Evaluating Mobile Agent Platform Security

Axel Bürkle, Alice Hertel, Wilmuth Müller, and Martin Wieser

Fraunhofer Institute for Information and Data Processing,
Fraunhoferstraße 1, 76131 Karlsruhe, Germany
{axel.buerkle, alice.hertel, wilmuth.mueller,
martin.wieser}@iitb.fraunhofer.de

Abstract. Agent mobility requires additional security standards. While the theoretical aspects of mobile agent security have been widely studied, there are few studies about the security standards of current agent platforms. In this paper, test cases are proposed to assess agent platform security. These tests focus on malicious agents trying to attack other agents or the agency. Currently, they have been carried out for two agent platforms: JADE and SeMoA. These tests show which of the known theoretical security problems are relevant in practice. Furthermore, they reveal how these problems were addressed by the respective platform and what security flaws are present.

1 Introduction

Over the past years software agents in general and mobile agents in particular have become more and more important in many areas of computer science such as distributed systems, autonomous systems and robotics as well as artificial intelligence.

Mobile agents are a special sort of software agents that possess the ability to migrate to other hosts. In contrast to classical distributed systems, where processes are bound to the host they were launched on, mobile agents can transfer their code and context to another host where their execution continues.

Especially the technical advantages of mobile agents such as delegation of tasks, asynchronous processing, adaptable service interfaces, and code shipping versus data shipping provide an interesting approach to distributed systems [1].

Application domains where mobile agents have proven to be valuable are telecommunication applications [2], IP-network configuration and management [3], sensor networks [4], electronic commerce, information retrieval [5], [6], mobile computing, and dynamic deployment of software, just to mention a few.

Despite these proven demonstrations of valuable contributions for building systems and applications in different domains, the number of commercial applications built with mobile agents is still rather small. One of the principle reasons for this is security concerns. Companies and individuals are skeptical of allowing an uncontrollable piece of code to appear on their machines and execute, which is basically the same as what a virus does [7]. Agent users are worried about the confidentiality and integrity of sensitive data carried by the agent, e.g. an

K. Fischer et al. (Eds.): MATES 2006, LNAI 4196, pp. 159–171, 2006.

electronic purse in an e-commerce application, when the agent is running on a remote and potentially malicious platform.

All application domains mentioned above demand guarantees of agent behavior and safe interaction with the underlying operating system and other involved legacy systems.

In the last decade a number of publications focused on security aspects of mobile code in general and mobile agents in particular. Some of them analyze theoretical security problems [8], [9], [10] while others propose mechanisms and architectures to overcome these problems [11], [12], [13], [14].

While the theoretical aspects have been widely studied, there is a lack of studies about the practical realization of security concepts in available agent platforms. This paper introduces test cases for the evaluation of security mechanisms of mobile agent platforms. They are based on the results of theoretical studies on mobile agent security and designed with respect to practical relevance.

The agent plattforms covered in this paper are JADE (Java Agent DEvelopment Framework) [15], an open source software distributed by Telecom Italia, and SeMoA (Secure Mobile Agents) [16], a freely available agent platform developed and distributed by Fraunhofer IGD. SeMoA was specially designed with focus on security aspects of mobile agents.

The paper is organized as follows: in Sect. 2, a taxonomy of possible attacks is introduced. Sects. 3 and 4 present test cases and results for JADE and SeMoA. Finally, Sect. 5 discusses the findings of our study.

2 Taxonomy of Possible Attacks

With mobile agents there are four different sorts of attacks according to whether the agent or the agency is malicious and whether the agent or the agency is attacked [1], [9].

2.1 Malicious Agents Attacking the Hosting Agency

- Denial of Service (DoS) attacks: The agent excessively consumes the resources of the agency such as memory, CPU cycles or bandwidth so that the agency is not able to provide its services to other agents.
- Unauthorized access to the agency's data: The agent tries to access confidential data, e.g. keystores or policy files (usually, these are locally stored on the agency's hard disk(s)), or tries to manipulate the agency's management mechanisms.
- Masquerading: The agent masquerades as another agent with more permissions and gains access to sensitive data or services.
- Complex attacks through cooperation with other agents [17]

2.2 Malicious Agents Attacking Other Agents (Located on the Same Agency)

- Changing the other agent's state or task
- Reading or manipulating the other agents' data

- Masking its identity to deceive other agents and gain sensitive information from them or using services on behalf of other agents without paying
- Retarding another agent or detaining it from fulfilling its task [17]
- Denial of Service attacks on other agents by sending spam messages

2.3 Malicious Agencies Attacking Other Agencies

- Eaves-dropping on the communication between two agencies and capturing agents to extract useful information from the agents' state or code
- Traffic analysis attempting to find patterns in the communication between two agencies to derive assumed behaviors based on these patterns
- Sending an agent to attack the agency - this could be either a malicious agent or an agent manipulated to act maliciously

2.4 Malicious Agencies Attacking Agents

- Accessing the agent's data: Reading confidential data, e.g. private keys, or manipulating the collected data
- Accessing the agent's code and / or workflow: Reading the migration path or the algorithms; permanently or temporarily changing the agent's behavior for the benefit of the malicious agency or to damage of other agencies [1].
- Delaying or even denying the agent's execution.
- Cut-and-Paste attack: The agency cuts data items from the agent and pastes it into a new agent. If this data is encrypted, the agent can migrate to the agency where a decryption is possible and come back with the decrypted data.

2.5 Security Solutions

There are multiple mechanisms for providing security against the above mentioned attacks. However, to list them all is out of the scope of this paper, so we refer to [1], [14], [18] and present only the most important security solutions:

- Encryption: The data the agent is carrying, or even the whole agent, is encrypted to prevent unauthorized access to data.
- Digital signatures and certificates: Using PKI (Public Key Infrastructure) is the most common way for communication partners to authenticate against one another, i.e. agent against agency and vice versa.
- Central management of security mechanisms and access authorizations: Each agency has its own security management where authentication and authorization of incoming agents and the monitoring of their actions during their stay is managed.
- Sandbox: Every agent is executed in a secure environment and any attempt to access anything outside this environment is strictly controlled by a security manager.
- Secure Socket Layer (SSL): This technique provides a secure way for mobile agents to migrate from agency to agency.

2.6 Test Purposes

From the above mentioned attacks we chose to consider only those initiated by malicious agents, as we did not intend to modify any code of the platforms. Consequently we disregarded the case of malicious agencies. Furthermore, we did not analyze the security mechanisms of the underlying operating system, the JVM (Java Virtual Machine) or the network. Neither did we examine the security of encryption algorithms or signatures. Our purpose was to test existing platforms for mobile agents on their ability to ensure security in practice.

So far, our tests were carried out for the platforms JADE and SeMoA. JADE has been chosen since it is probably the most widely used agent platform today. SeMoA was considered because of its focus on security. It was specially designed with respect to security and seems to provide the most elaborate security concept. It is interesting to see how these two platforms compare.

All test cases are based on the before described taxonomy of attacks. However, they were adapted to the individual platform in order to meet the peculiarity of each platform. E.g. JADE and SeMoA greatly differ in their architecture and their security concepts. Instead of developing a homogeneous test methodology for all platforms we considered it more appropriate to approach platform security from a practical point of view. The following sections present the test cases and test results for JADE and SeMoA.

3 JADE Security Test

JADE provides security features through an add-on [19]. It comprises the following services:

- **SecurityService**: Authentication using the corresponding Java functionality.
- **PermissionService**: Granting permissions to access Java libraries (using the Java Authentication and Authorization Service JAAS) and to perform agent-specific actions (e.g. start, kill or clone or send-to agents).
- **SignatureService**: Signing of messages to avoid falsification.
- **EncryptionService**: Encryption of messages to avoid unauthorized reading.

To test the JADE security add-on (we tested "Version 3[1].3" with JADE Version 3.3), especially in conjunction with mobile agents, we implemented a specific test environment. This test environment uses the JADE test suite tool [20] to manage the execution of the test cases. Test cases are small pieces of code to test a specific behavior of the security add-on; a sequence of logically related test cases is called a test suite. The JADE test suite tool provides a graphical user interface for configuring and starting test suites and base classes for test suite and test case agents.

3.1 Test Agents

The JADE security test environment comprises the following agents:

- a test suite agent for each test suite to parameterize and start the test cases, and to clean up the platform after test case execution,
- a test case agent for every test case which executes the test procedure step by step,
- a database access agent, which represents the normal application part and provides a service to read metadata of an image from a database according to the read conditions contained in the read request, and
- a user agent, which is able to represent a regular user agent who retrieves image data using the service of the database access agent, as well as a malicious user agent who tries to disturb the regular users and to attack the platform. The user agent is generated and stimulated by the currently active test case agent to behave as regular or as malicious user.

3.2 Test System

The test system consists of three JADE containers: The test container, where the test suite and test case agents are located, the user container, where the user agents are generated, and the main container, where the default JADE agents (ams, df, rma) and the database access agent are located. The main and the test container on one side, and the user container on the other side, have different owners, so that different owner-specific permissions can be granted within the policy file. Fig. 1 shows the testbed architecture.

3.3 Test Cases

Denial of Service (DoS) Tests. Denial of Service test cases check if malicious mobile agents are able to disturb or even prevent the tasks of regular agents by overloading the CPU or by extensive use of operating system resources. All DoS test cases start with generating a regular user agent in the user container and migrating it to the main container. Then a malicious user agent is started in the user container and migrated to the main container as well. The regular user agent then requests image data from the database access agent and the execution time is measured. Then, if requested by the test case, the malicious user agent is cloned to produce many malicious user agents. The malicious user agent(s) start(s) its (their) work, while the regular user agent executes image requests again. The execution time is measured again and compared with the execution time in the undisturbed case to find out if the DoS attack succeeded. The following test cases for DoS tests were implemented:

- TC1: Recursively cloning malicious agents.
- TC2: Malicious agents try to overload the agency by flooding the ams (agent management system) agent with agent search requests.

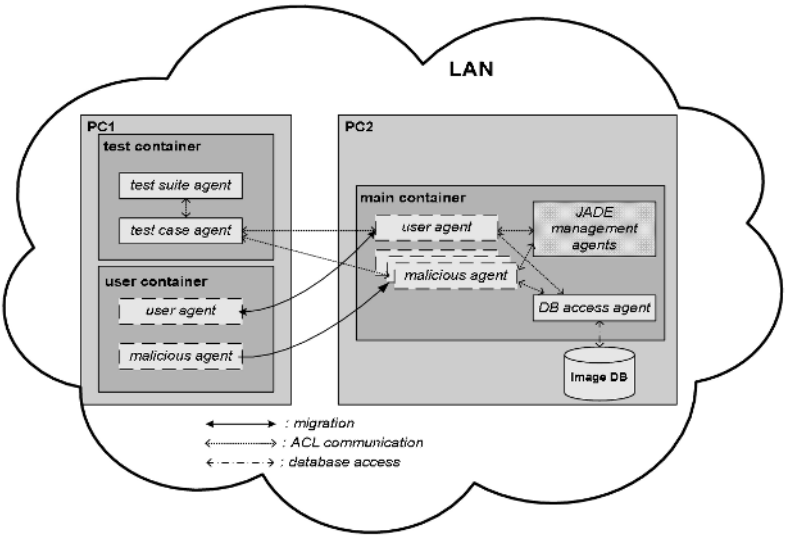

Fig. 1. JADE security testbed

- TC3: Malicious agents activate non-blocking behaviors, resulting in endless loops for many threads.
- TC4: Malicious agents send a message to many receivers, forcing message decoding for all receiving agents at the same time.

Unauthorized Access Tests. Unauthorized access test cases check if mobile agents can access vital functionality of the agent platform or of the runtime environment so that they can sabotage the platform operation or access confidential information. Unauthorized access test cases start with generating a regular user agent in the user container, migrate it to the main container, then a malicious user agent is started in the user container and migrated to the main container as well. After that the malicious user agent tries to attack the platform. The test case ends with removing all user agents from the main container. The following unauthorized access test cases were implemented:

- TC5: Try to modify the policy file.
- TC6: Try to replace the Java security manager.
- TC7: Try to kill another agent.
- TC8: Try to deregister another agent at the ams agent (i.e. deregister from white pages).
- TC9: Try to deregister another agent at the df agent (i.e. deregister from yellow pages).
- TC10: Try to create a new container.
- TC11: Try to kill the JADE platform.

Agent Attack Tests. Agent attack test cases check if malicious mobile agents are able to attack regular agents operating in the same container. The test case procedure is similar to the one used for DoS test cases. The following agent attack test cases are currently implemented:

- TC12: A malicious agent sends a lot of dummy requests to a regular user agent, trying to prevent it from doing its work.
- TC13: A malicious agent sends a lot of spam messages (*inform* messages with no useful content) to a regular user agent, trying to prevent it from doing its work.
- TC14: A malicious agent tries to suspend a regular agent.
- TC15: A malicious agent tries to send a signed message with a fake sender ID.

3.4 Test Results

All test cases have been carried out in four different environments to take into account and study hardware and operating system specific effects. The four test setups were (cf. Fig. 1):

- PC2 was a laptop computer (Pentium M 760, 2.0 GHz, 1 GB RAM) runnig Windows XP Professional
- PC2 was a PC (Pentium D 830, 3.0 GHz, 2 GB RAM) running Windows XP Professional 64 Bit
- PC2 was a PC (Pentium D 830, 3.0 GHz, 2 GB RAM) running SuSE Linux 10.0 (64 Bit)
- PC2 was a HP workstation (PA-8800, 900 MHz, 4 GB RAM) running HP-UX 11.0

The effectiveness of the DoS attacks (TC1 - TC4) and the attacks against other agents (TC12 and TC13) was measured by comparing the execution time of the regular agent in the undisturbed scenario to the execution time when under attack. Fig. 2 shows the mean execution times in milliseconds for 100 iterations of each test case in the respective environment. In case of the spamming attack, the user agent was completely blocked on the laptop. On the workstation, the mean execution time was beyond 30 seconds.

To build the JADE permission service, the JADE security add-on uses the Java security system and adds some agent-specific permission checks. Assuming that the Java security system and the JADE-specific add-ons cannot be corrupted (for example by falsifying the identity of an agent), and if the policy file is edited carefully, most of the described attacks can be prevented, except the following:

- Recursively cloning and non-blocking behaviors cannot be prevented.
- Registering at and deregistering from yellow pages can only be allowed for all agents or for none.

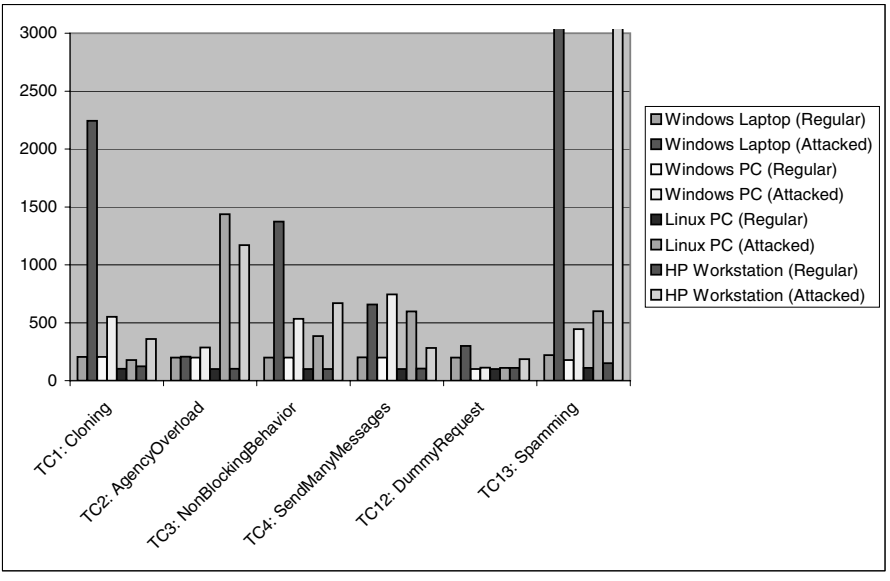

Fig. 2. JADE DoS test results (execution time im msecs)

- Spam and dummy requests cannot be avoided.
- During migration, the ownership of the agent is lost (non-persistent data) and can be easily replaced by an arbitrary ownership, which is used to validate the send-to permission. With the send-to permission obtained by fraud it is possible to deregister any other agent at the df.
- It is possible for an agent to generate a new pair of keys using a fake agent name in order to sign messages with a fake sender ID.

4 SeMoA Security Test

SeMoA provides several mechanisms in order to prevent attacks in terms of a layered security architecture. The first layer is the transport layer, where a protocol such as TLS or SSL can be used for agent transport. In the second layer an incoming or outgoing agent has to pass a pipeline consisting of several security filters. Each filter can accept or reject an agent. Furthermore this layer checks an incoming agent's signatures, decrypts it and finally assigns permissions to it [21]. In this context, Java's permission classes as well as SeMoA-specific classes can be used. An important example for the latter is the EnvironmentPermission. It can be granted for a specific path in an agency's environment and can allow an agent to lookup, publish or retract information or services. The assignment of permissions is done according to a configurable role-based security policy. If an agent tries to execute an action without the corresponding permission, access is denied and an exception is thrown.

After passing the filter pipeline successfully, every agent gets its own class loader. The class loader supports loading classes bundled with the agent and those that are specified as URLs in the agent's static resources. The classes to be loaded are verified against a list of cryptographic hash values signed by the agent's owner. This verification represents layer three. As a fourth security layer, a so-called sandbox is created for each agent: it gets its own threadgroup and is though strictly separated from other agents [21].

Every agent's static part is signed by its owner and the whole agent is signed again by every server that forwards it. Moreover, each agent gets a globally unique "implicit" name which is generated by means of its owner's signature [21].

4.1 Test System

For our tests we used the SeMoA distribution "semoa_complete_050812.zip", released on August 12, 2005. We created a simple agent (the user agent) that migrates from its home agency (AgencyA) to another agency (AgencyB) and requests a database lookup service there (a search for some images in an image database with the images' URLs as a result). After getting the results the agent migrates back to AgencyA and displays the URLs within a GUI. A malicious agent now migrates from a third agency (AgencyC) to AgencyB and attacks either the agency or the reporting agent (Fig. 3). In order to detect the impact of the attacks we launch the user agent in an endless loop and each time quantify the duration it needs to fulfill its task.

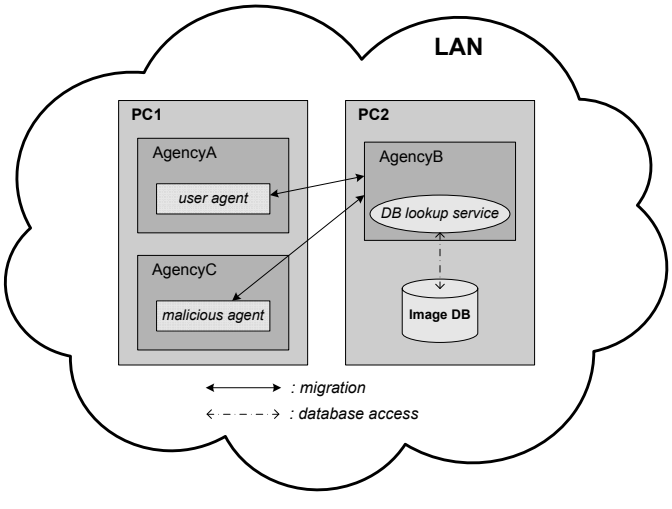

Fig. 3. SeMoA testbed

4.2 Test Cases

Denial of Service (DoS) Tests. In the majority of cases, an agent needs special permissions to be able to attack an agency. We assume that the

agency is not vulnerable unless these permissions are granted to the malicious agent.

Possible targets for DoS attacks are the agency's environment, the agency's computing resources and the Java Virtual Machine.

- TC1: Consuming the agency's computing resources (the malicious agent launches agents from AgencyC to AgencyB; each of these agents starts an endless loop where it just increments a counter).
- TC2: Consuming the agency's memory (nesting AWT-Threads).
- TC3: Spamming the agency with agents (as SeMoA does not provide a cloning mechanism we simulate it in terms of the malicious agent endlessly launching agents from AgencyC to AgencyB).
- TC4: Overloading the agency by too many database accesses (invoking the database access service in an endless loop)
- TC5: Spamming the agency by publishing too many objects (requires an *EnvironmentPermission* with a wildcard for publishing objects or services on AgencyB).
- TC6: Incorrect code (synchronization on the *Thread*-class).
- TC7: Shutdown of the JVM (executing *System.exit(0)*)

Unauthorized Access Tests. The malicious agent migrates to AgencyB and tries to directly access files in the agency's file system using Java's *FileInputStream*. This is possible with a corresponding *FilePermission* which can be granted for read and/or write for individual files or directories.

To access the database on AgencyB the malicious agent can use JDBC or invoke the database lookup service. Either way at least the permissions to access the data and the database driver (again *FilePermissions*) are required.

Another target for unauthorized access are the agency's management mechanisms. SeMoA has several management mechanisms which are located in the agency's environment under specific paths and are therefore only accessible if the *EnvironmentPermissions* for these paths exist. An exception is the policy file which can be manipulated like any file if and only if a corresponding *FilePermission* exists. The most important management mechanisms are:

- The *policy file* which contains all roles and permissions.
- The *vicinity service* which shows all available SeMoA servers in the LAN.
- The *FarSight service* which shows all available SeMoA servers within the network.
- The *ATLAS* (Agent Tracking and Location Service) which serves to track agents; an *ATLAS* client is integrated in each SeMoA server.
- The *security filters*
- The */agents/active* path which contains the contexts of all agents that reside on an agency.

For these management mechanisms we decided to implement the following attacks:

– TC8: Modifying the policy file.
– TC9: Replacing the policy filter.
– TC10: Replacing the Java security manager.
– TC11: Deregistering an agent from the */agents/active* path.

Agent Attack Tests. Attacking an agent in SeMoA is only possible if its implicit name is known. This name can be found either through accessing the agent's context with the *EnvironmentPermission* for the */agents/active* path or if the agent itself publishes its name via a service (if an agents wants to receive messages from other agents it must let these agents know its implicit name, which serves as the agent's address).

– TC12: Spamming the agent with messages / blocking the agent.
– TC13: Manipulating the agent's resources (changing the result of the database lookup).

4.3 Test Results

Fig. 4 shows the effectiveness of the DoS attacks (TC1 - TC5) in four test setups analogous to Sect. 3.4. Note, that for these test cases all necessary permissions have been granted.

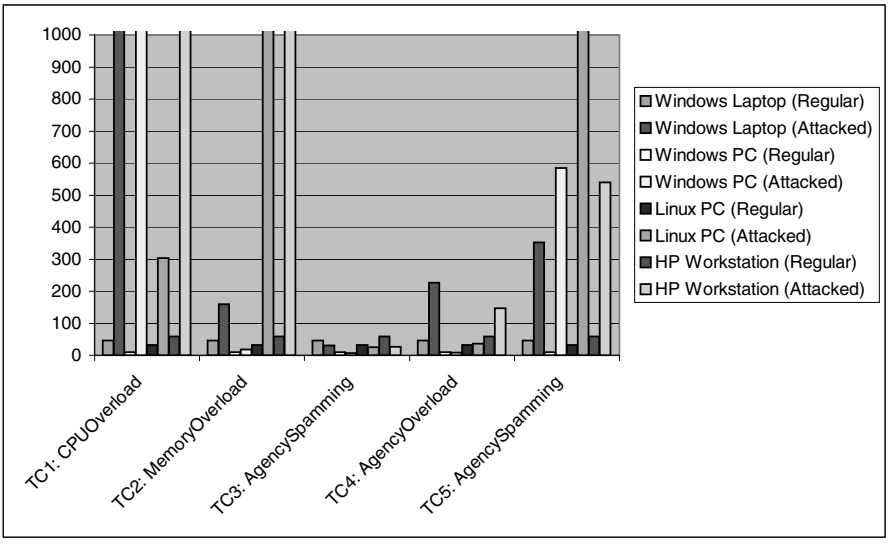

Fig. 4. SeMoA DoS test results (execution time im msecs)

Most attacks can be prevented by correctly configuring the security policy. Critical permissions like *EnvironmentPermission* for the security and management

mechanisms should only be granted to an agency's administrator and permissions for publishing objects in an agency's environment should not be granted with wildcards. Nevertheless, some attacks cannot be avoided:

- Too many agents on one agency are not critical if they terminate soon, but if they execute an endless loop, this heavily burdens the agency.
- The synchronization on the *Thread*-class is a widely known drawback of Java and cannot be prevented.
- The nesting of AWT-Threads is a crucial attack which crashes not only the agency but also the whole operating system. This attack cannot be prevented by the security policy.
- Spamming an agent cannot be prevented if the agent regularly reads its messages (otherwise the sender gets the message "recipient not available"). While the spamming itself does not cause much harm, it can result in blocking the agent, because all messages are stored in the agent's resources. So the agent can become too big to migrate. This can be avoided by emptying the mailbox before migrating.

5 Conclusion

The tests showed that the implemented security mechanisms of the evaluated agent platforms can prevent several of the theoretically possible attacks provided that the access permissions are configured appropriately. Nevertheless, DoS-attacks and spamming cannot be prevented completely. Granting permissions in a very restrictive way avoids some of the DoS-attacks on the SeMoA agent platform. It depends on the application scenario and system environment at hand, as to whether those permissions are necessary for the agents to perform their intended task.

The tests revealed some critical flaws in the security mechanisms of JADE if used in combination with agent mobility. Especially the possibility to fake the owner of an agent when migrating enables several severe attacks and undermines the available security measures. The SeMoA agent platform is well protected against most of the tested attacks, but is inferior with respect to agent communication and cooperation mechanisms.

The next steps will be to examine further agent platforms and define test cases for malicious agency scenarios.

References

1. Braun, P., Rossak, W.: Mobile Agents. Basic Concepts, Mobility Models & the Tracy Toolkit. dpunkt.verlag (2005)
2. Karmouch, A., Magedanz, T., Delgado, J., eds.: Proc. of the 4th Int. Workshop on Mobile Agents for Telecommunication Applications. Volume 2521 of LNCS., Springer (2002)

3. Yang, K., Galis, A., Guo, X., Liu, D.: Rule-Driven Mobile Intelligent Agents for Real-Time Configuration of IP Networks. In: Knowledge-Based Intelligent Information and Engineering Systems: 7th Int. Conf., KES 2003. Volume 2773 of LNCS., Oxford, UK, Springer (2003) 921 – 928
4. Fok, C., Roman, G., Lu, C.: Mobile Agent Middleware for Sensor Networks: An Application Case Study. In: Proc. of Fourth Int. Symposium on Information Processing in Sensor Networks, IEEE CNF 2005 (2005) 382 – 387
5. Brewington, B., Gray, R., Moizumi, K., Kotz, D., Cybenko, G., Rus, D.: Mobile agents in distributed information retrieval. Intelligent Information Agents, Springer (1999)
6. Thati, P., Chang, P., Agha, G.: Crawlets: Agents for high performance web search engine. In Picco, G.P., ed.: Mobile Agents, Proc. of the 5th Int. Conf. (MA 2001). Volume 2240 of LNCS., Atlanta, USA, Springer (2001) 119–134
7. Geirland, J.: The Feature: Mobile Intelligent Agents. `http://www.thefeature.com/article?articleid=26051` (2002)
8. Gray, R., Kotz, D., Cybenko, G., Rus, D.: D'Agents : Security in a Multiple-Language, Mobile-Agent System. In Vigna, G., ed.: Mobile Agents and Security. LNCS, Springer (1998) 154–187
9. Jansen, W., Karygiannis, T.: Mobile Agent Security. Special Publication 800-19, NIST (1999)
10. Roth, V.: Programming Satan's Agents. In Fischer, K., Hutter, D., eds.: Proc. of the 1st Int. Workshop on Secure Mobile Multi-Agent Systems, SEMAS 2001, Elsevier (2002)
11. Hohl, F.: Time Limited Blackbox Security: Protecting Mobile Agents From Malicious Hosts. In: Mobile Agents and Security. Volume 1419 of LNCS., Springer (1998) 92–113
12. Jansen, W.: A Privilege Management Scheme for Mobile Agent Systems. Electronic Notes in Theoretical Computer Science, SEMAS 2001, First International Workshop on Security of Mobile Multiagent Systems **63** (2002)
13. Tschudin, C.: Mobile Agent Security. In Klusch, M., ed.: Intelligent information agents: agent based information discovery and management in the Internet, Chapt. 18, Springer (1999)
14. Vigna, G.: Protecting Mobile Agents Through Tracing. In: Proc. of the 3rd ECOOP Workshop on Mobile Object Systems, Jyvalskyla, Finland (1997)
15. Bellifemine, F., Caire, G., Poggi, A., Rimassa, G.: JADE - A White Paper. `http://jade.tilab.com` (2003)
16. SeMoA. `http://www.semoa.org` (2006)
17. Santana Torrellas, G.: A Network Security Architectural Approach for Systems Integrity using Multi Agent Systems Engineering. Int. Symposium on Parallel Architectures, Algorithms and Networks (ISPAN) (2004)
18. Borselius, N.: Mobile Agent Security. Electronics & Communication Engineering Journal (2002)
19. JADE Board: JADE Security Guide. `http://jade.tilab.com` (2005)
20. Cortese, E., Caire, G., Bochicchio, R.: JADE Test Suite User Guide. `http://jade.tilab.com` (2004)
21. Roth, V., Jalali, M., Pinsdorf, U.: Secure Mobile Agents (SeMoA). `http://www.inigraphics.net/press/brochures/sec_broch/sec/Security.pdf` (2006)