ON MOBILE AGENT BASED TRANSACTIONS IN MODERATELY HOSTILE ENVIRONMENTS *

Niklas Borselius, Chris J. Mitchell, Aaron Wilson
Mobile VCE Research Group, Information Security Group,
Royal Holloway , University of London
Egham, Surrey TW20 OEX, UK
Niklas.Borselius@rhul.ac.uk, C.Mitchell@rhul.ac.uk, aaron@gmx.co.uk

Abstract When using mobile agents, numerous security issues must be considered. In this note we propose two methods to improve the security and reliability of mobile agent based transactions in an environment which may contain some malicious hosts.

Keywords: mobile agent, digital signature, transaction, security

1. Introduction

In this paper we consider strategies for the deployment of mobile trading agents to reduce certain security threats to their operation. In a future world of co-operating mobile and fixed devices, the mobile agent computing model is expected to become an increasingly important one. In the domain of e-commerce/m-commerce transactions, mobile trading agents could play a very useful role. Users could launch such agents to make transactions on their behalf, and the agents would look for the 'best buy' by visiting multiple merchant sites without any direct user intervention. Indeed such activity could take place while the user has no current network connectivity.

The mobile agent computing model gives rise to a range of security threats. These threats can be divided into two main classes:

*The work reported in this paper has formed part of the Software Based Systems work area of the Core 2 Research Programme of the Virtual Centre of Excellence in Mobile & Personal Communications, Mobile VCE, www.mobilevce.co.uk, whose funding support, including that of EPSRC, is gratefully acknowledged. More detailed technical reports on this research are available to Industrial Members of Mobile VCE.

- threats to the platform from malicious and/or unauthorised agents, including threats to the integrity of the platform and other agents, threats to the confidentiality of stored data, and denial of service threats, and
- threats to the agent from malicious platforms, including threats to the confidentiality of agent stored data, and threats to the integrity of the agent and its computations.

In this paper we are concerned with the second class of threats, and in particular with threats to agents deployed for trading applications. Specifically, users will need to give trading agents certain authority to authorise transactions, whilst at the same time users will wish to protect themselves against malicious merchants forcing an agent to make a non-optimal purchase.

We consider simple ways in which deployment of multiple agents can reduce the threat to trading agents from platforms outside of their direct control. We consider two general approaches. In the first approach multiple agents are equipped with 'shares' of the means to commit to a transaction. In the second approach a single trusted host provides a location for multiple agents to 'report back' information enabling a purchasing decision to be made.

The paper has the following structure. The next section explores threats to trading agents in more detail. This is followed in Sections 3 and 4 by a discussion of the models used here for agent platforms and for trading agents. Sections 5 and 6 then explore the two approaches to enhancing trading agent security.

2. Agent Security Issues

The use of mobile agents raises a number of security concerns. Agents need protection from other agents and from the hosts on which they execute. Similarly, hosts need to be protected from agents and from any party which can communicate with the platform. The problems associated with the protection of hosts from malicious code are quite well understood.

The problem of malicious hosts seems the hardest to solve. In fact some people hold the opinion that it is insoluble. The particular attacks that a malicious host can make have been described in [Hohl, 1998a] and [Hassler, 2000], and can be summarised as follows.

- Observation of code, data and flow control,
- Manipulation of code, data and flow control – including manipulating the route of an agent,

- Incorrect execution of code – including re-execution,
- Denial of Execution – either in part or whole,
- Masquerading as a different host,
- Eavesdropping of agent communication,
- Manipulation of agent communication,
- False system call return values.

There have been many attempts to address these threats either completely or in part. Most of these attempts fall into one of the following broad categories.

- The first category comprises approaches that do not allow an agent to leave a trusted environment. Solutions to this include using a host infrastructure that is operated by a single party, allowing agents to migrate only to trusted hosts [Farmer et al., 1996], or possibly hosts with a good reputation [Rasmusson and Jansson, 1996].
- The second category is pragmatic; it consists of solutions to a single part of the malicious host problem. These consist of agents detecting when they have been modified [Vigna, 1997], and proof verification techniques [Yee, 1997].
- The third class consists of assuming that there is special, tamper-proof hardware available, see for example [Yee, 1997] or [Wilhelm et al., 1998].
- The final category uses software methods to obscure the code from the host. Approaches include obfuscation [Hohl, 1998b] [Ng, 2000], mobile cryptography [Sander and Tschudin, 1998, Sander and Tschudin, 1997] and using environmental conditions to hide parts of the code [Riordan and Schneier, 1998].

The approaches described in this paper, based on replicating agents, do not fit into any of the above four classes. There appears to be relatively little literature devoted to this approach to dealing with the threats to agent security.

We now consider the threats to a trading agent in more detail.

2.1. Threats to trading agents

We now turn to look at the particular threats to an agent which wishes to purchase an item (or a service) from a merchant. These all fall into the categories above. We concentrate on the threats to an agent involved in a trade, rather than more general threats.

1 A malicious host lies about offer.

   Here a host lies about the offer it makes to an agent, in order to get the trade. The host would then charge a higher price at a later date. One way around this is to force the host to sign its bid, thereby committing to it.

2 A malicious host learns other offers and undercuts them.

   If a host knows that all offers but its own have been collected and finds out the best standing offer, it can undercut the best standing offer slightly (in fact the host need not know all other offers, it could just undercut the current offers). (In some circumstances letting hosts undercut each other might be considered a desirable feature.)

3 A malicious host learns the price a user is prepared to pay and bids just under this.

   In a similar fashion the host may charge more than its normal price, if it knows the maximum price the user is prepared to pay. Thus a host must be kept from learning the maximum price a user is prepared to pay, either by encrypting this information or by not sending this information with the agent.

4 A malicious host manipulates the requirements.

   This is when the host changes the requirements to favour its bid. For example, it could add a requirement to buy from a certain host, or remove constraints from the agent.

5 A malicious host alters the agents route.

   Here, the host keeps the agent away from its competitors, and thus secures the agent's trade. One way to prevent this is to use more than one agent (possibly an agent per host), send each agent on a different route and combine the offers on the agent's return. Another way is to use one agent with a 'star' like route – it returns home after visiting each host before being sent out to a different host.

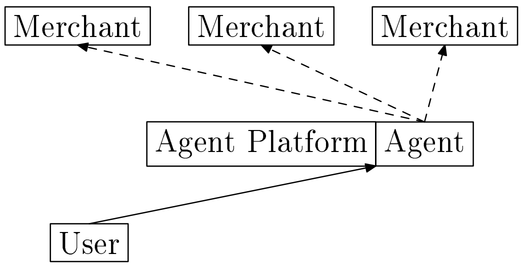

Figure 1. A model for agent platforms

6 A malicious host commits to purchases that the user does not wish to make.

This happens when a host can abuse the committal function that an agent has. A method to discourage this is to force the host to sign a transaction, as well as the user (thus providing traceability).

7 A malicious host denies the agent a service.

Here a host would stop an agent from moving further on its route. This of course could be traced if an agent reports when it arrives on a host.

8 A malicious host captures electronic money.

Here a host would remove the electronic money that an agent may have to purchase an item and either steal the money outright, or use it for a different purchase.

We do not consider the payment process here, as we are concerned only with the part of a transaction involved in selecting a merchant and committing to the transaction.

3. Models of Agent Platforms

Mobile agents roam between platforms. However, they can also communicate with each other, and with other hosts. This leads to the question as to the best “platform” model to use for trading (or indeed any other) agents. There are clearly two basic approaches which we now describe.

The first approach (see Figure 1) is to have a designated platform (or a collection of such platforms) to which we can send an agent to execute. This agent then communicates with merchant servers to seek information and commit to purchases.

The second model (see Figure 2) is to have an agent roam to each merchant server in turn and collect the information it requires. After collecting all the information the agent can then either return to the user to make the purchase, return to the chosen merchant to make the purchase or make the purchase from the final host.

In a mobile telecommunications environment it may also be beneficial to have a third model. This is where the requirements for a purchase are communicated to a 'home platform' (the user's home PC or a network operator controlled device) which then forms the agent and conforms to one of the above models.

In the above, any of the platforms may be malicious, with the possible exception of the home platform. The solutions proposed below can be made to fit into any of the above situations, although they both fit better into the first model.

The security risks associated with the above two models clearly differ. In the first case, the 'designated platform' might be trusted to keep secret certain agent information. An example of where this might be useful is when the agent contains details of the user 'expected' price (or maximum price), which it would be helpful not to reveal to the merchant. Of course, the threat then arises that one of the designated platforms will collude with one or more of the merchants. In the second case, it is clearly impossible to try and keep any information in the agent secret from the merchants. In both cases, however, as we will show in the remainder of this paper, there are potential benefits to be gained from the use of multiple agents, albeit not from the confidentiality perspective.

4. Model for a trading agent

We consider the information that an agent wishing to trade must know. Firstly, when initiating a purchase, a user will have a set of requirements (for instance the item to be purchased, the maximum price for that item, a time limit within which the purchase to be made). We

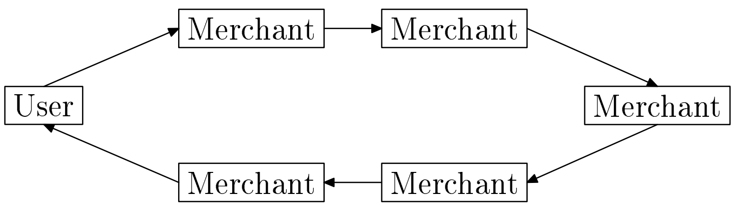

Figure 2. A second model for agent platforms

will assume that a user encodes these requirements into a string R which is understood by all parties. When a server quotes for a given purchase, it will also produce a similar string with its offer.

The agent, if it is to perform the purchase on behalf of the user, must also carry a function which will commit to the trade. This could be performed by, for instance, signing the details of the trade. One scheme to allow an agent to perform a signature operation on behalf of a user without revealing the user's private key to a host is proposed in [Kotzanikolaou et al., 2000]. In this scheme, using RSA, an agent carries both the hash value, h, of the requirements and the signed hash value, $h^d \bmod n$, where (d, n) is the user's private RSA key. To commit to a transaction for the user the agent calculates

$$(h^d)^x = h^{xd} = (h^x)^d \bmod n$$

effectively signing h^x where x is the server's offer. An alternative to this where the agent carries its own private key which the user certifies is given in [Borselius et al., 2001a].

Thus we assume that a trading agent will carry the following information:

- User Identifier – U
- Requirements for purchase – R
- A committal function – $\mathcal{C}$. The committal function is used by the agent to commit to a transaction on the user's behalf. $\mathcal{C}$ could be a signature function using a special private key provided to the agent by the user. Alternatively, $\mathcal{C}$ could be a function of the type described above, derived from the user's own private signature key. In any event, we assume that the function is designed so that only transactions within constraints defined by the user can be authorised.

Note that, if a single 'trading agent' is deployed there are a number of problems which might arise. Firstly, although the committal function will typically be limited to transactions conforming to user-defined parameters, there is still the possibility that the agent platform will force the agent to commit to a transaction which is less than optimal. It may also commit to more than one transaction, even if the user only intended to make at most one purchase.

One way to reduce this threat is to deploy multiple agents, a subset of which must agree to the transaction before it can be authorised. Such an approach is the focus of the remainder of this paper.

5. Threshold Scheme

We attempt to solve the malicious host problem by using multiple agents each of which has a 'vote.' If one of the possible transactions receives enough votes, then a transaction will be authorised with the relevant merchant. We begin by outlining the scheme, and then consider the details of what a secure vote can consist. We assume use of a (k, n) scheme – *i.e.* a server will need k votes out of a possible n to 'win'.

5.1. The Scheme

Let $T = \{T_1, T_2, \ldots, T_n\}$ be a set of agent platforms. The user then sets up a (k, n) voting scheme with shares $s_1, s_2, \ldots, s_n$. Clearly k should exceed the number of 'suspected' malicious hosts. Given no information about the system a sensible value would probably be $n/2 + 1$. The value of k reflects the level of trust in the system.

The user then forms n agents A_i $(1 \leq i \leq n)$ containing the following information

- User Identifier – U
- Requirements for purchase – R
- A vote – s_i

Each agent is then dispatched to its agent platform. At the platform there are two modes of execution:

1 The agent contacts each merchant itself, and gathers bids that meet the requirements.

2 The agent contacts a subset of the merchants and communicates the best bid to its peers.

We note that case (2), unless only contacting a single host, is a situation that must be carefully thought out. This is because if there is no overlap in the merchants, collusion may mean that attacking less than k servers is necessary.

When each agent has received all the information about each bid, the agent sends its vote to the merchant with the best offer. On receipt of the correct number of votes, the merchant or a nominated third party can construct (and verify) from the votes the authorisation for the bid. The merchant or nominated third party can then use this as evidence that the user has committed to transaction.

We now consider the security of the above scheme. The major advantage of the scheme is the need to corrupt either $n - k + 1$ agents to prevent the transaction or k hosts to divert or alter the transaction. Thus the choice of k is crucial.

This also means that a denial of service attack is harder as a server or set of colluding servers will need to terminate (or prevent from communicating their vote) $(n - k) + 1$ agents. Again to force a purchase a host or hosts must force k agents to offer their vote.

If an agent visits a subset of the servers involved, the information could then be used to help identify any malicious hosts.

5.2. The Votes

As mentioned above, the votes can be assembled by either the selected merchant or a nominated third party. Note that there are clear risks associated with giving votes to the merchant, since the merchant could now possibly commit the user to a transaction of the merchant's choice (within any constraints imposed by the string R). That is, the merchant is not forced to commit to the transaction as offered to the agents. Hence the use of a nominated third party to reconstruct the votes is the preferred approach. The possibility that this may not be feasible in practise leads to an alternative approach.

One approach is *threshold cryptography.* Threshold cryptography was first proposed by Desmedt [Desmedt, 1988]. A typical example of a threshold cryptosystem is one that would allow a set of t parties to sign any document such that any coalition of less than t parties cannot sign any other document. Schemes tend to rely on a combiner which does not necessarily need to be trusted. Schemes based on both RSA and El Gamal have been proposed.

Recently Shoup [Shoup, 2000] proposed an RSA scheme which is as efficient as possible; the scheme uses only one level of secret sharing, each server sends a single part signature to a combiner and must do work that is equivalent, up to a constant factor, to computing a single RSA signature. Although not perfect as a threshold signature scheme (as it relies on a trusted party to form the shares) this scheme is ideal in our setting. (Note that an alternative scheme without a trusted dealer is given in [Damgård and Koprowski, 2001]. This scheme also improves on Shoup's scheme by not relying on an RSA modulus made up of 'safe primes'). An example of an El Gamal scheme is given in [Langford, 1995]. We note that a (n, n) threshold signature scheme is just a multisignature; such schemes have been studied for many years — see, for example, page 488 of [Menezes et al., 1996].

We note, however, that such a threshold signature scheme does not provide a means for the shares to incorporate an encoding of the string

R. Thus, if there were k colluding hosts they could sign (and reconstruct a signature) for any document. One solution to this problem is for the user to generate a special signature key pair for the particular purchase (i.e. for this particular set of agents), and then to generate a certificate for the public key incorporating a copy of R. When the signature is reconstructed from the signature shares, it can be verified using this certificate. However, it is possible to merge the undetachable signature scheme given in [Kotzanikolaou et al., 2000] with the threshold signature scheme of Shoup [Shoup, 2000] and details of this are given in [Borselius et al., 2001b].

6. Using one trusted host

We consider a second solution to the problem, which employs a single trusted host. We note that the solution described below involves a user sending out agent(s) to individual merchant servers, whereas it could just communicate with them to ask for their bids. However, in a wireless communications setting where communication is expensive, slow, and/or unreliable, it is believed to be beneficial to be able to dispatch an agent into the fixed network. When the agent has finished its task it contacts the user or waits for the user to collect the result.

Let $S = \{S_1, S_2, \ldots, S_n\}$ be a collection of servers offering a service that a user wishes to purchase. Let T be a host that the user trusts to act honestly in this transaction. (Note that we do not need to trust this host fully – it just needs to be neutral in this transaction). Before the transaction commences we assume that each server S_i and T securely establishes a shared secret key K_i. Optionally, a key for message integrity checks could also be established.

The user despatches an agent A to the trusted host containing the information outlined in §4. We note that the committal function $\mathcal{C}$ may be of any form with which the user is prepared to trust the host T. However, to reduce the trust requirements we envisage that this will be the scheme outlined in either [Borselius et al., 2001a] or [Kotzanikolaou et al., 2000].

There are now several approaches for T. The first is to form a single subagent containing the following information

- Agent identifier – I
- Requirements for purchase – R
- Host identifier – T

which would then visit each of the servers in S in turn. We note that the requirements sent out do not need to include pricing information (that is the maximum price the user is prepared to pay) or any other information that the user wishes to be used to help make the decision, but does not wish to communicate to the server. Another approach is to form a single agent for each server. A third approach has the above agent visiting a subset $S' \subset S$ of the above servers. Whichever strategy is employed, at each host the agent performs the following actions:

1 Find out the server's bid B_i for the item specified in the requirements R.

2 Encrypts the concatenation of B_i, R, S_i and I using either K_i. At this point the server could also, optionally, attach a symmetric MAC (Message Authentication Code) to the bid to protect the integrity of the server's bid. Label the encrypted string E_i.

3 The agent then stores the pair (S_i, E_i).

The agent returns to T when it has finished visiting all of its servers. The agent on T then decides the best offer and commits to it using the committal function.

We note some of the features of the above scheme.

- Using an agent per server really alleviates the need to encrypt anything, assuming that agents are always transferred between hosts in encrypted form.
- Using a single agent leaves yourself open to some attacks.
- Using more than one agent that does not visit all the hosts could then be used to (help) identify a malicious host.

If we use a single agent and it visits all the hosts, or we have an agent that visits more than one host, the agent is subject to the following attacks:

- An approach to enable a malicious host to underbid its competitors, is as follows. The host forms a new agent containing the user's requirements, a fictitious user identifier, and its own host identifier. This agent would then traverse the route of the user's agent, and discover the bids offered for that set of requirements. The host could then under bid its competitors, but the user's agent would have had to have been kept on the malicious host in the interim period. Thus monitoring the progress of an agent could help determine if such an attack was being used.

- A simple denial of service attack: stop the agent in its tracks. If there is no progress monitoring (*e.g.* agent at host S_i) then this attack is hard to defeat.
- A malicious host could alter the pair (S_i, E_i) to read $(S_j, junk)$ (where *junk* is a random string of the correct length) to stop the decryption of a bid. However as the host cannot read the bid, for this to be successful (*i.e.* to delete those bids more attractive than those of the malicious host) the host would have to have knowledge of all the bids – which it would have to gather itself (possibly by cloning the agent).

If we use an agent that visits a subset of the hosts, and assume that the malicious host already knows the "best offer" at any given point, it will then try to undercut this (this undercut is a lie). If we then require, we can apply rules to the results of the other agents, and attempt to identify the malicious host. This also requires careful choice of agent destinations and routing.

Note that to force T to purchase from a malicious host, the host has to lie and then be unscrupulous, or just lie and possibly not profit as much as it would expect. That is if the malicious host M wants to force a user to trade with it, then it must have the best price. So it must either charge more than its advertised price (possibly breaking the committal function) or make less profit than it expects (because the price advertised is less than the host should sell for).

We now consider the extent to which the user must trust the host T. The user must trust that T does not favour a particular server for this transaction. However, with a sufficiently good committal function then this is the only trust requirement. For example using the Kotzanikolaou et al. undetachable signature scheme [Kotzanikolaou et al., 2000], as a committal function, T can be given the means to commit to the transaction without being trusted with a copy of the user's private signature key. This may be a situation where using an undetachable signature scheme has advantages over the creation of a separate signature key for each agent.

7. Conclusions

We have considered two different ways in which the deployment of multiple agents can reduce the threat to trading agents from potentially malicious agent platforms. In the first approach multiple agents are equipped with 'shares' of the means to commit to a transaction. A method implementing this idea using a threshold signature scheme, e.g. the recently proposed scheme of Shoup, [Shoup, 2000], was outlined. In the second approach a single trusted host is employed to collect information from multiple agents on possible transactions. This host then chooses the optimal transaction and commits to it.

The two approaches each have their own advantages. The first approach avoids the need for a single trusted host. However, implementing the first approach requires use of some potentially complex cryptographic signature functions. The second approach is potentially less complex from a cryptographic perspective, but does require a host which, if not completely trusted, is at least required to act neutrally with respect to the set of merchants. Both approaches are of potential practical importance in future mobile computing environments.

References

[Borselius et al., 2001a] Borselius, N., Mitchell, C. J., and Wilson, A. (2001a). A pragmatic alternative to undetachable signatures. Preprint.

[Borselius et al., 2001b] Borselius, N., Mitchell, C. J., and Wilson, A. (2001b). Undetachable threshold signatures. To be presented at the IMA Conference on Cryptography and Coding, December 2001 (proceedings to be published in the Springer-Verlag LNCS series).

[Damgård and Koprowski, 2001] Damgård, I. and Koprowski, M. (2001). Practical threshold RSA signatures without a trusted dealer. In Pfitzmann, B., editor, *Advances in Cryptology - EUROCRYPT 2001*, number 2045 in LNCS, pages 152–165. Springer-Verlag, Berlin.

[Desmedt, 1988] Desmedt, Y. (1988). Society and group oriented cryptography. In Pomerance, C., editor, *Advances in Cryptology – Crypto '87 proceedings*, number 293 in LNCS, pages 120–127. Springer-Verlag, Berlin.

[Farmer et al., 1996] Farmer, W., Guttmann, J., and Swarup, V. (1996). Security for mobile agents: Authentication and state appraisal. In *Proceedings of the European Symposium on Research in Computer Security (ESORICS)*, number 1146 in LNCS, pages 118–130. Springer-Verlag, Berlin.

[Hassler, 2000] Hassler, V. (2000). *Security Fundamentals for E-commerce.* Artech House.

[Hohl, 1998a] Hohl, F. (1998a). A model of attacks of malicious hosts against mobile agents. In *Proceedings of the ECOOP Workshop on Distributed Object Security and 4th Workshop on Mobile Object Systems: Secure Internet Mobile Computations*, pages 105–120.

[Hohl, 1998b] Hohl, F. (1998b). Time limited blackbox security: Protecting mobile agents from malicious hosts. In Vigna, G., editor, *Mobile Agents and Security*, number 1419 in LNCS, pages 92–113. Springer-Verlag, Berlin.

[Kotzanikolaou et al., 2000] Kotzanikolaou, P., Burmester, M., and Chrissikopoulos, V. (2000). Secure transactions with mobile agents in hostile environments. In Dawson, E., Clark, A., and Boyd, C., editors, *Information Security and Privacy, Proceedings of the 5th Australasian Conference ACISP 2000*, number 1841 in LNCS, pages 289–297. Springer-Verlag, Berlin.

[Langford, 1995] Langford, S. K. (1995). Threshold DSS signatures without a trusted party. In Coppersmith, D., editor, *Advances in Cryptology – Crypto '95 proceedings*, number 963 in LNCS, pages 397–409. Springer-Verlag, Berlin.

[Menezes et al., 1996] Menezes, A., van Oorschot, P., and Vanstone, S. (1996). *Handbook of Applied Cryptography*. Discrete Mathematics and Its Applications. CRC Press. Available on-line at http://www.cacr.math.uwaterloo.ca/hac.

[Ng, 2000] Ng, S.-K. (2000). Protecting mobile agents against malicious hosts. Master's thesis, The Chinese University of Hong Kong.

[Rasmusson and Jansson, 1996] Rasmusson, L. and Jansson, S. (1996). Simulated social control for secure internet commerce. In *New Security Paradigms '96*, pages 18–26. ACM Press.

[Riordan and Schneier, 1998] Riordan, J. and Schneier, B. (1998). Environmental key generation towards clueless agents. In Vigna, G., editor, *Mobile Agents and Security*, volume 1419 of *LNCS*, pages 15–24. Springer-Verlag, Berlin.

[Sander and Tschudin, 1997] Sander, T. and Tschudin, C. (1997). Towards mobile cryptography. Technical Report 97-049, International Computer Science Institute, Berkeley. Available at http://ww.icsi.berkeley.edu/ sander/publications/tr-97-049.ps.

[Sander and Tschudin, 1998] Sander, T. and Tschudin, C. (1998). Protecting mobile agents against malicious hosts. In Vigna, G., editor, *Mobile Agents and Security*, number 1419 in LNCS, pages 44–60. Springer-Verlag, Berlin. Available from http://www.icsi.berkley.edu/ sander/publications/MA-protect.ps.

[Shoup, 2000] Shoup, V. (2000). Practical threshold signatures. In Preneel, B., editor, *Proceedings of EuroCrypt 2000*, number 1807 in LNCS, pages 207 –220. Springer-Verlag, Berlin.

[Vigna, 1997] Vigna, G. (1997). Protecting mobile agents through tracing. In *Proceedings of the Third ECOOP Workshop on Operating System support for Mobile Object Systems.*

[Wilhelm et al., 1998] Wilhelm, U. G., Staamann, S., and Buttyán, L. (1998). On the problem of trust in mobile agent systems. Available from http://www.isoc.org/isoc/conferences/ndss/98/ndss98.htm. Network and Distributed System Security (NDSS'98) Symposium.

[Yee, 1997] Yee, B. (1997). A sanctuary for mobile agents. In *DARPA Workshop on Foundations for Secure Mobile code.* Available from http://www.cs.nps.navy.mil/research/languages/statemensts/bsy.ps.