key k_i is copied into the server's key folder using the key identifier $\mathrm{N}(k_i)$ as the name. This folder is subsequently encrypted and the resulting seal is placed in the *sctx/acl* folder. The key identifier for the server's key folder might be chosen freely and hence without name conflicts this time. When the agent arrives at a server, the server locates the key folders meant for it, finds the corresponding seals and decrypts the folders. Since a server can

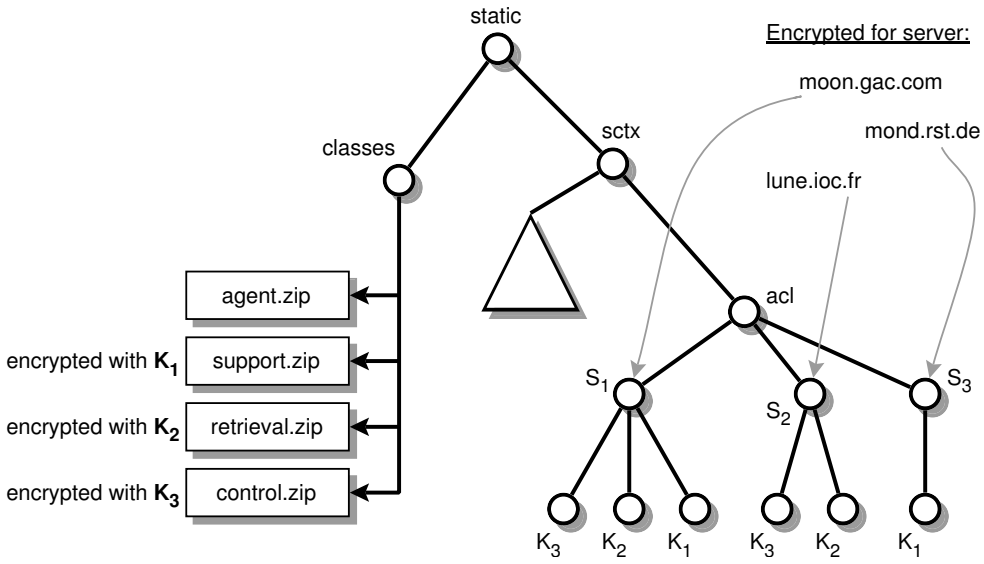

Fig. 2. An example of different access groups.

not decrypt (and therefor read) key folders of other servers, no name conflicts can occur during consecutive key searches.

The second solution is to combine a unique server name with the identifier of the encryption key, e. g. by hashing or concatenation. The resulting name is used to store the seal $\mathrm{E}_s(k_i)$ in folder *acl*. The first solution is illustrated in Figure 2; the seals required for decryption are not shown in the figure.

In order to be useful, the basic agent structure needs an accepted and agreed-upon interpretation. This interpretation is part of a non-negotiable (in contrast to negotiable) security policy which e. g. specifies that:

- the agent's *static* branch must be digitally signed
- the signer of the *static* part is assumed to be the rightful owner of the agent
- the server which sponsored the agent's last state change has to digitally sign the complete agent
- the order for searching seals and encryption keys is to descend first in *static/sctx/acl* and *mutable/sctx/acl* second for locating key identifiers
- if e. g. a fixed itinerary is given in the agent's *static* part then a log file must be present in its *mutable* part where all changes to the agent are authenticated and all servers' signatures must appear in the order given in the itinerary

The client software used by an agent's owner as well as agent servers might only support a limited set of ciphers. Therefor it is mandatory to match the cipher suites available to the server and to the client. Security preferences such as this might be formalised and stored in *static/sctx/policy* where the agent's security policy is defined. Since it is placed in the static branch only the agent's owner is able to change the security policy without breaking the agent's integrity. The *fair* negotiation of

negotiable security preferences requires the cooperation of the server, though.

4 The JAR Container

The agent structure described above is a logical structure which might be mapped on any representation which supports the efficient hierarchical signing an encryption of tree-like data repositories. A very suitable one for this purpose is the *Java Archive* (JAR).

A JAR file is basically a ZIP archive with a folder called `META-INF` which contains a manifest file `MANIFEST.MF`. For each file in the archive (except those in the `META-INF` folder) the manifest file contains an entry with the path to the file in the archive and a cryptographic hash of the file's contents. Files may be signed by a *signer*. Each signer is represented by a signature file in the `META-INF` folder with the suffix `.SF`. For each file listed in the manifest file which is to be signed by the signer the signature file contains an entry with the file's name and a cryptographic hash of its entry in the manifest file. The signature file is then signed and the signature is stored in the `META-INF` folder in a file of similar basename with a suffix of either `.RSA`, `.DSA`, or `.PGP` depending on the type of signature applied to the signature file. This allows multiple signers as well as multiple and overlapping signatures.

The `jar` tool distributed with JDK 1.1 however only supports the signing of a complete archive. For representing the mobile agent structure in a JAR file two extensions must be made:

- support for more fine–grained signing of JAR files must be programmed. This is facilitated by the Java packages *java.util.zip* and *java.security*.
- encryption is not supported by JAR files. Hence, extensions must be developed which allow the partial encryption of JAR contents in the way described in Section 3.

The structure of a mobile agent might thus even be edited in a subdirectory of some harddisk's filesystem. The directory tree of the agent might then be compressed using tools such as `PKZIP` or `jar`. A number of existing tools such as PGP might be used for signing and encrypting agents.

5 Conclusions

Above we presented a flexible and extensible structure for the representation of mobile agents which supports hierarchical access control, and proposed an initial

interpretation of this structure with respect to the roles in a general mobile agent model. The structure maps well on existing and widely distributed technology. Furthermore the interpretation is an initial step to define a security policy for mobile agents and agent servers as well; it also facilitates security management and the implementation of security services for both while supporting basic agent tasks and operations – the management of data.

The structure as well as the interpretation should undergo a refinement process while being explored within a security centered mobile agent framework. Complemented by a discussion of advisable and suitable extensions and modifications, the results should probably be input to a standardization process with the intention to establish a portable and interoperable agent structure which is acceptable to all major agent facilities.

References

[1] AGENT SOCIETY. Agent Society Main Frame. Internet WWW page, at URL http://www.agent.org/frmain.htm. (version current at 27 Nov 97).

[2] BENNET S. YEE. A Sanctuary for Mobile Agents. DARPA Workshop on Foundations for Secure Mobile Code, Monterey, CA, USA, March 1997. Position Paper.

[3] CARL A. GUNTER, HOMEIER, P., AND NETTLES, S. Infrastructure for Proof–Referencing Code. DARPA Workshop on Foundations for Secure Mobile Code, Monterey, CA, USA, March 1997. Position Paper.

[4] CHESS, D., GROSOF, B., HARRISON, C., LEVINE, D., PARRIS, C., AND TSUDIK, G. Itinerant agents for mobile computing. *IEEE Personal Communications* (October 1995), 34–49.

[5] COLIN G. HARRISON, DAVID M. CHESS, AND AARON KERSHENBAUM. Mobile agents: Are they a good idea? Tech. rep., IBM T. J. Watson Research Center, P.O. Box 704, Yorktown Heights, NY 10598, March 1995.

[6] FEIGENBAUM, J., AND LEE, P. Trust Management and Proof–Carrying Code in Secure Mobile–Code Applications. DARPA Workshop on Foundations for Secure Mobile Code, Monterey, CA, USA, March 1997. Position Paper.

[7] KARJOTH, G., DANNY B. LANGE, AND OSHIMA, M. A security model for aglets. *IEEE Internet Computing* (July·August 1997), 68–77.

[8] LEE, P., AND NECULA, G. Research on Proof–Carrying Code for Mobile–Code Security. DARPA Workshop on Foundations for Secure Mobile Code, Monterey, CA, USA, March 1997. Position Paper.

[9] MEADOWS, C. Detecting Attacks on Mobile Agents. DARPA Workshop on Foundations for Secure Mobile Code, Monterey, CA, USA, March 1997. Position Paper.

[10] TOMAS SANDER, AND CHRISTIAN F. TSCHUDIN. Protecting Mobile Agents Against Malicious Hosts, November 1997. submitted to the forthcoming LNCS on "Mobile Agent Security".

[11] WILLIAM M. FARMER, JOSHUA D. GUTTMAN, AND VIPIN SWARUP. Security for mobile agents: Authenication and state appraisal. Tech. rep., The MITRE Corporation, 202 Burlington Road, Bedford, MA 01730-1420, 1996. To appear in the Proceedings of the European Symposium on Research in Computer Security (ESORICS 96).

[12] WILLIAM M. FARMER, JOSHUA D. GUTTMAN, AND VIPIN SWARUP. Security for mobile agents: Issues and requirements. Tech. rep., The MITRE Corporation, 202 Burlington Road, Bedford, MA 01730-1420, 1996. To appear in the Proceedings of the National Information Systems Security Conference (NISSC 96).