J. of Comp. and I.T. Vol.1(2), 109-122 (2010). ISSN 2229-3531 (Print)
www.compitjournal.org 2455-9997 (Online)

Mobile Agent Technology -Security Threats and Measures

M A RIZVI[1], R K KAPOOR[2], M M MALIK[3] and SANJAY SHARMA[4]

[1]National Institute of Technical Teachers' Training and Research
Shamla Hills, Bhopal-462002, M.P. (INDIA)
marizvi@hotmail.com
917552661600 (O), 917552547434 (R), 09425012014 (M)
Fax 91755661996

[2]National Institute of Technical Teachers' Training and Research
Shamla Hills, Bhopal-462002, M.P. (INDIA)
sunkapoor@hotmail.com
917552661600 (O), 9175522493255 (R), 09425011468 (M)
Fax 91755661996

[3]Maulana Azad National Institute of Technology
Department of Physics MANIT, Bhopal-462007, M.P. (INDIA)
Mmjmalik2004@yahoo.co.in
917552670417 (O), 9175522493255 (R), 09827378294 (M)
Fax 9172670562

[4]Maulana Azad National Institute of Technology
Department of Computer Applications MANIT, Bhopal-462002, M.P. (INDIA)
ssharma66@rediffmail.com
917552670417 (O), 9175522493255 (R), 09425013166 (M)
Fax 9172670562

(Acceptance Date 27th September, 2010)

Abstract

Mobile agent technology offers a new computing paradigm in which a program, in the form of a software agent, can start its execution on a host computer that provides the necessary computational environment to run that agent, suspend its execution on the host computer, transfer itself to another agent-enabled host on the network, and resume execution on the new host. Incarnations of current generation agent can be characterized in a number of ways varying from simple distributed objects to highly organized software with embedded intelligence. A mobile agent is a particular type of agent with the ability to migrate from one host to another where it can resume its execution.

Thus, as the sophistication of mobile software has increased, the associated threats to security have also increased. This paper provides an overview of the varicty of threats that are being faced by the designers of agent platforms and the developers of agent based applications. The paper also identifies generic objectives of agent security, and presents a range of measures for countering the identified threats and fulfilling these security objectives.

1. Introduction

An agent is defined as "a person whose job is to act for, or manage the affairs of, other people". In the context of computers, software agents refer to programs that perform certain tasks on behalf of the user[1]. Agents are independent pieces of software capable of acting autonomously in response to input from their environment. Agents can be of differing abilities, but typically possess the required functionality to fulfil their design objectives. To be described as 'intelligent', software agents should also have the ability of acting autonomously that is without direct human interaction, be flexible, and in a multi-agent system, be able to communicate with other agents that are to be social.

In order to execute, the agents need an environment. The agent platform provides the necessary computational environment in which an agent operates. A computer host, the immediate environment of an agent, is ultimately responsible for the correct execution and protection of the agent.

The environment might also need certain protection from the agents that it hosts. An agent should, for example, be prevented from launching a denial of service attack through consuming all resources on a host, thus preventing the host from carrying out other things (such as executing other agents).

Agents may be stationary, always resident at a single platform; or mobile, capable of moving among different platforms at different times. Accordingly the security threats vary depending on the characteristic of the agent, whether they are stationary or mobile? This paper focuses mainly on the security issues that arise when mobility of agents comes into picture. Security issues related to the executing host become even more apparent for agents that are mobile.

2. *Security Threats* :

Threats to security generally fall into three main classes: disclosure of information, denial of service, and corruption of information. There are a variety of ways to inspect these classes of threats in greater detail as they apply to agent systems. Here, we use the components of an agent system to categorize the threats as a way to identify the possible source and target of an attack.

Because of its nature, the mobile agents are at potentially greater risk for abuse and misuse significantly.

There are a number of models existing

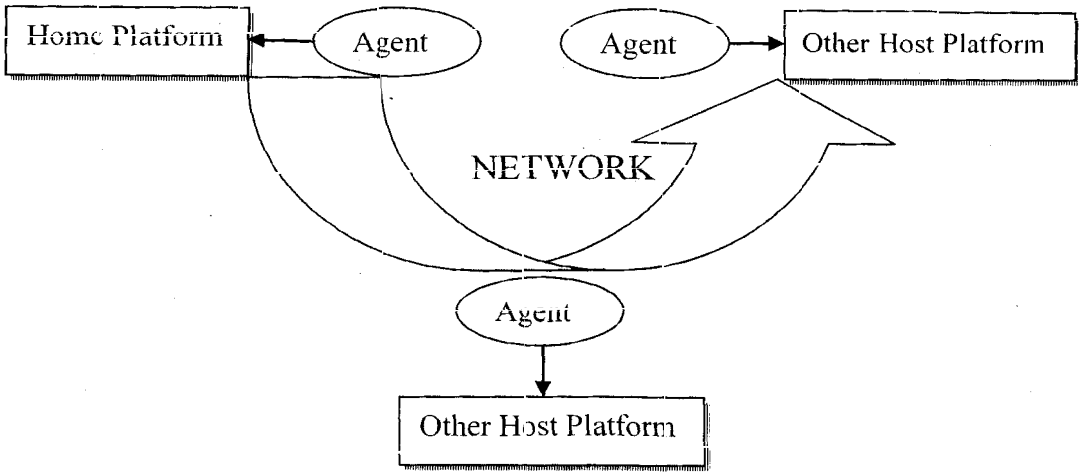

Figure 1. Agent System Model

to describe agent systems[2,6,7]; however, for the discussion of security issues it is adequate to use a simple model, consisting of two main components, the agent and the agent platform. Here, an agent is comprised of the code and information related to state that is required to carry out some computation. Mobility allows an agent to move, or hop, among agent platforms.

Four threat categories are identified: threats stemming from an agent attacking an agent platform, an agent platform attacking an agent, an agent attacking another agent on the agent platform, and other entities attacking the agent system.

2.1. *Agent-to-Platform :*

The agent-to-platform category represents the set of threats in which agents take advantage of security weaknesses of an agent platform or launch attacks against an agent platform. This set of threats includes masquerading, denial of service and unauthorized access.

2.1.1. *Masquerading :*

When an unauthorized agent claims the identity of another agent it is said to be masquerading. The masquerading agent may pose as an authorized agent and may try to gain access to services and resources to which it is not entitled.

2.1.2. *Denial of Service*

The mobile computing paradigm requires an agent platform to accept and execute an agent whose code may have been developed outside its organization. A rogue agent may carry malicious code that is designed to disrupt the services offered by the agent platform, degrade the performance of the platform, or extract information for which it has no

authorization to access.

Mobile agents can consume an excessive amount of the agent platform's computing resources and can launch denial of service attacks. These denials of service attacks can be launched intentionally by running attack scripts to take advantage of system vulnerabilities, or it may happen unintentionally by having some bugs in the programming.

2.1.3. *Unauthorized Access :*

Each agent visiting a platform must be subject to the platform's security policy. Access control mechanisms are used to prevent unauthorized users or processes from accessing services and resources for which they have not been granted permission and privileges as specified by a security policy. Applying the proper access control mechanisms requires the platform or agent to first authenticate a mobile agent's identity before it is instantiated on the platform. An agent that has access to a platform and its services without having the proper authorization can harm other agents and the platform itself. A platform that hosts agents must ensure that agents do not have read or write access to data for which they have no authorization, including access to residual data that may be stored in a cache or other temporary storage.

2.2 *Agent-to-Agent :*

The agent-to-agent category represents the set of threats in which agents take advantage of security weaknesses of other agents or launch attacks against other agents. This set of threats includes masquerading, unauthorized access, denial of service and repudiation.

2.2.1. *Masquerade :*

Agent-to-agent communication can take place directly between two agents or may require the participation of the underlying platform and the agent services it provides. In either case, an agent may attempt to disguise its identity in an effort to deceive the agent with which it is communicating.

Masquerading as another agent harms both the agent that is being deceived and the agent whose identity has been assumed.

2.2.2. *Denial of Service :*

In addition to launching denial of service attacks on an agent platform, agents can also launch denial of service attacks against other agents. For example, repeatedly sending messages to another agent, or spamming agents with messages, may place undue burden on the message handling routines of the recipient. Malicious agents can also intentionally communicate false or useless information to prevent other agents from completing their tasks correctly.

2.2.3. *Repudiation :*

Repudiation occurs when an agent, participating in a transaction or communication, later claims that the transaction or communication never took place. Whether the cause for repudiation is deliberate or accidental, repudiation can lead to serious disputes that may not be

easily resolved unless the proper countermeasures are in place. An agent platform cannot prevent an agent from repudiating a transaction, but platforms can ensure the availability of sufficiently strong evidence to support the resolution of disagreements.

2.3. *Platform-to-Agent :*

The platform-to-agent category represents the set of threats in which platforms compromise the security of agents. This set of threats includes masquerading, denial of service, eavesdropping, and alteration.

2.3.1. *Masquerade :*

One agent platform can masquerade as another platform in an attempt to deceive a mobile agent as to its true destination and corresponding security domain. The masquerading platform can harm both the visiting agent and the platform whose identity it has assumed.

When an agent arrives at an agent platform it is exposing its code, state, and data to the platform. Since an agent may visit several platforms under various security domains throughout its lifetime, mechanisms must be in place to ensure the integrity of the agent's code, state, and data.

2.4. *Other-to-Agent Platform :*

The other-to-agent platform category represents the set of threats in which external entities, including agents and agent platforms, may cause threat to the security of an agent platform. This set of threats includes masquerading, denial of service, unauthorized access, and copy and replay.

2.4.1. *Masquerade :*

Agents can request platform services both remotely and locally. An agent on a remote platform can masquerade as another agent and request services and resources for which it is not authorized. Agents masquerading as other agents may act in conjunction with a malicious platform to help deceive another remote platform or they may act alone. A remote platform can also masquerade as another platform and mislead unsuspecting platforms or agents about its true identity.

2.4.2. *Unauthorized Access*

Remote users, processes, and agents may request resources for which they are not authorized. Remote access to the platform and the host machine itself must be carefully protected, since conventional attack scripts can be used to destabilize the operating system and directly gain control of all resources.

2.4.3. *Denial of Service :*

Agent platform services can be accessed both remotely and locally. The agent services offered by the platform and inter-platform communications can be disrupted by common denial of service attacks. Agent platforms are also susceptible to all the conventional denial of service attacks aimed at the underlying operating system or communication protocols.

2.4.4. *Copy and Replay :*

Whenever a mobile agent moves from one platform to another it increases its exposure to security threats. A party that intercepts an agent, or agent message, in transit can attempt to copy the agent, or agent message, and clone or retransmit it.

3. *Security Requirements :*

The users of networked computer systems have four main security requirements: confidentiality, integrity, availability, and accountability. The users of agent and mobile agent frameworks also have these same security requirements. This section provides a brief overview of these security requirements and how they apply to agent frameworks.

3.1 *Confidentiality :*

Any private data stored on a platform or carried by an agent must remain confidential. Agent frameworks must be able to ensure that their intra- and inter-platform communications remain confidential. Eavesdroppers can gather information about an agent's activities not only from the content of the messages exchanged, but also from the message flow from one agent to another agent or agents. Since audit logs maintain a detailed record of an agent's activities on the platform, the contents of the audit log must also be carefully protected and remain confidential.

3.2 *Integrity*

The agent platform must protect agents from unauthorized modification of their code, state, and data and ensure that only authorized agents or processes carry out any modification of shared data. The agent itself cannot prevent a malicious agent platform from tampering with its code, state, or data, but the agent can take measures to detect this tampering. The secure operation of mobile agent systems also depends on the integrity of the local and remote agent platforms themselves.

3.3 *Accountability :*

Each process, human user or agent on a given platform must be held accountable for their actions, such as: access to an object, such as a file, or making administrative changes to a platform security mechanism. In order to be held accountable each process, human user, or agent must be uniquely identified, authenticated, and audited. Accountability is also essential for building trust among agents and agent platforms.

3.4. *Availability :*

The agent platform must be able to ensure the availability of both data and services to local and remote agents. The agent platform must be able to provide controlled concurrency, support for simultaneous access, deadlock management, and exclusive access as required. Shared data must be available in a usable form, capacity must be available to meet service needs, and provisions for the fair allocation of resources and timeliness of service must be made.

3.5. *Anonymity :*

The agent platform may need to balance an agent's need for privacy with the

platform's need to hold the agent accountable for its actions. The platform may be able to keep the agent's identity secret from other agents and still maintain a form of reversible anonymity where it can determine the agent's identity if necessary and legal. Moreover, what information belongs in public agent directories and under what conditions the information can be accessed from these directories must also be carefully controlled?

4. *Countermeasures :*

Many conventional security techniques used in contemporary distributed applications like client-server also have utility as countermeasures with in the mobile agent paradigm. Here these countermeasures are reviewed by considering those techniques that can be used to protect agent platforms, separately from those used to protect the agents that run on them.

Most agent systems rely on a common set of baseline assumptions regarding security. The first is that an agent trusts the home platform where it is instantiated and begins execution. The second is that the home platform and other equally trusted platforms are implemented securely, with no flaws or trapdoors that can be exploited, and behave non-maliciously. The third is that public key cryptography, primarily in the form of digital signature, is utilized through certificates and revocation lists managed through a public key infrastructure.

4.1. *Protecting the Agent Platform :*

One of the main concerns with an agent system implementation is ensuring that agents are not able to interfere with one another or with the underlying agent platform. One common approach for accomplishing this is to establish separate isolated domains for each agent and the platform, and control all inter-domain access. In traditional terms this concept is referred to as a reference monitor[12].

4.1.1. Software-Based Fault Isolation :

Software-Based Fault Isolation[13] is a method of isolating application modules into distinct fault domains enforced by software. The technique allows untrusted programs written in an unsafe language, such as C, to be executed safely within the single virtual address space of an application. Untrusted machine interpretable code modules are transformed so that all memory accesses are confined to code and data segments within their fault domain.

Access to system resources can also be controlled through a unique identifier associated with each domain.

4.1.2. *Safe Code Interpretation :*

Agent systems are often developed using an interpreted script or programming language. The main motivation for doing this is to support agent platforms on heterogeneous computer systems. Moreover, the higher conceptual level of abstraction provided by an interpretative environment can facilitate the development of the agent's code[14]. The idea behind Safe Code Interpretation is that commands considered harmful can be either made safe for or denied to an agent.

One of the most widely used interpretative languages today is Java. The Java programming language and runtime environment[2] enforces security primarily through strong type safety.

There are many agent systems based on Java, including Aglets[3,14], Mole[3], Ajanta[15], and Voyager[4]. However, limitations of Java to account for memory, CPU, and network resources consumed by individual threads[22] and to support thread mobility[2] have been noted.

4.1.3. *Signed Code :*

A fundamental technique for protecting an agent system is signing code or other objects with a digital signature. A digital signature serves as a means of confirming the authenticity of an object, its origin, and its integrity. Typically the code signer is either the creator of the agent, the user of the agent, or some entity that has reviewed the agent. Because an agent operates on behalf of an end-user or organization, mobile agent systems[10,11,14] commonly use the signature of the user as an indication of the authority under which the agent operates.

Code signing involves public key cryptography, which relies on a pair of keys associated with an entity. One key is kept private by the entity and the other is made publicly available.

Digital signatures benefit greatly from the availability of a public key infrastructure, since certificates containing the identity of an entity and its public key (*i.e.*, a public key certificate) can be readily located and verified.

Because the message digest is unique, and thus bound to the code, the resulting signature also serves as an integrity mechanism. The agent code, signature, and public key certificate can then be forwarded to a recipient, who can easily verify the source and authenticity of the code.

4.1.4. *State Appraisal :*

The goal of State Appraisal[6] is to ensure that an agent has not been damaged due to alterations of its state information. The success of the technique relies on the extent to which harmful alterations to an agent's state can be predicted, and countermeasures, in the form of appraisal functions, can be prepared before using the agent. Appraisal functions are used to determine what privileges to grant an agent based both on conditional factors and whether identified state invariants hold. An agent whose state violates an invariant can be granted no privileges, while an agent whose state fails to meet some conditional factors may be granted a restricted set of privileges. Both the author and owner of an agent produce appraisal functions that become part of an agent's code. An owner typically applies state constraints to reduce liability and/or control costs. When the author and owner each digitally sign the agent, their respective appraisal functions are protected from undetectable modification.

4.1.5. *Path Histories :*

The basic idea behind Path Histories[20] is to maintain an authenticable record of the prior platforms visited by an agent, so that a newly

visited platform can determine whether to process the agent and what resource constraints to apply. Computing a path history requires each agent platform to add a signed entry to the path, indicating its identity and the identity of the next platform to be visited, and to supply the complete path history to the next platform.

4.1.6. *Proof Carrying Code :*

The approach taken by Proof Carrying Code[7] obligates the code producer to formally prove that the program possesses safety properties previously stipulated by the code consumer e.g., security policy of the agent platform. Proof Carrying Code is a prevention technique, while code signing is an authenticity and identification technique used to deter, but not prevent the execution of unsafe code. The code and proof are sent together to the code consumer where the safety properties can be verified.

A safety predicate, representing the semantics of the program, is generated directly from the native code to ensure that the companion proof does in fact correspond to the code. The proof is structured in a way that makes it straightforward to verify without using cryptographic techniques or external assistance. Once verified, the code can run without further checking. Any attempts to tamper with either the code or the safety proof result in either a verification error or, if the verification succeeds, a safe code transformation.

4.2. *Protecting Agents :*

An agent is completely susceptible to an agent platform and cannot prevent malicious behaviour from occurring, but may be able to detect it.

The problem stems from the inability to effectively extend the trusted environment of an agent's home platform to other agent platforms. While a user may digitally sign an agent on its home platform before it moves onto a second platform, that protection is limited.

The second platform receiving the agent can rely on the signature to verify the source and integrity of the agent's code, data, and state information provided that the private key of the user has not been compromised. On the agent's subsequent hop to a third platform, the initial signature from the first platform remains valid for the original code, data, and state information, but not for any state or data generated on the second platform. While these simple schemes have value, they do not support the loose roaming itineraries envisioned in many agent applications.

An idea for the protection of mobile agents against the attacks of malicious hosts is to limit the execution time in the hosts. Malicious hosts need time to analyze and modify an agent in order to take some profit. Controlling the execution time in the hosts permits detecting manipulation attacks performed by malicious hosts during the agent's execution[16].

Some more general-purpose techniques for protecting an agent include the following:

· Partial Result Encapsulation,

- Mutual Itinerary Recording,
- Itinerary Recording with Replication and Voting,
- Execution Tracing,
- Environmental Key Generation,
- Computing with Encrypted Functions, and
- Obfuscated Code (Time Limited Blackbox).

4.2.1. *Partial Result Encapsulation :*

One approach used to detect tampering by malicious hosts is to encapsulate the results of an agent's actions, at each platform visited, for subsequent verification, either when the agent returns to the point of origin or possibly at intermediate points as well. Encapsulation may be done for different purposes with different mechanisms, such as providing confidentiality using encryption, or for integrity and accountability using digital signature. In general, there are three alternative ways to encapsulate partial results:

Provide the agent with a means for encapsulating the information,

Rely on the encapsulation capabilities of the agent platform, or

Rely on a trusted third party to timestamp a digital fingerprint of the results.

Another method for an agent to encapsulate result information is to use Partial Result authentication Codes (PRAC)[17], which are cryptographic checksums formed using secret key cryptography (*i.e.*, message authentication codes). The PRAC technique has a number of limitations. The most serious occurs when a malicious platform retains copies of the original keys or key generating functions of an agent.

Yee, who proposed the PRAC technique, noted that forward integrity could also be achieved using a trusted third party that performs digital time-stamping. A digital timestamp[18] allows one to verify that the contents of a file or document existed, as such, at a particular point ir time.

4.2.2. *Mutual Itinerary Recording :*

One variation of Path Histories is a general scheme for allowing an agent's itinerary to be recorded and tracked by another cooperating agent and vice versa[9], in a mutually supportive arrangement. When moving between agent platforms, an agent conveys the last platform, current platform, and next platform information to the cooperating peer through an authenticated channel. The peer maintains a record of the itinerary and takes appropriate action when inconsistencies are noted. Attention is paid so that an agent avoids platforms already visited by its peer.

4.2.3. *Itinerary Recording with Replication and Voting :*

A faulty agent platform can behave similar to a malicious one. Therefore, applying fault tolerant capabilities to this environment should help counter the effects of malicious platforms. One such technique for ensuring that a mobile agent arrives safely at its destination is through the use of replication and voting[19]. The idea is that rather than a single copy of an

agent performing a computation, multiple copies of the agent are used. Although a malicious platform may corrupt a few copies of the agent, enough replicates avoid the encounter to successfully complete the computation. For each stage of the computation, the platform ensures that arriving agents are intact, carrying valid credentials. Platforms involved in a particular stage of a computation are expected to know the set of acceptable platforms for the previous stage. The platform propagates onto the next stage only a subset of the replica agents it considers valid, based on the inputs it receives. One drawback of this approach is that, the additional resources consumed by replicate agents.

Stefan Pleisch and Andre Schiper identifies two important properties for fault-tolerant mobile agent execution: nonblocking and exactly-once. Nonblocking ensures that the agent execution can proceed despite a single failure of the agent or the machine, which is undesirable with operations that have side effects. Hence, they propose that fault-tolerant mobile agent execution be modeled as a sequence of agreement problems[10].

4.2.4. *Execution Tracing :*

Execution tracing[8] is a technique for detecting unauthorized modifications of an agent through the faithful recording of the agent's behaviour during its execution on each agent platform. The technique requires each platform involved to create and retain a nonrepudiable log or trace of the operations performed by the agent while resident there, and to submit a cryptographic hash of the trace upon conclusion as a trace summary or fingerprint. A trace is composed of a sequence of statement identifiers and platform signature information. The signature of the platform is needed only for those instructions that depend on interactions with the computational environment maintained by the platform.

4.2.5. *Environmental Key Generation :*

Environmental Key Generation[20] describes a scheme for allowing an agent to take predefined action when some environmental condition is true. The approach centres on constructing agents in such a way that upon encountering an environmental condition (e.g., string match in search), a key is generated, which is used to unlock some executable code cryptographically. The environmental condition is hidden through either a one-way hash or public key encryption of the environmental trigger. The technique ensures that a platform or an observer of the agent cannot uncover the triggering message or response action by directly reading the agent's code.

One weakness of this approach is that a platform that completely controls the agent could simply modify the agent to print out the executable code upon receipt of the trigger, instead of executing it. Another drawback is that an agent platform typically limits the capability of an agent to execute code created dynamically, since it is considered an unsafe operation.

4.2.6. *Computing with Encrypted Functions:*

The goal of Computing with Encrypting

Functions[11] is to determine a method whereby mobile code can safely compute cryptographic primitives, such as a digital signature, even though the code is executed in untrusted computing environments and operates autonomously without interactions with the home platform. The approach is to have the agent platform execute a program embodying an enciphered function without being able to discern the original function; the approach requires differentiation between a function and a program that implements the function.

4.2.7. *Obfuscated Code :*

Hohl[21] gives a detailed overview of the threats stemming from an agent encountering a malicious host as motivation for Blackbox Security. The strategy behind this technique is to scramble the code in such a way that no one is able to gain a complete understanding of its function (*i.e.*, specification and data), or to modify the resulting code without detection. A serious problem with the general technique is that there is no known algorithm or approach for providing Blackbox protection. A time limited variation of Blackbox protection is introduced as a reasonable alternative, whereby the strength of the scrambling does not necessarily imply encryption as with the unqualified one, but relies mainly on obfuscation algorithms.

One serious drawback to this technique is the lack of an approach for quantifying the protection interval provided by the obfuscation algorithm, thus making it difficult to apply in practice.

Another approach is suggested by Min-Hua-Shao and Ziyanying zhou. This approach allows the proper handling of time-sensitive offers and supports the gradual decision-making execution[5].

5. Conclusion

Execution of mobile agents on untrusted platforms is a factor introducing non-trivial security concerns, in particular related to correct agent execution and confidentiality of agent data. A wide variety of techniques for implementing security in agent systems is available. Not all are compatible with one another, nor are they all suitable for most applications. Many of these techniques must be implemented within the framework of the agent system, while a number of them can be applied independently within the context of the application.

While elementary security techniques should prove adequate for a number of agent-based applications, many applications are expected to require a more comprehensive set of mechanisms. Moreover, to meet the needs of a specific application, a flexible framework must exist in which a subset of mechanisms can be selected and applied.

The required security level and security measures must, as always, depend on the application. Current standardisation efforts are likely to facilitate greater use of agent technology in the future. There does not seem to be a single solution to the security problems introduced by mobile agents. Solutions to certain problems do exist, but for mobile agents to be more widely adopted this is an area that requires further research.

6. References

1. Parineeth M. Reddy, "Mobile Agents Intelligent Assistants on the Internet", RESONANCE July (2002).
2. A. Fuggetta, G. P. Picco, and G. Vigna, "Understanding Code Mobility," IEEE Transactions on Software Engineering, 24(5), May (1998).
3. Markus Straßer, Joachim Baumann, Fritz Hohl, "Mole - A Java Based Mobile Agent System," in M. Mühlhäuser (ed.), Special Issues in Object Oriented Programming, Verlag, pp. 301-308 (1997).
4. "ObjectSpace Voyager Core Package Technical Overview," version 1.0, Object Space Inc., December (1997).
5. Min-Hua-Shao, Ziyanying zhou "Protecting mobile-agent data collection against blocking attacks" Computer Standards & Interfaces 28, Issue 5 (June 2006) pp 600-611, 2006.
6. William Farmer, Joshua Guttman, and Vipin Swarup, "Security for Mobile Agents: Authentication and State Appraisal," Proceedings of the 4th European Symposium on Research in Computer Security (ESORICS '96), September, pp. 118-130 (1996).
7. G. Necula and P. Lee, "Safe Kernel Extensions Without Run-Time Checking," Proceedings of the 2nd Symposium on Operating System Design and Implementation (OSDI '96), Seattle, Washington, October, pp. 229-243 (1996).
8. Giovanni Vigna, "Protecting Mobile Agents Through Tracing," Proceedings of the 3rd ECOOP Workshop on Mobile Object Systems, Jyvalskyla, Finland, June (1997).
9. Volker Roth, "Secure Recording of Itineraries Through Cooperating Agents," Proceedings of the ECOOP Workshop on Distributed Object Security and 4th Workshop on Mobile Object Systems: Secure Internet Mobile Computations, pp. 147-154, INRIA, France (1998).
10. Stefan Pleisch, Andre Schiper, "Fault-Tolerant Mobile Agent Execution", IEEE Transactions on Computers Volume 52, February (2003).
11. Thomas Sander and Christian Tschudin, "Protecting Mobile Agents Against Malicious Hosts," in G. Vinga (Ed.), Mobile Agents and Security, Springer Verlag, Lecture Notes in Computer Science No. 1419, (1998).
12. "Trusted Computer System Evaluation Criteria," Department of Defense, CSCSTD-001-83, Library No. S225 711, August (1983).
13. R. Wahbe, S. Lucco, T. Anderson, "Efficient Software-Based Fault Isolation," Proceedings of the Fourteenth ACM Symposium on Operating Systems Principles, ACM SIGOPS Operating Systems Review, pp. 203-216, December (1993).
14. John K. Ousterhout, "Scripting: Higher-Level Programming for the 21st Century," IEEE Computer, March, pp. 23-30 (1998).
15. Neeran Karnik, "Security in Mobile Agent Systems," Ph.D. Dissertation, Department of Computer Science, University of Minnesota, October (1998).
16. Oscar Esparza, Miguel Soriano, Jose L. Munoz, Jordi Forne, "A protocol for detecting malicious hosts based on limiting the execution time of mobile agents", Proceedings of the Eighth IEEE International Symposium on Computers and Commu-

nications, pp. 251 (2003).

17. Bennet S. Yee, "A Sanctuary for Mobile Agents," Technical Report CS97-537, University of California in San Diego, April 28, (1997).
18. Stuart Haber and Scott Stornetta, "How to Time-Stamp a Digital Document", Journal of Cryptology, vol. 3, pp. 99-111, (1991).
19. F.B. Schneider, "Towards Fault-Tolerant and Secure Agentry," Proceedings 11th International Workshop on Distributed Algorithms, Saarbucken, Germany, September (1997).
20. James Riordan and Bruce Schneier, "Environmental Key Generation Towards Clueless Agents," G. Vinga (Ed.), Mobile Agents and Security, Springer - Verlag, Lecture Notes in Computer Science No. 1419, (1998).
21. Fritz Hohl, "Time Limited Blackbox Security: Protecting Mobile Agents From Malicious Hosts," G. Vinga (Ed.), Mobile Agents and Security, pp. 92-113, Springer-Verlag, Lecture Notes in Computer Science No. 1419 (1998).
22. Grzegorz Czajkowski and Thorsten von Eicken, "JRes: A Resource Accounting Interface for Java," ACM Conference on Object Oriented Languages and Systems (OOPSLA), Vancouver, Canada, October (1998).