PRACE NAUKOWE AKADEMII EKONOMICZNEJ WE WROCŁAWIU
Nr xxx 2004
Nowoczesne technologie informacyjne w zarządzaniu

Łukasz Nitschke, Marcin Paprzycki, Michał Ren

Adam Mickiewicz University, Poznań, Poland
e-mail: lukn@wmid.amu.edu.pl, marcin@amu.edu.pl, renmich@amu.edu.pl

MOBILE AGENT SECURITY – AN OVERVIEW

Abstract: The aim of this paper is to provide an overview of security issues facing mobile agent systems, and discuss ways of responding to them. We explore the state of the art, to assess if it is mature enough for use in real-life security-critical applications like contract signing or electronic transactions.

1. Introduction to mobile software agents

Software agents are believed to be a promising method of software development [Jennings, 2001]. The benefits are to include faster development, easier maintenance, scalability, and ease of creating distributed systems. Furthermore, the agent approach lends itself naturally to information management (processing and filtering) [Maes, 1994], and to development of systems in which AI in many forms can be implemented [Hendler, 2001, Brenner, 1998]. An agent system consists of agencies (servers with appropriate software which provide the environment for agents' execution) and agents themselves. Agents are programs which, according to Jennings [Jennings, 1998], exhibit certain properties, such as:

- *autonomy* – they act without the need of human input,
- *sociability* – ability to interact with other agents or humans if necessary,
- *reactivity* – ability to react to changes in the environment
- *proactivity* – taking initiative, when appropriate, in order to reach objectives

This definition of software agents is very broad, and encompasses such programs as the animated paperclip in Word, some computer viruses, bots in first-person shooter games, auction agents in online auction sites, or search engine' web spiders. More interesting are so-called strong agents, which according to Jennings [Jennings, 1995] have additional properties, such as: veracity, benevolence and

rationality (however, the list of properties used to define software agents is much broader and lacks consensus [Galant, 2001]). Finally, mobility – the ability to move from one server to another, is believed to be particularly useful as it allows the agent's user to be offline and unavailable while the agent is working on other computers. It also serves to decrease network load – bandwidth-intensive tasks can be performed locally. Finally, it allows for load balancing and resource management in a grid-type environment, where agents are means of carrying computationally intensive tasks [Di Martino, 2003]. Unfortunately, as this paper shows, mobility is extremely challenging from the security standpoint. It seems that security of mobile agent systems is one of the factors that prevents their actual use in e-commerce applications (for an in-depth discussion of current issues involving state of the art in agent system development, see [Paprzycki, 2003]).

2. Cryptographic goals and tools for mobile agent systems

It might seem that the main part of cryptography is encryption. However, in many circumstances (for example in mobile agent systems), other considerations turn out to be equally important. Let us consider the classic example of an air-fare search agent. The agent moves from host to host (airline servers), searching for the optimal offer (apart from the price, the agent can take into account such things as quality of service, or convenience of the itinerary).

In this scenario, we need an assurance that the agent-code and data carried by the agent have not been maliciously modified and that they have not fallen into the wrong hands. Additionally, both parties (agent and server) may be interested in the other's identity; furthermore, they may request a proof of it. Therefore, it is obvious that mobile agent systems must meet the following goals, most often referred to in cryptography as [Menezes, 1996]:

- ***confidentiality*** – keeping information secret from all but those who are authorized to see it,
- ***authentication*** – corroboration of the identity of an entity,
- ***integrity*** – ensuring that information has not been altered by unauthorized or unknown means.

To achieve these goals cryptography provides us with many useful tools such as:

- ***hashing functions*** – a computationally efficient functions that map strings of arbitrary bit length to some fixed length hash-values; hash functions are hard to invert and designed in such a way that it is extremely difficult to find two different strings which yield the same hash-value,
- ***Message Authentication Code*** (***MAC***) ***mechanism*** – a set of hashing functions indexed by values from the set K – $H=\{h_k: k \in K\}$; here each $k \in K$ can be used to produce an authentication value of message m: $h_k(m)$ (MAC-value), which can be verified only by someone who knows k,

- ***symmetric and asymmetric encryption systems*** – consist of two types of transformations: E – encrypting and D – decrypting; every transformation is determined by a value called "the key;" in symmetric encryption systems keys for the inverse transformations ($D(E(m))=m$) are the same or trivially easy to compute, as opposed to asymmetric encryption where the keys are different and the decryption key is hard to compute from the value of the encryption key – in this case we speak of public (encryption) and private (decryption) keys,
- ***digital signature schemes*** – similar to asymmetric encryption, involve two types of keys and two types of transformations: S – signing transformations determined by private keys and V – verifying transformations determined by public keys; $S_s(m)$ – signature under message m created using key private s, can be verified by a public function (determined by a public key v) V_v which simply "says" whether the signature is valid or not.

While a number of methods is available to support cryptographic needs, their successful application to mobile agent systems is not easy. Let us now look into the issues involved in this particular case in some more detail.

3. Mobile agent platform security

The above presented air-fare agent scenario can help us specify threats to the security of the host. The agent (sent by the competing airline, for instance) may spy on the host's databases, or disable services (e.g. performing a denial of service attack). In other words, it might act as a Trojan horse. To achieve this, the agent may pretend to be legitimate, i.e. sent by a customer or, more seriously, to be a part of the airline's agent system itself. Moreover, a benevolent agent may be corrupted by a third party before arriving at the airline's platform.

There is a whole range of options available to hosts to safeguard against such attacks, from simple to sophisticated [Loureiro, 2000]:

1) ***Sandboxing*** – one of the oldest methods of limiting resources available to mobile code, dating back to Java JDK 1.0. The code is executed in a virtual, very restricted machine, therefore preventing access to vital system resources. In some applications, for example when the code needs access to the file system or the network, sandboxing proves to be too inflexible.

2) ***Code signing*** – provides a method of distinguishing trusted and untrusted code using the mechanism of digital signatures. Trusted code can be granted access to critical system resources. Code signing was implemented, for instance, in Microsoft ActiveX and Java JDK 1.1.

3) ***Access control*** – takes code signing one step further, by allowing the owner of the executing platform to precisely define security policies. Instead of dividing code into broad categories of "trusted" and "untrusted", code is associated with the identity of its owner and granted appropriate, individual access privileges. Access

control was first implemented in Java JDK 1.2, and grew into a mature form in JAAS (Java Authentication and Authorization Services). Unfortunately, access control imposes a significant runtime overhead.

4) ***Proof-carrying code*** – avoids expensive runtime checks, by performing code checking only once, before execution. The code carries a proof of its good behavior. So far, this approach has not been implemented in practice and remains a purely theoretical one.

One can also imagine that instead of attacking the platform directly, agents try to attack other agents residing in it. For example, an agent may try to disable, destroy, or subvert agents of other customers. However, protection against such threats is really just another aspect of platform security, as it is the agency that has to protect the executing agents from one another.

4. Mobile agent security

Of course, it is not only the agency that has to include safeguard against hostile agents; agents themselves are exposed to attacks by a hostile agency as well. For example, if the agency belongs to an airline, it might try to brainwash the air-fare agents so they forget all competing (presumably better) offers. If the agent is empowered to accept an offer if it is "good enough," the platform might try to disassemble it and find its threshold value. In general, the more the agent is authorized to do (while representing its owner), the greater the risk that at least one agency on its route will try to subvert it. For instance, we could have authorized the agent to make actual payments or to digitally sign contracts. In order to do that, the agent must carry with it a "secret-token," such as the signing key, the credit card number, or even digital money. It is not difficult to see that stealing one of these may be disastrous to the agent's owner, and very lucrative to the thief.

To protect agents against malicious hosts, we need to ensure that the system has the following properties:

1) ***Privacy of computation*** [Sander, 1997, 1998] – which means that an agent is able to carry out computations without the host understanding what the agent is doing. This would allow the agent to carry secret values (signing key, e-money, credit card number) in an encrypted form but involve them in computations. To extract the decrypting code from the agent the attacker would have to analyze the code's meaning. The following solutions attempt to make this analysis infeasible:

a) **theoretical solutions**:
   i) *function encryption* – in case an agent has to calculate a security critical function f on argument x ($y=f(x)$) in a untrusted environment, it is for security reasons suggested that it should compute an encrypted form of the function – a superposition of functions g and f – $g(f(x))$; y can be retrieved in trusted environment using g^{-1}: $y = g^{-1}(g(f(x)))$;

security of the scheme is based on the difficulty of decomposition of the complex function; this model was applied only to rational functions so far and it was used to create a digital signature scheme for mobile code,

ii) *homomorphic rings* – agent uses a homomorphic function in two algebraic rings $h: R_1 \rightarrow R_2$; h is hard to reverse without knowledge of a secret key; agent makes his calculations on an untrusted server in ring R_2, and the final result is obtained on a trusted host using h^{-1}; it was shown that this method can be used to encrypt polynomials,

iii) *boolean circuts* – every function f on any number of inputs $f(x_1, x_2, \ldots, x_n)$ can be represented as a boolean circuit, and protocols exist that enable evaluation of such circuits in a distributed way, while keeping every participant unaware of all the inputs except for the ones belonging to him [Tate, 2003; Beaver, 1990], so if there are several agent present on different platforms, they can jointly compute a signature or any other function without revealing it to any one platform; in fact, the protocols allow even for a number of malicious platforms to cooperate,

b) **code obfuscators** – consists of scrambling the executable code making it difficult to understand and reverse engineer; a number of such tools is available for Java bytecode, and they use the following techniques [Hongying, 2001]:

i) *layout obfuscation* – renaming identifiers (e.g. methods loose their original names), and removing debug information (code execution cannot be introspected in debug mode),

ii) *control obfuscation* – altering execution flow by adding artificial branches,

iii) *data obfuscation* – reorganizing data structures.

A serious weakness of this method is the fact that it does not provide provable security; in fact, there is a constant race between obfuscators and disassemblers, and although it seems that so far the general-purpose disassemblers are outclassed by obfuscators, the situation may change radically as soon as specialized deobfuscating disassemblers appear.

2) ***Integrity of computation*** – gives a guarantee that the execution flow of agent code was not manipulated from outside of the agent. Attempts to provide this feature are holographic proofs. Here, the trace of the execution, showing not only the results, but also how they were obtained, is transformed into a holographic proof (which can be quickly, probabilistically checked) and appended to the agent.

3) ***Privacy and integrity of data*** – assurance that the data carried by the mobile agent has not been tampered, this applies also to the agent's itinerary, which can be seen as part of the data. An elegant solution, which meets this requirement, was presented in [Yee, 1997] in the form of PRACs (Partial Results Authentication

Codes). Every agent before leaving its home platform is supplied with a vector of keys. Every single key is used to create a MAC of the information gathered or computed on a certain server, and optionally to encrypt the data. The key is forgotten afterwards, preventing subsequent servers on the agent's path from tampering with gathered information. PRACs are used to preserve the integrity of dynamic data. Static, unchangeable data (i.e. agent's identity or itinerary), may be simply protected by a digital signature of the agent's owner. An attacker intending to change the agent's path without changing its identity would have to break the digital signature scheme. These two ideas of securing static and dynamic data were successfully implemented in Semoa agent platform [Roth, 2002].

Another, similar solution is giving the agent a public key, while the owner retains the private key. The agent may encrypt the information it collects with this public key, so that it can not decrypt it later. This ensures that nobody is able to cheat the agent, and pretend to be the home platform or the agent's owner. Unfortunately, public key cryptography is inefficient as compared to PRACs.

Apart from the solutions mentioned above, there are a few interesting attempts to address the above mentioned problems by utilizing cooperating agents. This approach makes an assumption that not all servers are malicious and not all of the corrupted ones want to collaborate with one another. It seems reasonable in a network of competing companies. Therefore, it is vital that the cooperating agents are scattered among the system nodes rather than located on the same server.

Roth in [Roth, 1999] proposes a scenario, in which agents work in pairs – let us call them agent *A* and *B*. *A* visits a set of hosts *H* of airlines, while his partner moves to a server which belongs to a rival of *H*. *A* walks the predefined path and passes the offers to *B*. Once *A* has collected all the information they can choose the best offer; moreover, they can pay for it using e-money shared by them using a secret sharing scheme[1] – *B* can send his part of the e-coin.

Authors of [Page, 2004] show that we can attempt to build self-supporting communities of agents, so that every member of the community has at least two guards of its security – the Shared Security Buddy Model.

5. Social threats to mobile agents

Finally, there are threats to mobile agents that stem from how the interactions between the agents and agencies are set up in the real world. One such threat is that it is the agency that is executing the agent-code, and that it is the agency that ultimately provides all the input that the agent receives. If an air-fare agent has a hardwired threshold below which it will accept an offer immediately, it does not

[1] Cryptographic mechanism which allows to split confidential data into pieces which separately reveal no information about the data and which can be used to recover the secret

help if it is encrypted and obfuscated. A fraudulent agency may simply execute the agent several times, each time with a different set of offers. Providing agents with fabricated environments is known as a Cartesian deception. Several ways of dealing with this problem have been developed:

1) ***High Security Modules (HSMs)*** – every platform must have a secure hardware device that cannot be tampered with, has a very restricted set of inputs and outputs, and is capable of performing cryptographic operations. Such a device can then be involved in interactions between the agent and the agency, ensuring for instance that they are performed only once. Unfortunately, such a solution is expensive, and implies trust in the maker of the HSM.

2) ***Clueless agents*** – it is possible to create agents that do not know their intended purpose [Riordan, 1998]. As an example consider an agent that performs a patent search by calculating hashes of strings, and trying to match them with a stored value. Here, if a certain string is in the database, it will be found, but one can not derive it from the agent beforehand, so the patent idea stays safe. Many variations are possible, including perfect obfuscation of agent's purpose until a condition is met. Note that this approach may give rise to new type of hard to defeat viruses and worms, which will not reveal their payload until they infect a system with a certain domain name, or a certain thing appears for sale on eBay. Unfortunately, clueless agents tend to be rather inefficient, and not every problem can be solved by them.

3) ***Agent/platform networks*** – certain real-world situations lend themselves naturally to creation of a network of agents which, while not cooperating per se, are able to communicate, and would be more resistant to corruption. Consider, for example a network of personal assistants, which all keep track of movies that their owners like. In order to get a recommendation, one could have his assistant to question assistants of people who fit a certain profile (liked the same movies). Even if a fraction of agents maliciously cheat (for example to promote a certain movie), the net effect is mitigated by the honest ones. Unfortunately, since creation of agents is not expensive, such networks are susceptible to corruption by masses of agents generated by one person and acting in concert. There are some possible solutions: accepting new members only by invitation (which defeats the purpose of open exchange of information), creating trust-networks or ensuring that a human spends some time before releasing next agent, by giving out a test that only a human can pass [Ahn, 2003] (which is not going to deter a determined adversary).

4) ***Dummy agents*** – certain forms of corruption of agent platforms can be detected like corruption of organizations in the real world – by undercover agents. An agent may pretend to represent a customer, and searching for the best air-fare price, but it might contain certain data, that should never change if the agencies are benevolent. If the data does change, one can be sure that at least one agency on the agent's path has acted malevolently. It is then easy to isolate the rogue agency by using more such agents with different paths.

6. Conclusions

In this paper we have attempted to summarize the main problems involved in assuring security of agent-based systems as well as state-of-the-art in solving them. We have found that as far as host security is concerned, problems appear to be almost solved. The only remaining issue is to make the existing solutions more efficient. Agent (mobile code) security, on the other hand, remains an open problem. Its nature lies in the character of the mobile code itself – it has to be run by untrusted hosts, which have to be able to trace every executed instruction in order to provide access control. The current weaknesses in the area of privacy and integrity of computation seem to preclude use of agents for truly security sensitive tasks, such as signing of contracts, electronic payments, and so on. Only some tasks can be securely performed by agents – for example information-gathering or database searches. This may change, if the promising idea of cooperating agents is developed further. From the practical point of view, even the simple database search can only be performed securely if the platform supports at least some security mechanisms, like PRACs in case of Semoa. Unfortunately, the most popular agent platforms such as Jade, Grasshopper, or Aglets were not built with agent security in mind.

Bibliography

Jennings N. R. (2001) *An agent-based approach for building complex software systems*, Communications of the ACM, 44 (4), pp. 35-41

Maes P. (1994) *Agents that Reduce Work and Information Overload*, Communications of the ACM, 37(7), pp. 31-40

Hendler J. (2001) *Agents and Semantic Web*, IEEE Intelligent Systems Journal, 16 (2), pp. 30-37

Brenner W., Zarnekow R., Wittig H., Schubert C. (1998) Intelligent Software Agents, Springer-Verlag

Jennings N. R., Wooldridge M. (1995) *Intelligent Agents: Theory and Practice.* The Knowledge Engineering Review, pp. 115–152

Jennings N. R., Wooldridge M. (1998) Applications Of Intelligent Agents, Springer-Verlag NY, Agent technology: foundations, applications, and markets, pp. 3-28

Galant V., Tyburcy J. (2001) *Intelligentny Agent Programowy*, Prace Naukowe AE Wrocław, Nr 891, pp. 46 – 57, in Polish.

Di Martino B., Rana O.F.(2003) *Grid Performance and Resource Management using Mobile Agents*, in: Getov, V. et. al. (eds.) Performance Analysis and Grid Computing, Kluwer, 2003

Paprzycki M., Abraham, A. (2003) *Agent Systems Today: Methodological Considerations*, Proceedings of the 2003 International Conference on Management of e-Commerce and e-Government, Jangxi Science and Technology Press, Nanchang, China, pp. 416-421

Menezes A., van Oorschot P., Vanstone S. (1996) *Handbook of Applied Cryptography*, CRC Press, CRC Press, Boca Raton, USA

Loureiro S., Molva R., Roudier Y. (2000) *Mobile Code Security*, Proceedings of ISYPAR 2000 (4ème Ecole d'Informatique des Systèmes Parallèles et Répartis), Code Mobile

Kirn S., Petsch M., Lees B. (1999) *Intelligent Software Agents: Security Issues of a New Technology*, Information Security Management: Global Challenges in the New Millennium, Dhillon G. ed., ch. 11,

Sander T., Tschudin C. (1997) *Towards Mobile Cryptography*, International Computer Science Institute technical report 97-049

Sander T., Tschudin C. (1998) *Protecting Mobile Agents Against Malicious Hosts.* Lecture Notes in Computer Science 1419, pp. 44

Tate S., Xu K. (2003) *On Garbled Circuits and Constant Round Secure Function Evaluation,* Computer Privacy and Security Lab, Department of Computer Science, University of North Texas, Technical Report 2003-02

Beaver D., Micali S., Rogaway P. (1990) *The round complexity of secure protocols*, Proceedings of the twenty-second annual ACM symposium on Theory of computing, pp. 503-513

Hongying L. (2001) *A comparative survey of Java obfuscators available on the internet*, http://www.cs.auckland.ac.nz/~cthombor/Students/hlai/

Yee, B. (1997) *A sanctuary for mobile agents*. Technical Report CS97-537, Department of Computer Science and Engineering, UC San Diego,

Roth V. (2002) *Empowering mobile software agents*, Proceedings of 6th IEEE Mobile Agents Conference, Suri N, ed., Lecture Notes in Computer Science, vol. 2535 pp. 47–63

Roth V. (1999) *Mutual protection of co-operating agents*, Vitek J., Jensen C. eds., Secure Internet programming: security issues for mobile and distributed objects, Lecture Notes In Computer Science vol. 1603, pp. 275-285

Page, J., Zaslavsky, A., Indrawan, M. (2004) *A Buddy Model of Security for Mobile Agent Communities Operating in Pervasive Scenarios*, Proceedings of Second Australasian Information Security Workshop (AISW2004), pp. 17-25.

Riordan J., Schneier B. (1998) *Environmental Key Generation towards Clueless Agents*, Mobile Agents and Security, G. Vigna, ed., Springer-Verlag, pp. 15-24.

von Ahn L., Blum M., Hopper N., Langford J. (2003) *CAPTCHA: Using Hard AI Problems for Security*, Advances in Cryptology, Eurocrypt 2003

Wilhelm U., Staamann S., Buttyán L. (1998) *Protecting the Itinerary of Mobile Agents.* Proceedings of the ECOOP Workshop on Distributed Object Security

Roth V. (2003) *Cryptographic Protection of Migratory Software*, NebraskaCERT Conference

Binder W., Roth V. (2002) *Secure mobile agent systems using Java – where are we heading?* Proceedings of 17th ACM Symposium on Applied Computing, Special Track on Agents, Interactions, Mobility, and Systems (SAC/AIMS)

Bettelli A. (2003) *The Trustworthiness in Software Agents' Electronic Signatures* Proceedings of LEA 2003: 2nd Workshop on the Law and Electronic Agents, pp. 81-95

Hartvigsen G., Helme A., Johansen S (1995) *A Secure System Architecture for Software Agents: The Virtual Secretary Approach*, BROADCAST (Basic Research On Advanced Distributed Computing: from Algorithms to SysTems) Technical Report

Tate S., Xu K. (2003) *Mobile Agent Security Through Multi-Agent Cryptographic Protocols*, Proceedings of the 4th International Conference on Internet Computing (IC 2003), pp. 462-468

BEZPIECZEŃSTWO SYSTEMÓW MOBILNYCH AGENTÓW – PRZEGLĄD

Streszczenie: Celem niniejszej pracy jest dokonanie przeglądu problemów bezpieczeństwa systemów mobilnych agentów, jak również niektórych metod radzenia sobie z nimi. Dokonaliśmy przeglądu aktualnego stanu wiedzy by ocenić czy jest wystarczająco zaawansowany by sprostać zastosowaniom dla których bezpieczeństwo jest warunkiem sine qua non, takich jak podpisywanie kontraktów albo elektroniczne pieniądze.