Mobile Agents and Telcos' Nightmares[*]

Joachim Posegga
Deutsche Telekom AG
Technologiezentrum
IT Security / FE34a
D-64276 Darmstadt

Tel. +49-6151 83-6715
Posegga@TZD.Telekom.DE

Günter Karjoth
IBM Research
IBM Zurich Research Laboratory
Säumerstr. 4
CH-8803 Rüschlikon

Tel. +41-1 7248 486
gka@zurich.ibm.com

November 2, 1999

Abstract

The paper analyses the current state-of-the-art of mobile agents technology wrt security, seen from the standpoint of a public network operator (PNO).

It is argued that the current state-of-the-art does not offer sufficient security for large-scale, commercial applications of mobile agents within the PNO's networks. To support this thesis, the most important security issues in this context are discussed, and a number of deficiencies are identified. Some of these deficiencies pose principal questions for future research that are not necessarily widely accepted by the agent community.

Keywords: mobile agents, security, cryptography, telecommuncation

[*]The opinions expressed in this paper are solely those of the authors, and do not necessarily reflect the views of their respective employers.

Contents

1 Introduction

Intelligent agents have become fashionable again in the late nineties, not only within the artificial intelligence community and in the popular press but also in traditional computer science communities. Whereas current commercial solutions for distributed processing are mostly centered on the client-server concept, with remote procedure calls as the standard way of interaction, script languages have emerged that explicitly support agent-oriented programming — in that they make available easy-to-use programming constructs to build autonomous objects that not only may accumulate knowledge and perform reasoning, but may also migrate between host machines by their own decision.

This paper will focus on the "mobility" attribute of intelligent agents and thus uses the term *mobile agent* to denote an autonomous mobile object to which a user delegates some or all of her decision-making in the respective problem domain, and that is able to move around an electronic network, communicating with other such active objects by means of message passing. Mobile agents also embrace the "smart network" approach where control scripts are dynamically downloaded and / or exchanged.

Nwana (Nwana, 1996) argues that mobile agents provide a number of *practical*, though not *non-functional*, advantages, such as reduced communication cost, limited local resources, easier coordination, and asynchronous communication. He sees mobile agents to provide a natural development environment for implementing "free market" trading services; new services can come and go dynamically and much more flexible services may co-exist with inferior ones, providing more choices for consumers.

As the trend to an open "electronic market" of services continues, where a wide spectrum of communication and information services will be offered, ranging from simple communication services up to complex distributed multimedia applications, and where the instant provision of new services and the customization of existing services become critical issues, mobile agents will become to an increasing degree an essential aspect in the design of new telecommunication networks to handle the ever-increasing complexity (Magedanz *et al.*, 1996).

Agents have been early employed in the construction of "smart" networks, routing or filtering messages sent to a user and seeking information or services on the user's behalf (Reinhardt, 1994). Among the numerous issues concerning service integration, Plu sees mobile agents used in telecommunication companies to deal with the following problem areas (Plu, 1998):

- multiple authority cooperation,
- interoperation with heterogeneous information managed by heterogeneous software,
- physical distribution of computation.

Other areas of mobile agent employment are active networks: programmable switches, active network management; and the push / pull model (Tennenhouse *et al.*, 1997; Bieszczad *et al.*, 1998) (Alexander *et al.*, 1997; Alexander *et al.*, 1999b; Alexander *et al.*, 1999a). A collection of papers can be found in (Jennings & Wooldridge, 1998; Hayzelden & Bigham, 1999).

routing: (Schoonderwoerd *et al.*, 1997; Bonabeau *et al.*, 1998; Di Caro & Dorigo, 1998; Minar *et al.*, 1999; Kramer *et al.*, 1999)

The evolution of networks is in the direction of large, high-speed networks, consisting of diverse equipment and supporting mobility. Furthermore, there is clearly a trend towards more and more reliable network-centric services. Given such an evolution, the centralized network management paradigm is no longer sufficient to handle management tasks effectively.

The capability to dynamically place control and management software processes at the most appropriate location will have significant impact on the architecture and the related (signaling) protocols of telecommunication systems (Magedanz *et al.*, 1996). The ability of agents to handle heterogeneous environments, the potentially adjustable (small) footprint of an agent, and the potential to dispatch and activate agents as necessary during problem resolution are properties useful for network management. Telecom companies, and in particular France Telecom, are building infrastructures to allow software agents to provide more and more sophisticated services to multiple users (Plu, 1998). This strong interest resulted also into an activity to to standardize agent systems: the Foundation for Intelligent Physical Agents (FIPA). In a different setting, the Object Management Group has recently published its Mobile Agent System Interoperability Facility (MASIF) specification.

(Appleby & Steward, 1994)

The number of workshops and conferences in 1999 devoted to mobile agents and telecommunications identify agent technology also indicate that mobile agents to be an interesting research topic.

IATA'99,IWAN'99,MATA'9

All agent-based systems, impressive as their potential benefits may rightly seem, may, however, be jeopardized in a commercial world by the lack of appropriate security mechanisms. Indeed, most agent systems do not address error recovery and auditing, privacy, and various aspects of trust among the participants.

(Baldi & Picco, 1998)

Network safety, security, and reliability are crucial issues for a Public Network Operator (PNO). Mobile agents are certainly a potential security threat. PNOs must therefore critically evaluate agent-based scenarios before actually thinking about introducing them for commercial applications. Although many promoters of mobile agent technology acknowledge that security represents a key factor for the successful employment, they tend to ignore it in their work:

> Security against mobile agents is a very big issue or a red herring depending on what one is trying to do (Virdhagriswaran, 1997).

Whereas a first generation of mobile agents has been written in languages such as Telescript and Safe Tcl/Tk, the omni-presence of Java, mainly due to its platform-independent executables fetchable from the network, makes it a prime platform for the implementation of agent systems. Aglets (IBM) (Lange & Oshima, 1998), Concordia (Mitsubishi) (Castillo *et al.*, 1998), Odyssey (General Magic), and Voyager (Open Space), all developed within industry, are examples of recent Java-based agent languages/environments. This means that on the one hand agents have potential access to all Java class files on the host;

on the other hand they rely on the security of the Java interpreter for their proper execution. Thus, agent security and Java security go hand in hand. All the security concerns raised about Java (see for example (McGraw & Felten, 1998)) also affect the safe execution of agents. A small, local bug in the implementation of the hosting Java interpreter will affect the security of the entire agent system.

Even more problematic is the fact that there are also threats that originate in malicious hosts. For example, if there is no mechanism to prevent attacks, a host can implant its own tasks into an agent or modify the agent's state. This can lead in turn to theft of the agent's resources if it has to pay for the execution of tasks or to loss of the agent's reputation if its state changes from one host to another in ways that alter its behavior in negative ways. Taking together, mobile agent technology faces a number of barriers that hinder penetration into industrial practice.

This paper argues that there are still too many unsolved problems with respect to security, safety, and reliability in agent-based systems; it is therefore too risky for PNOs to apply these techniques today for building commercial application within public networks.

Note that the aim of this paper is *not* to discuss the idea underlying mobile agents in general: only issues related to safety, security, and reliability are considered. For an introduction into agents, we refer to (Nwana, 1996; Rothermel & Popescu-Zeletin, 1997). For the uninitiated reader, we refer to the literature on security in computer networks, e.g. (Ford, 1994; Kaufman *et al.*, 1995), that covers the material required to appreciate the present discussion.

In Section 2, we discuss why security and reliability are of utmost importance to any public network operator. Section 3 elaborates on three categories of agent-related security problems: Section 3.1 looks into the protection of the host, Section 3.2 changes the view and looks into the protection of the agent, and Section 3.3 looks into a list of issues that relate to the proper functioning of the underlying network, but are not related to security in the strict sense as discussed before. Based on the discussion before, we elaborate in Section 4 on potential protection mechanisms usable within restricted trust models, and then conclude.

2 Security, the Crucial Point in Public Networks

Public network operators tend to be rather paranoid about network security issues. This is for very good reasons: a telephone network can be seen as an enormously large distributed system which represents the PNOs' capital. While banks use networks for working with money, the network *is* the money for PNOs.

What makes Telecommunications networks so attractive for intruders is the amount of money that these networks represent: in 1996, the operating revenues of the world's three major Telecoms Operators (AT&T, NTT, and Deutsche Telekom) summed up to over 200 billion US-$. It is obvious that such networks need to be strictly protected.

While in the early days Telecommunications networks basically consisted of wires and mechanical switches, today's systems carry out quite complex functions which are implemented mostly in the form of software. Thus, PNOs are becoming more and more confronted with the challenge of managing a software-based, distributed system of growing complexity.

To make things even worse, this system has a number of very nasty properties:

It is **highly distributed and heterogeneous:** since Telecommunications networks work world-wide, a large number of operators have to interact; many of them use different equipment and transmission techniques (standard cable, fiber optics, radio communication, satellites, etc.).

It is **safety critical,** since our society *depends* on the proper functioning of these networks. The most efficient way to paralyze an industrial country is probably to interrupt its information infrastructure. Even short-time breakdowns can be disastrous, for instance when failing to process an emergency call.

It must be **continuously running** since Telecommunications networks cannot be halted for examination, debugging, or rebooting.

It must be **fault-tolerant** since faults and malfunctions within the system or at the customer's site will always happen; such events, however, must not cause the whole network to crash.

It also must meet **real-time requirements** because setting up a (world-wide) call cannot take arbitrarily long.

Setting up and running such a network is a true technical challenge; it actually is getting harder and harder, since the complexity of services is constantly growing: more and more, mostly software-based features and services are being added—in parallel, the problems involved with running those systems increase.

PNOs are already paying huge amounts of money for coping with this situation, and it looks like things will be getting worse. There is some hope that better software engineering through increased application of formal methods might bring some relief, but these techniques are just moving out of research. So, plenty of problems already exist, and new technologies have to be critically examined before they can be applied.

3 Agents, just Another Nightmare?

Telecommunications networks are certainly attractive for applying mobile agent technology: While the Internet still ponders the question of how to work with real money in the Net, this has been done for years in Telecommunications networks. Thus, commercial applications of mobile agents could exploit an existing billing procedure in Telco networks.

A scenario with mobile agents unleashed in Telecommunications networks, however, brings new problems for network security. In order to precisely analyze these problems, one must first clarify the concrete scenario to be considered:

As long as Telecommunications networks served just as a transport medium, one could argue that it did not matter too much what was transported. Even then, PNOs probably had an obligation to protect their customers. But such a clear separation between agents scenarios and Telecommunications networks is naive, since technology is moving in the opposite direction: Most PNOs are becoming involved into the Internet business, and Telecommunications networks themselves will probably merge with the Internet in the future. By "merging", accessing mutual functionality is meant: services of Telecommunications networks and Internet services will interoperate. When considering a possible agent scenario, we can assume that PNOs become more involved than with simply providing the underlying transportation platform.

Thus, PNOs must be concerned about agent security when considering the introduction of such a technology and opening up Telecommunications networks to agents: as any other commercial company, PNOs will only support a new technology if the potential benefits outweigh the potential risks. In the end, this decision will be based on financial considerations: among other things, the potential damage to security weaknesses and their likeliness is an important criterion.

The problems that arise in an agent-scenario can be divided into three categories:

- *Protecting hosts in a network,*
- *protecting the agents,* and
- *protecting the network itself.*

These three fields, which are actually not as clearly separated as might appear, will be discussed in the sequel.

Interestingly, research in these three areas is at very different stages: while the problem of protecting hosts from agents seems well recognized, less work has been carried out on protecting agents from their hosts. Barely any approaches at all cover issues of controlling the overall behavior of a network with mobile agents. A number of surveys about the state-of-the-art can be found in (Moore, 1998; Greenberg *et al.*, 1998; Tschudin, 1999; Jansen & Karygiannis, 1999); a collection of security papers are contained in (Vigna, 1998; Vitek & Jensen, 1999).

3.1 The nightmare of host attacks

The most important law of system security reads *Never, ever run code from untrusted sources.* In most companies this means that employees are not allowed to use software that did not come through the standard channels, e.g. the purchase department.

A number of new technologies are already contradicting to this principle, for instance Java and ActiveX. Whilst with these still leave some control over incoming code, in the sense that they have at least to be actively downloaded, agents can even move by themselves. An agent-based scenario must therefore provide a very convincing answer to the question: *How can I protect myself from malicious agents?*

One attempt for such an answer, currently the most popular one, is the sandbox approach that is, for instance, followed by Java (McGraw & Felten, 1997).

3.1.1 The Sandbox Placebo against Host Attacks

The idea is rather simple: an agent cannot control a machine, if it runs in a "sandbox" that blocks access to the outside (ie: the real machine): the sandbox offers as much functionality to a program running inside as one wants to grant.

Several approaches like Java (Fritzinger & Mueller, 1996; Yellin, 1995), Telescript (Tardo & Valente, 1996) or SafeTcl (Ousterhout *et al.*, 1998) follow this idea, all in somewhat different technical details. Since Java is certainly the most popular of these today, Java's security concept will be considered in more detail. It does not matter much in this context that Java is currently not actively supporting agents, since it makes not much of a difference for the sandbox approach.

Java's security concept can be divided into four layers: the *language*, the *byte code verifier*, the *class loader* and the *interface-specific features*. The overall idea behind this approach seems reasonable, but details are still unsatisfactory:

The language Java is claimed to be a safe language, where this basically means that the language is safely typed and that the host system's memory is protected. Noteworthy, the programming language Java does not matter much for the security aspects considered here: it is the Java byte code that is transferred.

Noteworthy, the level of security on this layer crucially depends on Java's type system. However, this type system is still under formal studies (cf. for instance (Alves-Foss, 1999)). Although the type system for a larger part of Java has been shown to be sound, it is still to early to draw strong conclusions about the actual security offered. As the work continues and the descriptions of the type system grow, applications of theorem provers seem to be necessary to deal with the complexity in a satisfactory way. Thus, until final results are achieved we can at most *hope* (though based on some evidence) that the language is as safe as claimed.

The byte code verifier is supposed to analyze incoming applets to ensure that they comply with a couple of safety criteria (see (Yellin, 1995) for details).

> The first question a person with some background in formal methods will ask is *If it is a verifier, then where is the specification?* Verification as an isolated notion does (in the formal sense) not make sense: one can only verify something *against* some other thing. A program can be verified against some specification, provided we have formal semantics for both at hand.
>
> Neither of these exist today in the case of the Java byte code "verifier", so again no strong conclusions about properties of an applet that passed this process can be drawn.[1] The provision of a formal model for bytecode verification is also a topic of current research (see for example (Posegga & Vogt, 1998; Qian, 1999)). It will take some time to influence the different commercial bytecode verifiers on the market. Until this happens we even have the risk that our agent might not be accepted by hosts employing a different bytecode verifier due to different interpretations of the required safety properties.
>
> The Java language and the Java virtual machine are two different beasts. While the Java and JVM type systems are related, they are not the same. Since JVM programs, e.g. applets, are the medium of providing programs for execution, their semantics should be stated at the level of the JVM. The question whether a given JVM program preserves the behavior of a given Java program is secondary for the security of hosting platform.

Similar observations can be made for the two other security layers: the **class loader**, although rather simple in principle, is not formally described, neither are the **interface-specific features** offered by the sandbox.

What do you mean by "i/f-specific features?

Overall, the situation with one of the most prominent sandbox approaches today, the Java virtual machine, is not very satisfactory: there are certainly some good ideas, but it remains somewhat obscure what these precisely do and how they interact. Furthermore, Java fails completely to protect against denial of service attacks.

From a security's point of view, holes in a vaguely rather than rigorously specified security concept like Java's are not surprising. An impressive number of such security holes have in fact already been detected (Sun Microsystems, n.d.; SIP, n.d.). Most of these bugs are serious threats to system security; none of them is tolerable.

Any comment on the latest findings of the Marburg student?

It is very interesting to consider Sun Microsystems' reaction to these bugs (Hamilton, 1996):

> *The security bugs found were the result of implementation bugs, not errors in Java's underlying security model.*

The question that remains open is to what extend this distinction matters to a victim of such bugs, or to a victim of future bugs that can certainly be expected

[1]A note to the interested reader: what the byte code verifier actually performs is – besides some syntactical checks – something like a fixed-point induction to ensure that the operand stack does not overflow during run time. This is the operation that is closest to verification.

to show up. Such an attitude wrt system security might be acceptable in a low-risk domains like university networks; in commercial setting, where attackers might exploit such bugs to cause significant damage, it is not acceptable.

However, it has to be recognized that Sun did take one very wise decision with making the source code of their Java implementation open to all. This is why most of the bugs have been detected, and this openness guarantees a certain degree of security by itself–at least for those who constantly follow the relevant information resources.

3.1.2 The Sandbox Approach – *Instruction Leaflet*

Sandbox approaches aim at protecting a host from foreign code by setting up a restricted execution environment for the incoming code. Although this always trades functionality for security, it is a good approach in principle.

It is important that such a sandbox comes with a clear security model and a security policy. This is indispensable for actually judging the risk that is involved with providing the sandbox. The most popular approach today, the Java virtual machine, is not (yet) sufficiently formal; see (Alves-Foss, 1999) for a variety of approaches to formal specifications, execution models, and analysis of Java programs.

Alternative approaches to Java (Telescript (Tardo & Valente, 1996) or SafeTcl (Ousterhout *et al.*, 1998)) follow similar principles and are in some respects more elaborated. However, none of the present approaches currently fulfills strong security and safety requirements. While this might be acceptable for settings where the potential damage by abuse is low, it seems not acceptable for use within a Telecommunications network.

3.1.3 Cryptography for Agents-Placebo

Cryptography, based on well-founded mathematics, can for instance be used for authentication, digital signatures, and encryption in the context of agents. For protecting a host while executing remote code, authentication and digital signatures seem suitable for guaranteeing that the mobile code has not been modified somewhere on its previous “hops”, and for ensuring that the code actually comes from where it is thought to come from.

Before discussing this in more detail, it is important to explicitly point out what authentication actually offers, or – more importantly – what it does *not* offer:

Authentication and digital signatures can, if applicable and applied correctly, provide very strong evidence that what is arriving is exactly what has been sent by the authenticating party. It *cannot guarantee this,* and it *does not give any information about the actual contents* of what is arriving. Especially the latter seems important, since properly signed agents might still be intruders, as recent experience with ActiveX nicely demonstrates (Neumann, 1997a; Neumann, 1997b; Digicrime, n.d.). Note, that even “good-natured” code from a trusted source might contain security holes that could be exploited by third parties. Thus, authentication and digital signatures can *per se* not guarantee that a particular agent is harmless.

Even with these principal limitations, there are problems with authentication in the context of agents (see (Chess, 1996; Ordille, 1996) for a more detailed discussion):

Firstly, cryptography suffers from a principle problem when applied to agent scenarios: I might trust the agent's creator, but how do I know that an intermediate host did not tamper with the agent?

As we will see in more detail in Nightmare II, there is no known procedure to guarantee integrity of agents by cryptographic means if agents carry variable data (for instance state) to be used while traveling. This, however, seems indispensable for agents. But then, deciding the level of trust for an incoming agent requires to consider all the nodes that have been visited by this agent (Ordille, 1996).

Second, even if we had an applicable procedure, authentication in a scenario where many parties can send out agents had to assume that trust is *scalable* and *transitive*.

- *Scalability* is required because a large number of agents and hosts these can visit is what makes agent-based systems just attractive. However, all these nodes and agents have to be involved in a trust-based concept. This is problematic in large-scale networks since, obviously, trust is never a 100% concept, but there is always some risk involved. But the more people (systems? agents?) you trust, the more overall risk you have.
- *Transitivity* of trust is required, because one would not want to define the set of visited hosts for an agent in advance. Unfortunately, the longer the chain of trust is, the more the risk grows in practice. (Remember that the risk is defined by the actual content of an agent, respectively by what it will do on a host. You might pass a signed, blank check to your best friend, but would you also agree to have passed it to the best friend of your best friend's ... best friend?)

  There are a number of people working on trust management: Feigenbaum, Minar, Jøsang, Maurer.

3.1.4 Cryptography – *Instruction Leaflet*

Overall, it is to be observed that cryptographic techniques cannot tell anything about what the agent will actually try to perform: even if one runs only agents from trusted sources, these agents might be a security hole, or contain one.

Cryptographic techniques like digital signatures to prevent forging seem hardly compatible with mobile agents, since agents are usually supposed to change while traveling in a network. Otherwise, agent-based scenarios make sense only for very limited applications. In this case, however, digital signatures fail for being applied to this variable data, and trust depends upon the agents source *and* all hosts visited so far. (This is explained in more detail in Nightmare II, below.)

3.2 The Nightmare of attacked agents

A point that is often overlooked in the literature is the need for protecting an agent from the host it visits. Without an appropriate concept for this, a commercial application based on agents would have serious problems:

Assume one sends out an agent searching a particular product for a best bid with some upper limit. It would, in principle, be possible that some host where the agent comes along reads this information out and manipulates the agent such that it not looks further and reports back that there is only one vendor, namely the one controlling the manipulating site. This vendor could then offer the product for just a little bit below the limit.

Such a possibility is certainly not acceptable. Even worse scenarios are possible if agents had themselves some authority, for instance by being able to spend money.

Note, that protecting agents is also closely connected to protecting network hosts to be visited by agents: if such host is supposed to run an agent, it will want to make sure that the mobile code has not been manipulated to try to abuse the host – unless the host has a foolproof runtime system for agents, which will hardly ever exist.

3.2.1 The Placebo of Cryptographically Protecting Agents

At first sight digital signatures, authentication, and encryption schemes, particularly those based on public keys, seem like what is needed for protecting agents against manipulation. Unfortunately, such techniques sort of contradict to the spirit of agents (Ordille, 1996):

The agent itself cannot be encrypted, since it must run on a host and must therefore be readable (or decryptable) for the host. Moreover, applying a digital signature to an agent is quite pointless, since most scenarios will require that the agent changes its state while roaming. All authentication is always relative to the current host.

The source of the problem is that an agent must instruct the current host to perform any operations, rather than being able to act on itself; therefore, an agent cannot carry any secrets on its journey, since, firstly, the agent must be readable to the host, and second, for using such a secret it had to be passed to the run time environment of a visited host. This has two major consequences:

1. An agent can only encrypt information gathered during its trip with public keys, therefore any encrypted information is unreadable for the agent until it arrives at a trusted place with the secret key for decrypting it again.

2. Any signature that is applied to an agent or its data has only as much trust as the host where it was created.

This results in the very unpleasant situation that determining the trust to be put into an incoming agent requires to look at its complete travel history.

There are, however, some scenarios where cryptography is at least of some use in agent-based systems: if an agent is separated into a program part and a data part[2], it is at least possible to sign the program part. However, it is questionable how much this helps, since the data gathered while roaming is probably what the agent has been sent out for. Furthermore, the collected data can also be expected to have an influence on the agent's behavior.

There are approaches that propose to control an agents behavior by voting techniques (Minsky *et al.*, 1996), which offer a solution to the problem to some extend. These approaches, however, tolerate individual, malfunctioning agents which is somewhat unsatisfactory.

3.2.2 Cryptography for Protecting Agents – *Instruction Leaflet*

Unfortunately, cryptography seems to be somewhat incompatible with the idea behind agents, and currently fails to offer satisfactory solutions. This is particularly problematic, since non-experts often tend to trust blindly into cryptography, and take security of such approaches for granted.

It is interesting to investigate the underlying reasons for the "incompatibility" between cryptography and agents, since it uncovers a common misunderstanding about agents:

The notion "mobile agent" suggests an autonomously acting individual. This, however, is an illusion in the case of software agents:

- Firstly, an agent cannot act autonomously: it does not have its own processor, but the visited host performs all actions "on behalf" of the agent. Thus, sending an agent to a host is in principle equivalent to asking the host to carry out certain actions.
- Second, an agent is not an individual, in the sense that it is separated from its environment: to the visited host, all details and internals of an incoming agent are totally visible. In a certain sense, the agent and the host "merge" when the agent is executed: the agent *becomes* the host and vice versa.

A consequence of this is that agents cannot carry any secrets, which is, simplified, the reason for the limitations of applying cryptography to agents.

3.3 Nightmare III, or *what on Earth is going on in my System?*

A question that has barely been addressed in research so far is how to control a network with mobile agents: It is already very difficult to safely design dis-

[2]This can also be seen as splitting an agent into two agents: an actor (the program) and its assistant holding the data.

tributed systems with interacting, concurrent processes; agents make the situation even worse, since processes will also move within the network. Chess (Chess, 1996) observes quite correctly:

> *As a system becomes more complex and more interconnected, it is more likely to exhibit large-scale behaviors that we would not predict from knowledge of the individual elements.*

An impressive example of such effects for Telecommunications networks can be found in (Travis, 1990).

It can be expected that agents running in a large distributed system will cause new problems; little experience in this respect has been gained so far. There is a clear need for basic research in this direction; results from software engineering and formal methods are desperately needed.

4 Recent Approaches

Already in (Chess *et al.*, 1995), Chess et al describe the requirements on a secure and reliable agent system in which the agent

- carries with it the minimal, necessary and sufficient information to accomplish its task;
- can prove its authority to any host it may visit;
- can accumulate knowledge in the course of its itinerary and make its own decision based on that information.

In addition, the agent system should permit the detection of hosts which tamper with agents and their data, and provide sufficient auditing for hosts to prove that they have behaved properly.

Recent work in security theory has provided several ideas that can serve to establish security mechanisms for mobile agents. The DARPA Workshop on Foundations for Secure Mobile Code (DARPA97, 1997) tried to articulate the structures, semantics, security models and formal methods that form the foundation for remote evaluation. It became clear during the meeting that a careful examination of the fundamental issues surrounding secure mobile code is needed. The crux of many arguments for various strategies often depends on hidden assumptions about the model and the level of security one expects. Without an understanding of the model, it is very difficult to prove, or even argue for, interesting security properties in this paradigm.

4.1 CryptoComputing

In 1998, Sander and Tschudin gained considerable attention in the mobile agent community when they proposed the technique of “computing with encrypted functions” to overcome the difficulty of mobile code protection (Sander & Tschudin, 1998). The challenge is to find a transformation for a program into an encrypted form such that it is still executable. Thus, the code is protected during execution in the hostile environment without the need for trusted hardware. No interaction with the code producer should take place during execution.

Unfortunately, general-purpose software protection is one of the major unsolved areas in computer science and cryptography. This is true for the protection of software against piracy as well as for the protection of mobile code. But there are a number of special cases of the general problem that have been solved successfully, for example oblivious RAMs, where the memory access pattern of a CPU is concealed, and Private Information Retrieval, where a user can choose and secretly obtain one of n entries in a database, such that the database does not learn which entry the user has received. However, all known solutions for general secure multi-party computation are not sufficient because they require the participants to collaborate during the evaluation process. Typically, the protocols need one round of interaction *per elementary operation* of the program. A scheme that involves such strong interaction would clearly not qualify as mobile code.

A viable method seems to find an encryption scheme $E : X \to Y$ with cleartext space X and ciphertext space Y that has special homomorphic properties (Rivest *et al.*, 1978): There should exist operations $+$ and $\times$ on cleartexts and operations $\oplus$ and $\otimes$ on ciphertexts such that for cleartexts $x_1, x_2 \in X$,

$$E(x_1) \oplus E(x_2) = E(x_1 + x_2)$$

and

$$E(x_1) \otimes E(x_2) = E(x_1 \times x_2).$$

(The operation $+$ on X and $\oplus$ on Y are homomorphic and $\times$ on X and $\otimes$ on Y are homomorphic.)

A scheme that allows to encrypt polynomials, based on the Goldwasser-Micali encryption scheme, has been presented by Sander and Tschudin (Sander & Tschudin, 1998). An encryption scheme based on Error Correcting Codes, using the McEliece public-key scheme, is presented in (Loureiro & Molva, 1999).

We quote from (Young & Yung, 1999):

> Little progress has been made over the last 20 years in deciding whether or not these [algebraic homomorphic encryption] schemes exist. Promising candidates for such schemes have not been found, nor has evidence been given that such schemes cannot exist.

Finally, let us remark that the availability of general purpose encrypted functions would also affect the protection of the host: as the (internal) behavior of the program is hidden, no security analysis can be performed.

4.2 Voting techniques

General introduction ....

The basic idea in secret sharing is to divide the secret key into pieces and distribute the pieces to different parties so that certain subsets of the parties can get together to recover the key. (Minsky *et al.*, 1996; Schneider, 1997; Johansen *et al.*, 1999)

Secret sharing is the process of giving shares in a secret to a number of individuals so that together, but not individually, they may reconstruct the entire secret. Any network, electronic directory or computer system with access restrictions must have passwords or keys. It is desirable to have the most secure areas accessible only by mutual co-operation of a number of people which permits maximum security.

A more sophisticated and often more useful way to split up a secret among people is called *threshold* secret-sharing. Suppose the secret is to be shared among 10 people. A threshold is chosen, say 7. Then the secret is split in such a way that any seven of the parts are sufficient to reconstruct the secret but any smaller subset reveals no information about the secret.

4.3 Proof Techniques

4.3.1 Proof-carrying Code

Proof-carrying code (Necula, 1997; Necula & Lee, 1998) is a recently developed technique that aims at improving mobile code security by protecting hosts. The principle behind proof-carrying code is to obligate the code producer to carry out a formal (mathematical) proof that the code has certain specific safety properties. This proof is transmitted together with the code, such that a host that wants to carry out the code can always check that it satisfies the claimed safety properties. Technically, these proofs are then encoded as expressions in typed λ-calculus, such that proof-checking can be carried out by type checking the expression representing a proof.

The "philosophy" behind proof-carrying code is to minimize the amount of external trust needed for ensuring safety properties: the "authority" certifying that the transmitted code has certain safety properties is not a person or an institution that signed the code and its properties, but mathematical logic itself. This is certainly an advantage, since it minimizes the required infrastructure compared to cryptographic approaches, but it also leaves a number of important questions open:

Which is the generally accepted, standardized formal framework that supports proof carrying code?
Such a framework is needed to express safety properties, to carry out the proofs, and to check the proofs before run time. Furthermore, it requires the code to be written in a language that has a formal semantics in this framework.

The inventors of proof-carrying code suggest typed λ-calculus as the platform and first-order logic to express properties, which has the advantage that proof-checking can be carried out by type checking expressions. However, it is far from being obvious that typed λ-calculus is an adequate logic that meets users' requirements.[3]

[3] A purist might argue here that people could well use different logics as long as they can be

What do you do with the safety properties? Coping with safety-properties is a highly non-trivial task. What is required is a local security policy that is based on the language these properties are expressed in. Furthermore, relating these properties to this policy can be an arbitrary complex task, for instance requiring automated reasoning.

Where do the proofs come from? Formally proving properties of programs is a very difficult task. There are two principal ways to determine the safety properties of programs: automated, or interactive.

An automated approach can necessarily cope only with very limited languages for describing the properties, since it depends on decidability of the language. This restriction is serious, it does, for instance, not allow to include arithmetics. It is unclear if such a restricted language is in fact sufficient in practise.

An interactive approach can result in more complex, therefore potentially more useful properties, but requires the author of the program to carry out program verification. To say the least, programmers in today's life are quite a while away from this.

Overall, proof-carrying code is certainly an interesting approach to tackling the problem of protecting hosts in an agent-based scenario, but not yet applicable in practice. It is still unclear if all problems involved with the approach can eventually be solved in the future.

In the proof-carrying code approach, the binary is accompanied by annotations an a proof. Unfortunately, the annotations and the proof can disclose a lot of valuable information about design and implementation. Commercial vendors may be reluctant to reveal these information and require protection of their intellectual property rights (Devanbu & Stubblebine, 1997).

State appraisal is a similar approach proposed by Berkovits et al (Berkovits *et al.*, 1998). Goal is to ensure that an agent has not become malicious as a consequence of alternations of its state. Reputable manufacturers of mobile agents may provide agents with an appropriate (application-specific) state-appraisal function to be used each time a server starts an agent.

Proofs To guarantee that an agent's computation was done according to the program specified in the agent, one's possibility is to forward the entire execution trace to the originator, who checks it: probabilistically checkable proofs ("holographic proofs") and computationally sound proofs (Yee)

4.4 Secure Coprocessor

Yee as well as Wilhelm have proposed the use of a trusted and tamper-resistant hardware device, called Trusted Processing Environment (TPE) by Wilhelm,

related via semantics. This, however, just moves the problem from finding a common logic to a common semantics, which is, from a pragmatic point of view, equivalent.

as the secure execution environment for mobile objects (Wilhelm *et al.*, 1999; Yee, 1999). The TPE runs a virtual machine that can execute the mobile objects and it is only accessible via an interface controlled by the operating system of the TPE. The TPE contains a private key not known to any other entity. To download an application, the user sends the public key of his / her TPE to the provider of the application, who will encode the mobile objects that comprise the application with this key and send them to the user. Then the encrypted objects are uploaded to the TPE, which will decode and execute them. The manufacturer of the TPE guarantees the integrity of the execution environment to the provider of the mobile objects. The encryption of the objects under the public key of the TPE protects against manipulation and disclosure when in transit.

As mobile agents only move encrypted over the network and only execute within the TPE, Wilhelm has reduced the problem of general (multi-hop) security to client / server security. The TPE manufacturer has to ensure adequate protection of the TPE's internals against malicious code it may execute.

Secure Coprocessors begin to emerge on the market. At the high-end, there are general processing environemts with extreme built-in hardware (and software) security that satisfy Wilhelm's requirement that the TPE is a complete computer. For example, the IBM 4758 PCI Cryptographic Coprocessor provides a tamper-sensing and tamper-responding environment in which to run security-sensitive applications. Upon detection of physical attack, including penetration, radiation, voltage, or excessive cold or heat, the device is "zeroized" and the sensitive information erased (Smith & Weingart, 1999). At the low-end, the latest developments in smart card technology allow programs written in Java. For example, the Java Card provides a subset of Java that takes resource restrictionsdoes not include dynamic class loading, cloning, threads, or a security manager, and does not support types `char`, `double`, `float`, and `long` (Baentsch *et al.*, 1999). In general, none of the standard Java classes are supported in Java Card. On the other hand, Java Card provides support for persistent objects and atomic transactions. We believe that even with a simple trusted card, which only provides basic cryptography (signing, encryption and secure storage of private keys) and limited computation capabilities, interesting security services can be realized.

For example, Young and Yung introduce 'sliding encryption' that enciphers a small amount of data within a larger block and slides away a small fraction of the result, constituting the 'ciphertext' (Young & Yung, 1997). Thus, an agent can gather small amounts of data from several nodes across the network, and public key encrypt each piece of data without wasting an excessive amount of storage space. Sliding encryption can be used to efficiently store data in embedded cryptographic devices such as smartcards that have limited storage capacity. It can also be used for tracking the path traversed by an agent generating private audit trails.

trust splitting (Balfanz & Felten, 1999)

4.4.1 Smartcards

Devanbu and Stubblebine developed a protocol that allows to check the integrity of large data structures, based on stacks and queues, stored on a potentially hostile platforms (Devanbu & Stubblebine, 1998). Whenever an operation on the data structure is performed, a smartcard analyses and certifies the values. For that purpose, the smartcard keeps an initial secret value for each data object on the host. Hash chains link the initial secret with the elements added to and / or removed from the data object.

The use of a restricted tamper-resistant hardware, for example a smart card, has been proposed by Fünfrocken (Fünfrocken, 1999). Only some of the mobile agent operations, the security-sensitive ones, are executed within the protected environment.

ITSEC E6 certification for Mondex

4.4.2 Tamper-resistant software

In (Aucsmith, 1996), Aucsmith introduces the notion of *tamper-resistant software* that can be trusted, within certain bounds, to operate as intended even in the presence of a malicious attack. He applies a series of software design principles that produce tamper resistant *Integrity Verification Kernels (IVKs)* which can be inserted into programs to verify their integrity. An IVK is splitted into a number of independent cells that are self-decrypting and installation unique. All of their functions are interleaved and self-verifying. Additionally, IVKs can be interlocked so that a failure or by-pass of any IVK will be detected by another. Aucsmith notes that there is probably no protection against software debugging when processor emulators or other specialised hardware analysis tools are used.

Clueless agents are another approach to make software difficult to debug. In (Riordan & Schneier, 1998), Riordan and Schneier introduces a collection of cryptographic key constructions built from environmental data that are resistant to adversarial analysis and deceit. They introduce keys that depend upon temporal, spatial, or operational conditions. Using these keys, agents could receive encrypted messages that they could only decrypt if some environmental conditions were true. A clueless agent has a cipher-text (data or code) and a method for searching through the environment for the data needed to generate the decryption key. When the proper environmental information is located, the key is generated, the cipher-text is decrypted, and the resulting plain-text is acted upon. Without the environmentally supplied input, the agent cannot decrypt its own text (i.e. it is *clueless*), and can be made cryptographically resistantr to analysis at determining the agent's fucntion. Possible environments are data channels such as Usenet news groups or Web pages, or time. Clueless agents can be used as blind search engines as well as (bad) directed viruses.

In (Hohl, 1998), Hohl employs methods for "messing up" the agent's code such that it becomes hard to analyze it. Additionally, time stamps limit the validity of all token-based values (money, ...). Both ingredients together are claimed to give the agent a certain time period in which it runs safely as the host has not enough time to manipulate the agent: the first performs a semantic-preserving encryption that creates a "black box" strong enough to resist analysis within the given life time of the token-values. Thus, the gained information is useless since other trusted sites will not accept "expired" data.

This approach cannot solve all possible attacks against an agent, for instance, destruction of the agent cannot be prevented. Additional means are necessary to prevent "testing" by parallel execution of copies of an agent and observing its internal changes cannot be prevented. But it offers at least some protection against malicious hosts. Unfortunately, this approach does not offer strong results, in the sense that it is not mathematically provable that an agent can really not be analyzed with the given time. Thus, it ends up being security by obfuscation, which is generally believed not to be a satisfactory solution. Furthermore, the approach crucially depends on a network-wide, accurate time server.

4.5 Solutions based on time constraints

In the following we present some of the few approaches for protecting agents from hosts without requiring a trusted computing base. One of the biggest threats to an agent is the host's ability to determine its behavior by analyzing its internals; i.e. its implementation. If a host could not do this, it could still destroy an agent, but it could not manipulate it. The basic idea is that breaking of the system requires the attacker to break single agents in a *short* period of time. To counteract, agents are periodically refreshed (changed) before breakage or that most agents are secure most of the time.

4.5.1 Proactive Security

The model of Proactive Security does not make the assumption that one or more systems are always secure, i.e. are never controlled by the attackers. Instead, it considers cases where all components of the system may be broken-into. Recent results in cryptographic protocols show that many security goals may be achieved even under this assumption – as long as most systems are secure most of the time. Furthermore, these protocols do not even require to know when a system is broken into, or after the attacker loses control; instead, they pro-actively invoke recovery procedures every so often, hoping to restore security to systems over which the attacker lost control. These protocols combine cryptographic techniques from the area of "threshold cryptography" together with periodical refreshments of cryptographic data.

4.6 Reactive and social control

Reactive security and social control are after-the-fact approaches to security. A reactive security component complements social control mechanisms when it is used to feed back information about the outcomes of interactions between users and agents.

In (Vigna, 1997), Vigna proposes to audit all execution commands of the mobile agent on its itinerary. Based on the assumption that all interpreter implementations are certified to respect the semantics of the language, each receiving host has to audit its part of the execution trace of the agent. A trace is a sequence of statements together with the data received from outside of the execution environment. After execution, the host sends the hash values of final state and of the trace to a trusted party but keeps the complete trace and final state locally for possible later inspection. If the agent creator suspects that a host has cheated while executing the agent, he can ask the hosts to provide the trace. First it checks whether the trace satisfies the checksums stored at the trusted party. Next, the creator can replicate the execution of the agent providing the input values stored in the trace to check if it reaches the same final state. If any of these validation steps fails, the creator can identify the cheating party.

There are, however, two fundamental limitations in the proposal. First, the algorithm does not protect against attacks based on code and data inspection. Second, Vigna does not provide any criteria to determine if a received final state together with the checksums of the partial execution traces and results contain an attack.

Kelsey and Schneier use a cryptographic coprocessor to certify the outcomes of software programs (Schneier & Kelsey, 1997).

Rasmusson et al (Rasmusson *et al.*, 1997) argue to let agents handle security issues in Internet markets. The motivation for this is the difficulties in having external or centralized control over a system as open as the Internet. The agents use social control to warn each other of malicious agents. A security assistant aids the user in deciding the maliciousness of an agent by looking at its behavior, not by trusting the “good will” of the sender or a certificate from the system maintainer.

5 Conclusion

Agent-based software systems are very problematic with respect to security, since the principle of mobile code brings new problems for system security, reliability, and safety. Not very long ago, there was a widespread belief that an interpreted program is already a safe program. However, over time, mobile agent people became concerned about security and regard it today to be one of the biggest obstacles that have prevented wider employment of mobile agents.

There are three basic problems in agent-based systems: firstly, the hosts must be protected from potentially harmful agents, second, the roaming agents must be protected against manipulation from the visited hosts, and third the overall behavior of an agent-based system must be controlled.

Although, in principle, it is possible to secure a host against mobile agents, the price might be very high: hardware-based protection, reduced flexibility, and high cost. It is fundamentally impossible to protect an agent against a malicious host if no trusted hardware is available. There is a need for security profiles for agent execution environments to make informed judgments about tradeoffs between security and other desired features.

The current state-of-the-art still fails to deliver an acceptable approach to cope with these problems: available technology (like sandbox-approaches, cryptography) can cover only certain aspects, and even fails to solve these aspects sufficiently. A sound overall concept to cope with the problems seems not yet in sight. Mobile agents constitute an active research field and security of mobile code is (still) an area for further research.

It seems therefore prohibitive to fully opening Telecommunications networks for agents at this point. The only acceptable scenario that might to some extend be tolerable is to run only proprietary agents within controlled and isolated networks. Even this seems risky, since there is in practice no way to rigorously control a network beyond a certain size. PNOs know this quite well.

It is not sufficient to come up with attractive applications based on mobile agents. What is also needed is an applicable concept for coping with the inherent security problems. Such an overall concept must meet very strong requirements, otherwise it will not be accepted where real money comes into play. It is strongly suggested that the agent community takes security very serious if it is their goal to establish mobile agents in commercial computing and public networks. A taste of what criteria secure systems and platforms have to comply with can, for instance be gained by looking at the Common Criteria evaluation assurance levels[4].

Clearly, no running system can be 100% secure, and one has to compromise. In companies, however, compromises are governed by economic considerations: the potential earning must (at least in the long run) outweigh the costs, and the risk involved must be assessable. Thus, from a PNO's perspective, the future of commercial applications of mobile agents is hardly predictable; what is predictable, however, is a promising future for security experts.

Acknowledgments

The idea to write this paper originated at a panel session about mobile agent's security within the 1st International Mobile Agents Workshop 1997. Impressed by Andreas Pfitzmann's clear analysis about what is possible and, in particular, what is not, we decided to survey all relevant security issues in a particular application area.

The first author would like to thank his colleagues from the cryptography groups at Deutsche Telekom, in particular Ernst Giessman and Günter Lassman, for valuable comments on Section 3.1.3.

[4]See (British Standards Institution, n.d.) for details

References

. 1998. *Proceedings of the IEEE Symposium on Research in Security and Privacy*. Research in Security and Privacy. Oakland, CA: IEEE Computer Society Press.

ALEXANDER, D. SCOTT, ARBAUGH, WILLIAM A., KEROMYTIS, ANGELOS D., & SMITH, JONATHAN M. 1997. A Secure Active Network Environment Architecture. *IEEE Network Magazine, special issue on Active and Controllable Networks*, **12**(3), 37–45.

ALEXANDER, D. SCOTT, ANAGNOSTAKIS, KOSTAS G., ARBAUGH, WILLIAM A., KEROMYTIS, ANGELOS D., & SMITH, JONATHAN M. 1999a. The Price of Safety in an Active Network. *In: SIGCOMM '99*.

ALEXANDER, D. SCOTT, ARBAUGH, WILLIAM A., KEROMYTIS, ANGELOS D., & SMITH, JONATHAN M. 1999b. Security in Active Network. *In:* (Vitek & Jensen, 1999).

ALVES-FOSS, J. (ed). 1999. *Formal Syntax and Semantics of Java*. Lecture Notes in Computer Science, no. 1523. Springer.

APPLEBY, S., & STEWARD, S. 1994. Mobile Software Agents for Control in Telecommunications Networks. *BT Technology Journal*, **12**(2), 104–113.

AUCSMITH, DAVID. 1996. Tamper Resistant Software: An Implementation. *Pages 317–333 of:* ANDERSON, ROSS J. (ed), *Information Hiding 1996*. Lecture Notes in Computer Science, no. 1174. Springer.

BAENTSCH, M., BUHLER, P., EIRICH, T., HÖRING, F., & OESTREICHER, M. 1999. JavaCard – From Hype to Reality. *IEEE Concurrency*, **7**(4), ?–?

BALDI, M., & PICCO, G. 1998. Evaluating the Tradeoffs of Mobile Code Design Paradigms in Network Management Applications. *Pages 46–155 of: 20th International Conference on Software Engineering (ICSE'98)*. IEEE Computer Press.

BALFANZ, DIRK, & FELTEN, EDWARD W. 1999. Hand-Held Computers Can Be Better Smart Cards. *In: 8th USENIX Security Symposium*.

BERKOVITS, S., GUTTMAN, J.D., & SWARUP, V. 1998. Authentication for Mobile Agents. *In:* (Vigna, 1998).

BIESZCZAD, A., PAGUREK, B., & WHITE, T. 1998. Mobile Agents for Network Management. *IEEE Communications Surveys*, **1**(1), 2–9. /urlhttp://www.comsoc.org/pubs/surveys.

BONABEAU, E., HENAUX, F., GUÉRIN, S., SNYERS, D., KUNTZ, P., & THERAULAZ, G. 1998. Routing in Telecommunications Networks with Ant-Like Agents. *Pages 60–71 of:* ALBAYRAK, S., & GARIJO, F.J. (eds), *Intelligent Agents for Telecommunications Applications '98 (IATA'98)*. Lecture Notes in Computer Science, no. 1437. Springer-Verlag, Berlin Germany.

CASTILLO, A., KAWAGUCHI, M., PACIOREK, N., & WONG, D. 1998 (June). *Concordia as Enabling Technology for Cooperative Information Gathering*. Japanese Society for Artificial Intelligence Conference.

CHESS, D. 1996 (May). Security Considerations in Agent-Based Systems. *In: First Conference on Emerging Technologies and Applications in Communications (etaCOM)*.

CHESS, D., GROSOF, B., HARRISON, C., LEVINE, D., PARRIS, C., & TSUDIK, G. 1995. Itinerant Agents for Mobile Computing. *IEEE Personal Communication Systems*, **2**(5), 34–49.

DARPA97. 1997 (Mar.). *DARPA Workshop on Foundations for Secure Mobile Code*. http://www.cs.nps.navy.mil/research/languages/wkshp.html.

DEVANBU, P., & STUBBLEBINE, S. 1997. Research directions for automated software verification: Using trusted hardware. *In: 12th IEEE Int'l Conference on Automated Software Engineering – ASE'97*. IEEE Computer Society.

DEVANBU, PREKMKUMAR T., & STUBBLEBINE, STUART G. 1998. Stack and Queue Integrity on Hostile Platforms. *In:* (Oak, 1998).

DI CARO, G., & DORIGO, M. 1998 (Jan.). Mobile Agents for Adaptive Routing. *In: 31st Hawaii International Conference on System Science*.

DIGICRIME, INC. *A full service criminal computer hacking organization*. http://www.digicrime.com.

FORD, W. 1994. *Computer Communications Security*. Prentice-Hall.

FRITZINGER, J.S., & MUELLER, M. 1996. *Java Security*. Tech. rept. Sun Microsystems, Inc.

FÜNFROCKEN, STEFAN. 1999. Protecting Mobile Web-Commerce Agents with Smartcards. *In: ASA/MA 99*.

GREENBERG, M.S., BYINGTON, J.C., & HOLDING, T. 1998. Mobile Agents and Security. *IEEE Communications MAGAZINE*, **36**(7), 76–85.

HAMILTON, MARC A. 1996. Java and the Shift to Net-Centric Computing. *IEEE Computer*, Aug., 31–39.

HAYZELDEN, ALEX L.G., & BIGHAM, JOHN (eds). 1999. *Software Agents for Future Communication Systems: Agent Based Digital Communication*. Springer-Verlag, Berlin Germany.

HOHL, F. 1998. Time Limited Blackbox Security: Protecting Mobile Agents From Malicious Hosts. *In:* (Vigna, 1998).

JANSEN, W., & KARYGIANNIS, T. 1999. *Mobile Agent Security*. Tech. rept. National Institute of Standards and Technology.

JENNINGS, N.R., & WOOLDRIDGE, M.J. 1998. *Agent Technology: Foundations, Applications, and Markets*. Springer-Verlag, Berlin Germany.

JOHANSEN, DAG, MARZULLO, KEITH, SCHNEIDER, FRED B., JACOBSEN, KJETIL, & ZAGORODNOV, DMITRII. 1999. NAP: Practical Fault-Tolerance for Itinerant Computations. *Pages 180–189 of: 19th IEEE International Conference on Distributed Computing Systems (ICDCS'99)*.

KAUFMAN, C., PERLMAN, R., & SPECINER, M. 1995. *Network Security: Private Communication in a Public World*. Prentice-Hall.

KRAMER, KWINDLA HULTMAN, MINAR, NELSON, & MAES, PATTIE. 1999. Tutorial: Mobile Software Agents for Dynamic Routing. *Mobile Computing and Communications Review*, **3**(2), 12–16.

LANGE, DANNY B., & OSHIMA, MITSURU. 1998. *Programming and Deploying Java Mobile Agents with Aglets*. Addison-Wesley.

LOUREIRO, SERGIO, & MOLVA, REFIK. 1999. Function Hiding Based on Error Correcting Codes. *In: International Workshop on Cryptographic Techniques and E-Commerce (CrypTEC '99)*.

MAGEDANZ, T., ROTHERMEL, K., & KRAUSE, S. 1996. Intelligent Agents: An Emerging Technology for Next Generation Telecommunications? *Pages 464–472 of: IEEE INFOCOM 1996*.

BRITISH STANDARDS INSTITUTION. *Common Criteria V1.0 Web Page*. http://www.itsec.gov.uk/itsechtml/ccv1.0/ccv1.htm.

SIP, PRINCETON SAFE INTERNET PROGRAMMING GROUP. *Web Page*. http://www.cs.princeton.edu/sip/News.html,.

SUN MICROSYSTEMS. *Security FAQ*. http://www.javasoft.com/sfaq,.

MCGRAW, G., & FELTEN, E.W. 1997. *Java Security: Hostile Applets, Holes, and Antidotes*. Wiley.

MCGRAW, G., & FELTEN, E.W. 1998. *Securing Java: Getting Down to Business with Mobile Code*. Wiley.

MINAR, NELSON, KRAMER, KWINDLA HULTMAN, & MAES, PATTIE. 1999. *Cooperating Mobile Agents for Dynamic Network Routing*. Springer-Verlag, Berlin Germany. Chap. 12.

MINSKY, Y., VAN RENESSE, R., SCHNEIDER, F.B., & STOLLER, S.D. 1996. Cryptographic Support for Fault-Tolerant Distributed Computing. *Pages 109–114 of: Seventh ACM SIGOPS European Workshop.*

MOORE, JONATHAN T. 1998 (May). *Mobile Code Security Techniques.* Tech. rept. MS-CIS-98-28. University of Pennsylvania, Department of Computer and Information Science.

NECULA, G.C. 1997 (Jan.). Proof-carrying code. *Pages 106–119 of: Proceedings of the 24th ACM Symposium on Principles of Programming Languages (POPL'97).*

NECULA, G.C., & LEE, P. 1998. Safe, Untrusted Agents using Proof-Carrying Code. *In:* (Vigna, 1998).

NEUMANN, PETER G. 1997a (February). *RISKS DIGEST, Vol. 18, No. 61.* http://catless.ncl.ac.uk/Risks/.

NEUMANN, PETER G. 1997b (February). *RISKS DIGEST, Vol. 18, No. 82.* http://catless.ncl.ac.uk/Risks/.

NWANA, H.S. 1996. Software Agents: An Overview. *Knowledge Engineering Review*, **11**(3), 205–244.

ORDILLE, J.J. 1996 (May). When agents roam, who can you trust? *In: First Conference on Emerging Technologies and Applications in Communications (etaCOM).*

OUSTERHOUT, J.K., LEVY, J.Y., & WELCH, B.B. 1998. The Safe-Tcl Security Model. *In:* (Vigna, 1998).

PLU, M. 1998. Software Technologies for Building Agent Based Systems in Telecommunication Networks. *In:* (Jennings & Wooldridge, 1998).

POSEGGA, J., & VOGT, H. 1998. Byte Code Verification for Java Smart Cards Based on Model Checking. *Pages 175–190 of: 5th European Symposium on Research in Computer Security (ESORICS).* Lecture Notes in Computer Science. Springer.

QIAN, Z. 1999. A Formal Specification of Java Virtual Machine Instructions for Objects, Methods and Subroutines. *In:* (Alves-Foss, 1999).

RASMUSSON, L., RASMUSSON, A., & JANSON, S. 1997. Using agents to secure the Internet marketplace – Reactive Security and Social Control. *In: Practical Applications of Agents and Multi-Agent Systems 1997 (PAAM'97).*

REINHARDT, A. 1994. The Network with Smarts. *Byte*, Oct., 15–66.

RIORDAN, J., & SCHNEIER, B. 1998. Environmental Key Generation towards Clueless Agents. *In:* (Vigna, 1998).

RIVEST, R.L., ADLEMAN, L., & DERTOUZOS, M.L. 1978. On Data Banks and Privacy Homomorphisms. *Pages 169–177 of*: DEMILLO, R.A., DOBKIN, D.P., JONES, A.K., & LIPTON, R.J. (eds), *Foundations of Secure Computing*. Academic Press.

ROTHERMEL, K., & POPESCU-ZELETIN, R. (eds). 1997. *First International Workshop on Mobile Agents (MA '97)*. Lecture Notes in Computer Science, no. 1219. Springer.

SANDER, T., & TSCHUDIN, C. 1998. Towards Mobile Cryptography. *In:* (Oak, 1998).

SCHNEIDER, F.B. 1997 (Sept.). Towards Fault-tolerant and Secure Agentry. *In: 11th Int. Workshop on Distributed Algorithms*.

SCHNEIER, B., & KELSEY, J. 1997. Remote Auditing of Software Outputs Using a Trusted Coprocessor. *Journal of Future Generation Computer Systems*, **13**(1), 9–18.

SCHOONDERWOERD, R., HOLLAND, O., & BRUTEN, J. 1997. Ant-like Agents for Load Balancing in Telecommunications Networks. *In: Agents'97*. ACM Press.

SMITH, S.W., & WEINGART, S.H. 1999. Building a High-Performance, Programmable Secure Coprocessor. *Computer Networks and ISDN Systems (Special Issue on Network Security)*, **31**(Apr.), 831–860.

TARDO, J., & VALENTE, L. 1996. Mobile agent security and Telescript. *Pages 58–63 of*: *IEEE CompCon '96*.

TENNENHOUSE, D.L., SMITH, J.M., SINCOSKIE, W.D., WETHERALL, D.J., & MINDEN, G. J. 1997. A Survey of Active Network Research. *IEEE Communications Magazine*, **35**(1), 80–86.

TRAVIS, P. 1990. Why the AT&T Network crashed. *Telephony*, **218**(4), 11.

TSCHUDIN, C. 1999. Mobile Agent Security. *Pages 431–445 of*: KLUSCH, M. (ed), *Intelligent Information Agents: Cooperative, Rational and Adaptive Information Gathering on the Internet*. Springer-Verlag, Berlin Germany.

VIGNA, G. 1997 (June). Protecting Mobile Agents through Tracing. *In: Third Workshop on Mobile Object Systems*.

VIGNA, G. (ed). 1998. *Mobile Agents and Security*. Lecture Notes in Computer Science, vol. 1419. Springer.

VIRDHAGRISWARAN, SANKAR. 1997 (Jan.). *Agents – State of The Art & The Incredible Future*. `http://hunchuen.crystaliz.com/research/sub/agents.pdf`. CrystalIZ, Inc.

VITEK, J., & JENSEN, C. (eds). 1999. *Secure Internet Programming: Security Issues for Mobile and Distributed Objects.* Lecture Notes in Computer Science, no. 1603. Springer.

WILHELM, U.G., STAAMANN, S., & BUTTYÀN, L. 1999. Introducing Trusted Third Parties to the Mobile Agent Paradigm. *In:* (Vitek & Jensen, 1999).

YEE, B.S. 1999. A Sanctuary for Mobile Agents. *In:* (Vitek & Jensen, 1999).

YELLIN, FRANK. 1995 (Dec.). Low Level Security in Java. *In: Fourth International Conference on the World-Wide Web.*

YOUNG, A., & YUNG, M. 1997. Encryption Tools for Mobile Agents: Sliding Encryption. *In:* BIHAM, E. (ed), *Fast Software Encryption.* Lecture Notes in Computer Science, no. 1267. Springer-Verlag, Berlin Germany.

YOUNG, T. SANDER A., & YUNG, M. 1999. Non-Interactive CryptoComputing For NC$\hat{1}$. *In: 40th Annual Symposium on Foundations of Computer Science.*