Mobile agent security based on payment

Michael Sonntag, Rudolf Hörmanseder

*(Institute for Information Processing and Microprocessor Technology (FIM),
Johannes-Kepler-University Linz, Altenbergerstr. 69, A-4040 Linz, Austria.
E-Mail: sonntag@fim.uni-linz.ac.at, hoermanseder@fim.uni-linz.ac.at)*

Mobile agents are autonomous entities that handle tasks for their owner. Agents act on their own by reacting to changes and by planning their course of action. These agents can move from one server to another. In the future, agents will also be supplied with real money in some form to pay for resources or services.

In this paper we discuss a dynamic security architecture, in which permissions are assigned in exchange for information (money). The decision as to which permissions are available, as well as how much they cost, is based on the source of the code, the owner/user of the code and what other information the agent is willing (or able) to provide.

We discuss the advantages and limitations of assessing permissions in monetary terms, rather than binary granting or denial of permissions according to pre-set classes. A test-framework has been implemented using Java.

1. Introduction

In the scenario envisioned in this paper, mobile agents ([Kotz99]) possess some valuable information, such as e-cash or a credit-card number, and thus have to be protected through extensive security [Vigna98]. The agents need to be protected from each other on a host, as well as during transmission, which would preclude tampering. Protecting agents from the host is currently not possible [Tschudin99]. These transportable valuables allow agents to use permissions and resources on servers, which would not be available to them otherwise. Thus agents only receive certain resources by paying for them.

In many cases, agents will not provide this information initially, but only at a later point in time. Consider this example: an agent looks for a certain service anonymously and reveals its identity (and credit-card number) only to the server on which the service is ultimately purchased. This requires a dynamic security architecture, in which permissions can change over time (unlike e. g. Aglets [Aglets] or Java [Gosling96] itself).

2. A security model for mobile agents based on money

Our security model is based on the (in fact, capitalist) assumption that (almost) every permission and every resource can be assessed in terms of money. Some of these may be very cheap (e.g. free for agents of employees in a particular company) or very expensive (hitherto unknown agents) or not available at all (e.g. infinite price for agents without certificates, which ensures denial). Permissions are allocated according to the origin of the code as well as the

owner who is responsible for the execution of the code. Both identities are established and verified by the use of certificates. Agents that cannot produce a certificate will usually be treated very cautiously or not granted a permission at all. The price of permission reflects neither the availability of resources, nor the load of the host, as in electronic market models such as [Bredin98] or [Yemini98] (although this can be included); but it does reflect the cost and risk to the host in granting permission.

2.1. Discussion of the model

This model has a number of advantages over binary decision models based on granting or denial of permissions according to a limited number of classes (e.g. a permission may or may not be used according to the category of an agent: "our" agents, trusted agents, foreign agents, …).

Fine-grained security. Each permission is assessed according to the risk of being used by certain code or persons, as well as according to the consequences of possible misuse. One example is the following. Many providers grant permissions to unknown agents, despite the danger of misuse, if the agents pay more than the cost of adding disk space or temporarily denying access to other agents, when resources become exhausted. This also allows small differences (e.g. a slightly higher price) between groups of agents, which would not be possible if just granting or denying access.

Ease of understanding. Everybody can understand the policy used, as the value charged for the permission correlates to the real costs and the estimated risk. This is also easier to communicate to, for example, managers who are not actively involved in establishing the security policy, but who are proficient in assessing risks and calculating necessary prices.

Billing. This model lends itself perfectly to billing for services (constrained execution). Since permissions may not only consist of a certain type of access, but also of a certain quality-of-service or amount (e. g. high priority, a temporary file up to 150 kB, use of special hard- or software, etc.), agents can pay for the precise services they need. Servers can utilize their resources to the full, since they can distribute them to the highest bidder and need not reserve them for possible future use by an agent that may actually never use it. Another option would be to charge agents for the time or amount a privilege is used or granted.

There are, however, some limitations to this model.

Unsuitability for classification according to creditworthiness. A major drawback of this scheme is that agents cannot be directly classified according to their creditworthiness. It makes no sense to charge them more money if it is more likely that they will never actually pay their debts. Therefore, payment methods for agents must either be equivalent to cash (immediate payment, e.g. e-cash) or agents must attain a higher trust level before being able to buy privileges at all (or using forms of payment other than cash, such as debiting from an account).

Large number of decisions. As there are three aspects defining the actual price of a permission (origin of the code, owner of the agent, and permission desired), configuration is com-

plicated. If each of these three aspects possesses only 4 groups, there are already 64 combinations altogether. For each of the 64 combinations, a decision whether, and at which price (=risk), the permissions should be granted must be made. Because of this large number, a viable pre-configuration, suitable for most cases, is needed. Grouping sets of permissions together eases the work needed for configuration, but the necessity remains to build groups, assess the risk-levels and calculate the prices.

Impossibility of delegation. Trusted code cannot delegate a task to another code because the necessary permissions cannot be transferred. This limitation is inherent in the model, since each agent is assessed only according to the source of its code, and owner. It may be possible to allow an agent to be the owner of other agents, thus creating a hierarchy of agents that ultimately terminates at an external owner - certified by a recognized certificate authority - at the top of the tree. This would be problematic since the transmission of trust along a series of agent-owners is even harder to evaluate than the direct association of an agent to an owner. (For example, how well do you trust the sub-agent of the sub-agent of your own agent, if the agent you created selected them itself?)

Permissions which cannot or should not be strictly enforced are especially suited to dynamic modification. One example is the amount of hard disk space an agent may use. Even the agent itself probably cannot tell exactly how much space it is going to need; hence, it might purchase a minimum amount of space, and may be granted an optional quota according to the trust it holds (which may be unlimited). The system hosting the agent can set an upper limit, forcing the agent to purchase more space. In the meantime, the host trusts the agent that it will pay for the space actually used. This disk space may be cheaper (if, for example, the agent has always promptly paid for the services used) or more expensive (if, for example, the host knows nothing about the agent). In any case, if the agent does not pay, the owner could be billed directly according to the agent's certificate. This would be more cumbersome and expensive and would take longer; the agent must therefore pay a premium for the risk to the system and owner.

2.2. Basic evaluation according to the code-signer and the owner

In order to capture both static and dynamic aspects, security is based on two considerations; the origin of the agent's code (static; signed code); and who the owner of the agent is (dynamic; identity of the agent). This distinction is necessary as most users will employ agents created by other persons.

The code of an agent from a well-known producer will have a larger set of options than code from unknown companies or persons, since it is likely to be of higher quality (fewer bugs and less possibility of misuse). Also, the probability of the code being infected by a virus is lower in the case of a professional company that uses extensive countermeasures against these risks [Garfinkel97]. Since an agent must show (at least some) intelligent behavior to be classified favorably, much depends on its parameterization, which is done by the owner of the agent. Even a perfectly innocent code can be tricked into doing something harmful through misleading parameters. (For example, enquiries may be made by a shopping agent about illegal

or embarrassing items under a false identity.) It is therefore necessary also to include this aspect when considering the permissions to be granted to an agent. To ease administration, producers of code and owners can be grouped according to the following considerations.

- *Producers of code.* Grouping is facilitated by the certificate whose private key was used to sign the code and name of package or class. Certain servers may also accept unsigned code, although the permissions will usually be very restrictive. These groupings can be very fine-grained, depending, for example, on which certificate authority provided the certificate, and therefore on the amount of guaranteed compensation (i.e. liability) in the event of wrong content. Individual assignment according to a list of trusted agent-code is also possible.
- *Owners of agents.* The distinction between groups depends on whether the agent can (or is willing to) provide an owner-certificate, or, when applicable, whether the agent belongs to an arbitrary selection of individual users. Again, agents without a certificate, or with a certificate that was not issued by a recognized certificate authority, will be more limited in their actions. Since security is based on monetary aspects, grouping according to owner can also be carried out by considering the way the agent wants (or is able) to pay for additional permissions. In this case, agents paying by electronic cash will receive permissions more readily than those paying by invoice. Agents can also be divided into groups based on the level of warranty the certificate authority provides, or on the company the owner belongs to or is authorized to sign for (this information might also be included in a certificate).

At any given moment, an agent belongs only to one code group and one owner group, respectively. The groups to which an agent is assigned can change over time (e.g. if the agent provides some proof of identity, or if checking of certificates used for code-signing is deferred to a later time). So a dynamic change of permissions is required. This dynamic behavior of the system is in strong contrast to the static model of permissions in standard Java security [Jsec].

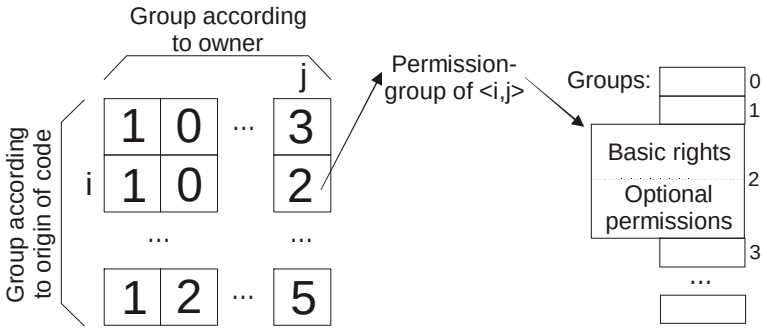
Permissions within a group are further divided into two subgroups, as follows:

Basic rights. These are permissions an agent (of this group) always owns, because they are needed for basic work. These permissions have a negative cost level to show that they are assigned to the code automatically upon arrival or start.

Optional permissions. These are permissions an agent may receive under special circumstances; for example, by making a payment, or on a temporary basis. These permissions may cost nothing for some agents, or medium or large sums for others, and may be unavailable for other agents. This concept restricts the agents to a minimum possible set of rights while allowing them selected permissions on demand if additional conditions are met. Some agents are restricted from ever getting certain possibly dangerous permissions.

The *actual set of permissions* granted to an agent is retrieved from a matrix (see figure below), and reflects the level of trust of the code origin and the owner. In many cases, the resulting permissions will be the intersection of those of the agent code and those of the owner.

Nevertheless, there are cases in which special permissions apply. Indirection via a matrix allows the implementation of a very flexible policy that can be adapted to almost any need.



2.3. Advanced evaluation according to additional classification information

To allow more flexibility, an agent can offer some additional classification data in order to gain more permissions or to obtain a cheaper price. This fact could be modeled as additional dimensions within the matrix. Nevertheless, these classifications can be very diverse and would therefore not usually be standardized.

Additional classification could be used with “negative permissions” too. If an agent misbehaves, the system might dynamically revoke certain permissions. In this way, one can include history-based access control, as described by [Acharya97], because the identification of an agent can be carried out using code checksums or certificates. On the other hand, based on recorded information, cheaper or more permissions (e.g. reduced prices) might be offered to regular customers.

Classifications can have a number of different states.

- *Request check.* This is only a temporary state, during which the agent provides information and requests the system to verify it.
- *Unknown classification.* The agent has provided some additional information, but the system did not recognize it. (This can occur in the absence of standardization.)
- *Check currently impossible.* The agent system will sometimes be unable to check the classification information; e.g. when no connection to a certificate authority can be established. Because this is a temporary error condition, the agent is free to request this check again later.
- *Check impossible.* The system cannot check the information offered because of some permanent problem. Resubmission is discouraged since no change is likely to occur. An example is a signature with an unknown certificate authority.
- *Rejected.* The information has been checked, but the check failed. This contrasts with the “check impossible” state in that the checking procedure did not fail, but the content contained an error (e.g. the identity of the agent claiming to be a regular customer could not be found in the database).

- *Accepted.* The classification information was accepted (and possibly checked).

Examples of additional classifications and their consequences include:

- Regular customers (i.e. who have already been in the system and have bought permissions several times without problems) receive a discount.
- If an agent has bought a large number of units in the past, the signer of the code will receive a commission or a credit entry.
- By providing a membership card (=certificate), the agent is allowed to use a local drive.
- If some money has to be refunded to an agent, this can be done through an additional classification ("voucher").

3. Payment for permissions / services

There are two different methods of payment:

Prepaid. The agent has to pay in advance; i.e. before permission is granted. This payment method favors the system since the system can remove permission at any time.

Postpaid. Payment when a permission is returned or the agent leaves the server; this clearly favors the agent. If it is unwilling or unable to pay when the permission is returned, the system either loses money or has to look elsewhere (e. g. other agents of the same owner or the owner itself) for it. This is similar to extending credit to the agent, and might therefore be limited to a certain amount, with the consequence of requiring payment by installments.

Postpayment allows permissions to be paid according to the amount of resource usage (e.g. cpu time, hard disk space, ...). The alternative – in which the agent has to pay for a certain amount of service in advance - puts the agent at a significant disadvantage, and complicates the situation: the system has to refund money for unused services that have been paid for.

The agent or its owner must also receive a detailed invoice (period of use, price, classification information), which would enable the owner to verify the transactions (in contrast to the lack of information on, for example, credit card invoices, which typically contain only a single line of text). Such an invoice, in combination with signed evidence of granting and payment for permissions, could be used in automatic dispute resolving mechanisms.

Payment for services is currently implemented only in a rudimentary way, partly because no practical solution for secure exchange of payment between host and agent is available. We assume a trust relationship between the agent and its host. This is not an exceptional assumption, because, for example, every customer has to trust his computer center. Hence, in the future every mobile agent may carry a list of trusted servers.

If trust is not possible, then the only solution to the problem of secure exchange is the post-payment option: the agent has to pay after it has left the host and has arrived at a trusted host. This extended version of post-payment allows secure exchange using a trusted third party; the communication link to the third party is controlled by neither of the two parties in the trans-

action. (Even sophisticated exchange protocols, such as the one presented by [Zhou], rely on guaranteed and unchanged delivery of the messages to the third party.)

An interesting possibility of securing agents from the host they reside on is described in a paper by [Sanders98], in which private data and the program are inseparably mixed. This concept allows an agent to hide its private data and keep it secret even from the host it executes on. However, it is rather limited at the moment, and it is unclear whether it is extendible to general programs.

4. Conclusions

In this paper, we have presented a system in which permissions are bought, and mobile agents are classified, according to owner and code, as well as according to additional information agents are able or willing to provide. Advantages and disadvantages of this approach have been presented, and the methods of payment were discussed. Issues concerning payment were explained. The main problem is lack of security in the exchange of information during the period when the agent is on the host that demands payment.

5. Future Plans

The dynamic security system, as it is described here, has been implemented in Java as a first prototype in a test-framework for mobile agents. Further work on design and implementation is planned especially in the following areas:

- *Payment for permissions.* Currently, permissions are allocated only when paying in advance. This shall be extended by storing the time or amount of usage, and by introducing a scheme for paying by installments or paying after use. An invoice signed by the system will also be created and transmitted to the agent upon leaving the server.
- The set of permissions supported will be extended to threads and priorities, as both can be controlled from within the Java virtual machine.
- Currently, certificates are checked only against stored root certificates of certificate authorities. These groups shall be enhanced to include the result of a check whether certificates have been revoked.

We plan to do further research on a special problem related to the use of mobile agents; namely, how can the agent check that it gets what it pays for? For example, if some permission is only granted for an agreed span of time, the agent can only guess the amount of time it has used through the work it has completed. Nevertheless, (e.g. with different priorities) the amount of work will vary widely, and so no hard evidence can be produced. One (not completely satisfactory) suggestion is to use timestamps from a trusted third party.

Acknowledgment

This paper is a result of a project sponsored by the Anniversary Fund of the Austrian National Bank (Project 7742).

References

[Acharya97] Anurag Acharya, Guy Edjlali: History-based Access Control for Mobile Code. Technical report TRCS97-25 (ACM-CCCS-98)

[Aglets] IBM Research Lab: Aglets Homepage http://www.trl.ibm.co.jp/aglets (7.10.1999)

[Bredin98] Jonathan Bredin, David Kotz, Daniela Rus: Market-based Resource Control for Mobile Agents. In: Proceedings of the Second International Conference on Autonomous Agents. ACM Press 1998

[Garfinkel97] Simson Garfinkel, Gene Spafford: Web Security & Commerce. Sebastopol: O´Reilly 1998

[Gosling96] James Gosling, Bill Joy, Guy Steel: The Java Language Specification. Addison-Wesley, 1996

[Jsec] J. Steven Fritzinger, Marianne Mueller: Java Security. http://java.sun.com/products/jdk/1.2/docs/guide/security/spec/security-spec.doc.html (12.19.99)

[Kotz99] David Kotz, Robert S. Gray: Mobile Agents and the Future of the Internet. Operating Systems Review 3/1999, 7-13

[Kun00] Yang Kun, Guo Xin, Liu Dayou: Security in Mobile Agent System: Problems and Approaches. Operating Systems Review 1/2000, 21-28

[Sanders98] Tomas Sanders, Christian F. Tschudin: Protecting Mobile Agents Against Malicious Hosts. In: G. Vigna (Ed.) Mobile Agents and Security. Berlin: Springer 1998 (Lecture Notes in computer science; Vol. 1419)

[Tschudin99] Christian F. Tschudin: Mobile Agent Security. In: Matthias Klusch (Ed.): Intelligent Information Agents. Agent-Based Information Discovery and Management on the Internet. Berlin: Springer 1999

[Vigna98] Giovanni Vigna (Ed.): Mobile Agents and Security. Berlin: Springer 1998 (Lecture notes in computer science; Vol. 1419)

[Yemini98] Y. Yemini, A. Dailianas, D. Florissi, G. Huberman: MarketNet: Market-Based Protection of Information Systems. In: Proceedings of the First International Conference on Information and Computation Economies. ACM Press 1998

[Zhou99] Jianying Zhou: Achieving Fair Non-repudiation in Electronic Transactions. Journal of Network and Computer Applications. Academic Press (to appear)