Secure Tunnels for Mobile Multi-Agent Systems

S. Pozo, R. M. Gasca, M. T. Gómez-López

Departamento de Lenguajes y Sistemas Informáticos
ETS Ingeniería Informática, Universidad de Sevilla
Avda. Reina Mercedes S/N, 41012 Sevilla, Spain
sergiopozo@us.es, {gasca,mayte}@lsi.us.es
http://www.lsi.us.es/~quivir

Abstract. Nowadays the scientific community is trying to design new techniques in the search for solving security problems in mobile agent technology. There are now some industry initiatives for using agents in real environments and they need a solution for some of their security problems. In this paper we present a proposal for using TCP secure tunnels in order to add confidentiality and integrity to any agent platform, present or future, without making any changes to its source-code. Our system is currently implemented and working as part of a complete mobile multi-agent system. During our research we have also detected many other problems regarding our tunnelling approach, such as tunnel dodging and private IP address routing. Although this approach is working properly, it does not scale well in large and/or changing environments.

1 A new perspective of security in mobile multi-agent systems

Mobile MAS systems can be used with real success in a growing number of applications nowadays. For example, there is a lot of research effort in agent cooperation and negotiation protocols and their applications, such as cooperating agents in problem resolution and negotiating agents in ecommerce.

Most negotiation and cooperation protocols that are widely deployed over big networks or over Internet have been used with mobile multi-agent systems. A lot of these applications also require some degree of security to be usable in production environments and public networks. There are even other kinds of applications with specific requirements imposed by legislation according to security, like the applications used in health care environments.

There are several security threats in the agent paradigm (for both mobile and static agents), which allows classifying them as follows [1]:

- Disclosure of information
- Denial of service (DoS)
- Corruption of information
- Interference or tampering

A new classification could be formed depending on the entity of an agent environment which carries out an attack [1]:

- Agent vs. Platform. This category represents the set of threats through which agents exploit security weaknesses on a platform or launch attacks against it.
- Platform vs. Agent. This category represents the set of threats through which platforms compromise the security of agents.
- Agent vs. Agent. This category includes attacks by agents against other agents, either directly (direct denial of service attacks) or indirectly (by exploiting vulnerabilities of other agents). It should be taken into account that many agent platforms are themselves composed of agents who provide internal services.
- Other entities vs. Platform. This category represents the attacks which can be done against a platform by any other entity (whether this entity is another platform or internal or external agents).

In this paper we present an alternative and working way of protecting agents and platforms using existing tools and technologies in order to create secure tunnels. An in-depth performance evaluation with different use-case sceneries, which has provided us with very surprising results, has also be done [22]. The work is structured as follows: first we present a brief introduction to security vs. mobility tradeoffs and its current research efforts. Then we indicate our approach to the problem and the goals for our system architecture. Finally we propose a secure tunnel arternative to securing multi-hop agent communications. We finish the paper indicating the conclussions, advantages and disadvantages of our solution to the problem.

2 Mobility Vs. Security tradeoffs in mobile multi-agent systems

Mobile software agents are goal-directed programs capable of suspending their execution state on one platform and moving to another, where they resume their execution. Static agents have all the same characteristics except that they are not mobile, i.e., they cannot move from the system they are running on (which is also the system which created them.)

There are different kinds of mobility: strong, in which code and state are moved; and weak, where only the code is moved. That is, every time the agent arrives at different destinations, it runs back from the beginning, or from a marker expressly placed for this purpose, each time it arrives at a different destination.

Mobility can also be classified as single hop and multi-hop. Most security problems which arise in mobile multi-agent paradigm occur with multi-hop mobile agents. This is due to the fact that only the home platform (the platform where the agent was first created) is trusted. That is, if we are not sure that a mobile agent is from a trusted source, in which we know that neither the code nor the state have been altered, we cannot know whether this agent will carry out an unplanned activity on the destination platform.

The most secure location for an agent is its home platform. Although neither agents nor home platforms are invulnerable, a number of conventional techniques can be applied to construct adequate defences. Each time an agent migrates, security risks arise, and so it is needed a way to transmit this trusted environment to other platforms where agents may travel. The greatest problem with multi-hop MAS is just the trust relationship which can be established in single-hop MAS between two platforms due to the security mechanisms derived from Client/Server architecture. These trust relationships are not transitive, nor have to be bilateral.

3 Current threats in mobile agent technology

There are some practical solutions for securing communications in multi-hop MAS, but the vast majority of them include restrictions on itineraries. For example, it is possible to set up a restricted itinerary in which all of the platforms have mutual trust in all of the others, so agents can move freely between them (each platform signs and/or encrypts the agent before it migrates).

There are other alternatives based on the usage of a transport level protocol (usually SSL [21]) to secure communications, and it has been applied with success to some agent platforms [2]. However, some of them are static MAS [3] and all proposals are directed towards a specific MAS.

3.1 Platform protection

- Software-Based Fault Isolation [5]. This method consists on isolating the different modules of an application into different fault domains enforced by software. The aim of this technique is to ensure that programs written in insecure languages, such as C, are executed in a virtual address space restricted to the application, as if it were a closed box within which the application is confined. This technique, commonly referred to as sandboxing, is very efficient compared to others (such as those based on hardware tables), especially when there are many processes running in trusted domains, since these do not incur any overhead.
- Safe Code Interpretation [23]. Many agent platforms have been developed using interpreted or scripting languages (e.g. Java, TCL...). This technique consists of taking advantage of the interpreter stepping through the code to include the ability to detect instructions considered to be "harmful". The most well-known secure code interpreter is probably Safe TCL [24].
- State Appraisal [6]. This technique aims to predict whether an agent has been subverted by inspecting any alterations to its state. These evaluation functions must be designed and implemented before the first agent run. Then they are executed together with the agent. The success of this technique depends mainly on the ability to predict malicious state changes.
- Path Histories [7, 8]. The idea behind this technique is to maintain a record (of guaranteed integrity) with data identifying the platforms the agent has visited, so that when an agent arrives at a platform, this platform can review the list and determine whether to allow the agent to execute, or whether to restrict its access, depending on the platforms the agent has visited.

- Proof Carrying Code [9]. This technique obliges the code producer (the author or the organization which purchases it) to formally prove that the code owns security properties. This is therefore a preventive technique, while the usual techniques are used for detection.

3.2 Agent protection

- Partial Result Encapsulation. The basic idea behind this technique is similar to the state appraisal used for the protection of platforms: the agent encapsulates the result of its actions each time it visits a platform, and this can be examined later by the agent itself or by another entity to detect any anomalies.
- Execution Tracing [10]. This technique detects unauthorized alterations of an agent using a record of the behavior of the agent on the platform. This requires each platform to maintain a log for each agent which executes on that platform, and to provide mechanisms to guarantee the integrity and confidentiality of these logs. If any suspicious result occurs, these logs can be verified to determine exactly where the anomalous behavior occurred and thus detect the malicious platform. The problem is that in order to produce these logs, the platform must collaborate.
- Environmental Key Generation [11]. This technique consists of incorporating a fragment of encrypted code into an agent in such a way that neither the platform nor other agents are aware of it. This code fragment will be executed when a particular condition exists in the execution environment. The idea is to hide part of the agent's code to prevent it from being modified.
- Computing with Encrypted Functions [12]. The aim of this technique is to try to determine a method whereby the mobile code can compute cryptographic primitives (such as digital signatures) securely in an untrusted execution environment with no need for interaction with the home platform. This technique, however, does not solve many of the attacks against agents (such as denial of service, replay, extraction of information and others)
- Obfuscated Code [13]. The strategy behind this technique is very simple: to scramble the agent code in such a way that it is impossible to completely understand (and much less modify) the function of an agent by reading its code. This technique is also known as *blackboxing*.

4 Proposed approach

Nearly all research efforts pointed in the previous chapter require modifications to agent platforms (or even use a different programming language), and most of them to then agent themselves. Others are computationally expensive, such as state appraisal, proof carrying code and partial result encapsulation. Most of them aren’t easy to implement in a viable way (computing with encrypted functions). And nearly all of them need a hard research, design, implementation and test jobs before they can be used in a production (or near-production) environments.

We propose to divide protection techniques into three categories: protecting platforms, protecting agents and protecting communications. Our work is focused in the third category, providing no direct protection to platforms or agents themselves. Our system provides protection to the communication channel and platform authentication between each other.

In this paper we propose a different approach to securing communications. Our main goals were to obtain a secure agent community usable in production environments with existing technology with little impact in network topology, existing agent applications or changing agent platforms. Specifically we were looking for a system that provides:

- Integrity and confidentiality
- Data origin authentication
- Multi-agent system independence
- Compliance with standards
- Existence of cryptographic acceleration hardware
- Cost-saving and reuse of existing software and hardware infrastructure.

As we are going to expose, all of these goals have been accomplished at different levels. Most of the problems encountered with this new secure tunnel approach are common to all of the approaches, so not all approaches until the final solutions are going to be exposed with the same level of detail, due to length constraints. The most important features of the approaches are going to be exposed with more detail through.

Aglets [14] have been used for our reference implementation in the lab, since it is Open Source, cross-platform, easy to develop, has strong mobility (maximum mobility in a Java-coded MAS), high acceptance and relatively good documentation [15].

5 TLS, SSH2 and proprietary-protocol secure tunnels

Tunnelling is the capability of encapsulating one protocol within another, using this second protocol to traverse network nodes. A secure tunnel encapsulates an insecure protocol (like FTP or HTTP) within a secure one (like SSL). Tunnels may also be used to bypass firewalls, and are also vulnerable to denial of service attacks, since they use a public and untrusted network as transmission media.

Due to the independence which the secure communications system must have from any mobile MAS, research has focused on systems which generate application-independent secure tunnels.

5.1 Stunnel

Stunnel [16] is an application which acts as SSLv3 server and/or client, providing a secure SSL-based secure tunnel (wrapper) for insecure protocols or applications with the only need of the installation of the application in each of the systems that needs to secure. Stunnel is distributed under a GPL license and has versions for Microsoft Windows, some flavors of UNIX and many other OS. Stunnel also supports cryptographic accelerator hardware and client and server authentication with X.509 digital certificates (as SSL does).

The way Stunnel works is wrapping ports on the client and server systems (Fig. 1). It wraps the port the real service has assigned at the server (e.g. 4000 in Aglets server to another port, say 4001), and at clients it creates a local port the client service has to communicate with to access the Aglets network service (say 4001). The client connects to its 4001 local port. Then Stunnel wraps the connection and redirects the request to the 4001 port on the server node, which is also a port wrapped by Stunnel. The server-side Stunnel unwraps and redirects data to Aglets platform on port 4000 (which is usually blocked by a firewall to prevent direct external access). Authentication (server and optionally client) also takes place in the process at the beginning of the communication (like in whatever other SSL connection).

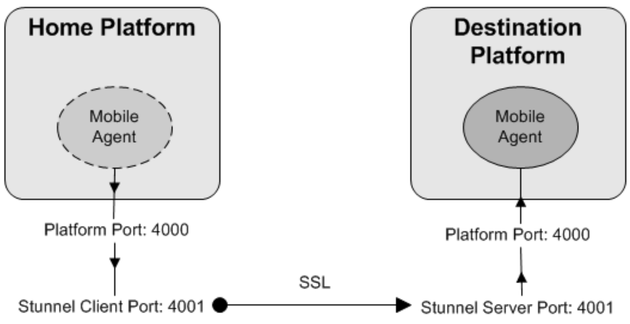

Fig. 1. Example Stunnel port-wrapping mechanism

In the case of a mobile MAS, the server and client ports are the ones where the mobile agent platform resides, although in the agent paradigm does not exist the concept of server and client, we will use them to refer to the platform where the agent dispatches, that is the source or client platform, and the platform where the agent moves to (destination or server platform). These roles of the platforms are always changing in multi-hop mobile MAS, and are inverted from the natural ones of a typical Client/Server architecture (Fig. 1)

A typical mobile multi-agent community can be modeled as some nodes that act as home platforms for the vast majority of agents, and other nodes that play the role of intermediate or visit nodes. The number of visit nodes is usually far bigger than the number of home nodes in a typical multi-agent community. Although any node can create an agent and become a home node, in real applications with multi-agent systems, the community is usually divided into platforms that create the majority of agents (sources) and other platforms that doesn’t create mobile agents (destinations).

We have detected some issues regarding Stunnel, at the same time that other problems have arised during our research with secure tunnels (Table 1).

Table 1. Stunnel features

Feature	Explanation	Can be solved?
UDP tunneling	Stunnel cannot wrap UDP protocol.	No
Node authentication	Stunnel uses digital certificates. A lot of certificates need to be issued.	N/A
Integrity checking	As Stunnel uses SSL protocol, it also uses the same mechanisms of data integrity.	N/A
Port-range mapping	Stunnel cannot map a range of ports in a unique process.	No
Tunnel dodging	Problems with Stunnel and all protocol wrappers in general.	Yes
IP address routing	When using IP addresses as basic authentication mechanism and NAT servers.	Yes

- Node authentication. A feature derived from using SSL is the need to use digital certificates for node-authentication. At least a digital certificate needs to be used in each destination platform. Client-authentication (source nodes) is optional in SSL.

  Another authentication problem is derived from the fact that the role of server and client nodes are inverted from the natural ones of a typical Client/Server architecture, where the lesser number of nodes are the servers (and are the only ones that really needs digital certificates). With the Stunnel approach it is needed to create at least a digital certificate for each destination node of a community; furthermore as the role of each platform is always changing in multi-hop mobile MAS, each node needs a digital certificate. That is not a big problem with small communities, but with bigger or small but easily growing ones, it could be a strong restriction, because we must maintain a community-restricted PKI, and deploy strong authentication mechanisms for nodes to prevent tampering of certificates. A beneficial collateral effect derived from the deployment of strong authentication systems and PKI infrastructure is that no agent coming from external parties can be injected in our community. It is a natural isolation mechanism.

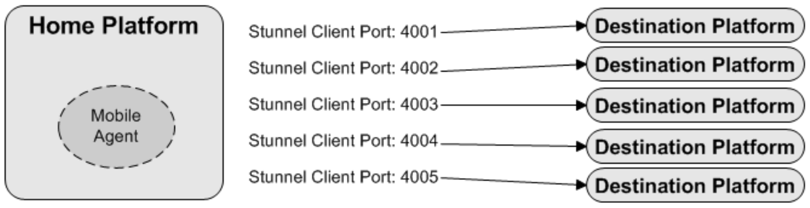
Home Platform
Mobile Agent
Stunnel Client Port: 4001
Stunnel Client Port: 4002
Stunnel Client Port: 4003
Stunnel Client Port: 4004
Stunnel Client Port: 4005
Destination Platform
Destination Platform
Destination Platform
Destination Platform
Destination Platform

Fig. 2. Multiple destination Stunnel setup

- Port mapping. Stunnel communication model have some problems regarding the way it maps ports. Stunnel needs to map separate ports with separate IP addresses for each destination platform in order to be able to establish the tunnels (that is, it cannot map a range of ports or a range of IP addresses). One Stunnel process is needed on each source platform (5Mb) for each destination to which the agents may travel to. On a network with 2000 possible destinations, the RAM required on the each source platform would become unfeasible (10,000 Mb of RAM, or 10Gb). It should be taken into account that the source platform is always client (Fig. 2.)

Another problem is that some agent platforms (such as Aglets) work on a fixed but user-configurable port for agent migration, but it uses a random port from a range for message transmission. Due to design restrictions of Stunnel, this actually means having one Stunnel process for each possible messaging port, which is totally unfeasible for the same reason as before (there could be 1000 ports in the range, and each Stunnel process requires 5Mb). Multiply this quantity of needed RAM to the one obtained from running one more Stunnel process for each possible destination and you could easily need a Terabyte-capable system to run an effective SSL-tunnelled mobile MAS in large communities.

- Tunnel dodging. Since Stunnel is external and transparent to the application which uses it, when an agent is asked for a handler or proxy in order to be able to reference it, the platform gives the address and port of the system where it is residing. It does not give the port where Stunnel is located, but the real platform port, and so the communication, if established, is going to be untrusted (bypassing Stunnel). Some agent platforms provide a proxy mechanism to encapsulate its transmission protocol in another protocol, creating a tunnel for it. This is the case of Aglets, where it is possible to define an HTTP proxy (host and port) to use when transferring agents. That is, Aglets encapsulates ATP over HTTP, and then HTTP requests are encapsulated in SSL through Stunnel and transmitted.
- IP address routing. There is a problem related to private IP addresses, NAT servers and some mobile MAS platforms (such as Aglets). This problem is not related to tunneling itself but to specific agent platform design itself. If the mobile MAS platforms have private IP addresses and are behind NAT servers, the agents need to traverse that NAT server, which is going to change the IP address of the packets to the one the NAT server has. Some mobile MAS hardcode the home platform IP of the home platform in the agent on dispatch. This address is then compared with the one of the packets arriving at the destination platform, and if it is not the same, the agent will be discarded. This is a general problem with NAT and could not be easily resolved because it depends on the design of each mobile MAS platform. Some mobile MAS tries to resolve the hardcoded IP address of dispatched agent, but if the address is in the private range it is not possible to resolve it unless there is an available DNS server that has registered that private IP.

5.2 Secure Shell (SSH2)

SSH [20] is a protocol for securing network services over an insecure network. It is traditionally used for protecting insecure UNIX protocols such as telnet, rlogin, etc. Moreover SSH can be used to secure other services creating a wrapper around them using a local port redirection scheme very similar to the one used by Stunnel. SSH is widely accepted by the scientific community as being a trusted security protocol.

SSH architecture is very similar to SSL or TLS one, and provides basically the same functionality. For that reason we are not going to explain SSH protocol in more detail.

SSH can use different authentication schemes such as preshared secret or digital signatures. It also uses Diffie-Hellman key agreement protocol (like TLS) and HMAC one-way hash function. Different implementations of SSH protocol can use different encryption algorithms. SSH features are summarized in Table 2.

Table 2. SSH (OpenSSH) implementation features

Feature	Explanation	Can be solved?
UDP tunneling	SSH doesn't support UDP.	No
Node authentication	Stunnel uses digital certificates. A lot of certificates need to be issued.	N/A
Integrity checking	SSH uses HMAC hashes.	N/A
Port-range mapping	OpenSSH cannot map a range of ports in a unique process.	No
Tunnel dodging	Problems with SSH and all protocol wrappers in general.	Yes
IP address routing	When using IP addresses as basic authentication mechanism and NAT servers.	Yes

There are several implementations of SSH protocol, ranging from Open Source to commercial ones and for a wide variety of operating systems. Although very similar to Stunnel at protocol level and working way of OpenSSH implementation, this application has been considered for its wide acceptation and use.

5.3 Zebedee

Zebedee [18] is another Open Source application used to create secure tunnels with implementations in Windows, UNIX, Linux, Java and Ruby. Zebedee (from its documentation) has a small memory footprint and low wire protocol overhead.

Zebedee does not use SSL, but a plain Diffie-Hellman protocol for key agreement and a symmetric key cryptographic algorithm, Blowfish [25]. Zebedee does not provide any features for data integrity.

In counterpart, Zebedee solves some of the design and implementation problems of Stunnel, providing us with a more flexible way of creating tunnels. In exchange we get a weak authentication mechanism and no integrity checking. Zebedee also supports compression (through zlib and bzip2 libraries) with a selectable range of speed and efficiency. It doesn't support cryptographic acceleration hardware because it isn't based on standards or libraries supporting crypto hardware.

- Tunneling of UDP protocol. Zebedee can be used to tunnel UDP protocols, but even in UDP-mode the created secure tunnel uses TCP protocol to wrap it.

- Node authentication. Zebedee permits a sort of really weak identity checking mechanism for authentication (it is not possible to use digital certificates for authentication): one based on IP addresses, and the other based on the generation of a private and permanent key, that is used (along with the modulus and generator of the Diffie-Hellman phase) for the generation (usign a hash function not specified) of a public fingerprint which needs to be copied in every Zebedee server it needs to communicate with.

  As neither the private key nor the modulus and generator change, and although Zebedee uses session keys that change regularly, this identity checking scheme is vulnerable to known-plaintext attacks. Zebedee documentation also points that its identity checking mechanism is also vulnerable to man-in-the-middle attacks.
- Port mapping. Zebedee can also map a range of IP addresses and ports for destination platforms in only one process, thus releasing the huge RAM constraint of Stunnel on big agent communities and the random port selection when sending messages in some agent platforms, thus improving the maintenance of the architecture.

Table 3. Zebedee features

Feature	Explanation	Can be solved?
UDP tunneling	Zebedee uses TCP to wrap UDP protocol.	N/A
Node authentication	Really weak authentication scheme based on IP addresses.	No
Integrity checking	Zebedee does not support any integrity checking mechanism.	No
Port-range mapping	Zebedee can map a range of ports in a unique process.	N/A
Tunnel dodging	Problems with Zebedee and all protocol wrappers in general.	Yes
IP address routing	When using IP addresses as basic authentication mechanism and NAT servers.	Yes

In spite of its weak authentication mechanisms, this weakness can be used to simplify security maintenance, as it doesn’t need any private secret to be permanently stored at any platform (it should be remembered that key generation is automatic). Adding nodes to the infrastructure is straightforward, simple and cost-effective, although very insecure. We have to note that a PKI isn’t needed this time.

The lack of standards supported by Zebedee, weak authentication mechanisms, lack of integrity checking mechanisms, and the impossibility to resolve private IP address routing make Zebedee a near-unusable alternative to Stunnel or SSH. Zebedee features are summarized in Table 3.

5.4 Security and availability problems with TCP-based secure tunnels

In addition to the exposed problems in all secure tunnel architectures, whether it is a SSL-based system or a proprietary one, there is another serious security and availability problem related to all TCP-based tunneling systems.

The capability of inferring TCP sequence numbers (commonly named blind spoofing attack) have been a fact for many years [19]. This can be used to inject packets in an open TCP conversation or to send connection termination packets to one or both parties, causing severe delays and in the end a DoS failure [17]. It is possible to carry out the attack injecting packets from a third party before tunnel establishment (although it is difficult to carry on a blind spoofing attack out of a LAN), since TCP-based secure tunnels doesn't authenticate TCP packets until the tunnel has been established.

6 Conclusions

Mobile multi-agent systems introduce a great deal of security issues. Some of them are new and others are very similar to client-server paradigm ones. Most of the solutions are directed towards a concrete mobile MAS or are rather theoretic. Mobile MAS applications can be used nowadays in industrial applications, since there are even commercial MAS platforms. But as programming languages are becoming cross-platform, cross-platform security systems are needed for at least some of the security issues we have explained.

We have proposed in this paper a new approach for protecting agent communications using secure tunnels using existing applications and protocols for their implementation. We were looking for a cheap, multi-platform and generic solution to several mobile MAS platforms. Our work has been focused on protecting communications, but also provides some collateral effects that improve the protection of agents and platforms, creating a secure agent community. Using a key distribution scheme (i.e. a PKI) for our community, guarantees that only agent platforms whose digital certificate have been signed by the certificate authority of the community PKI can communicate with other platforms of the community.

We have done an in depth study of the solutions that secure tunnel applications can provide to existing mobile agent platforms, using Aglets as the reference one. During the study new non-security related problems have also been detected (for example, routing of IP addresses and authentication scheme of some agent platforms). Some of them have been solved, but others are limitations of the applications that have been tested. There are some other applications we have not tested (such as SSLWrap) because they do provide the same functionality as the ones we have presented. We kept with the most representative and actual ones.

The main difficulties we have encountered have been the tunnel dodging problem, which have been solved using a proxy, the private IP address routing problem, which has been solved using a community internal DNS, and finally the port-range problem, which is derived from agent platform design and implementation and cannot be solved without redesigning the implementation of secure tunnel tools. An in-depth performance study has been conducted [22], but not presented here due to length constraints.

The main advantages of our solution compared to the others we have explained in an earlier section are derived mainly from the fact that the secure-tunnel solution is an all-in-one solution providing a lot of improvements in a unique deployment:

- Privacy, integrity and authentication through well know and respected cryptographic protocols
- Cross-platform, cross MAS. This solution can be used in all major OS, and with near all MAS platforms, but the three secure tunnels studied are not interoperable between them
- Does not need code modifications to MAS or agents themselves (cost saving). No modifications are required to MAS platforms or agents. Our solution is straightforward
- It is a standard-compliant solution. Two of the three studied systems used for implementation in the lab are based on standards. These systems use well-known tested cryptographic libraries
- Because of standard compliancy, cryptographic acceleration hardware can be used, such as crypto-accelerated network cards
- A trusted agent community is created between communicating agents. If a new MAS platform is going to communicate with other platforms in the community, it needs a digital certificate signed by the same certification authority (internal to our agent community) which signed all the other nodes certificates

In counterpart, there are some inconveniences in this solution:

- Certificate deployment for authentication could be difficult and a tedious task, as well as reconfiguration of node tunnels each time a new node is enter or leave the community. This issue causes a serious scalability problem. A PKI could be a viable solution.
- Some MAS platforms authentication mechanisms are based on agent IP address, which they hardcode before an agent migrates to another platform. This authentication scheme could cause problems if an agent passes throw a NAT server. This problem can be solved using a DNS server which could be accesses by all MAS platforms, and containing all of the IP addresses used for authentication.

Thus, our approach is only valid for small communities. For big or changing communities our approach is very limited, because of the manual configuration needed at all platforms, the use of a PKI for easy certificate managing, and the architectural problems that prevented us from running multiple application processes because of the linear memory growth with some applications.

But for small communities in well defined environments (i.e. industrial ones), our approach is very useful, since it is very easy to deploy, cheap and usable with almost any agent platform and application with little or no modification.

We've also stated that TCP-based secure tunnels are vulnerable to blind spoofing attacks, since TCP packets aren't authenticated until de tunnel has been established. This is a common problem for all TCP-based tunnels.

7. Acknowledgements

This work was funded in part by the Fifth Framework IST program of the European Commission under the TeleCARE IST-2000-27607 project, and by the Spanish company SKILL Grupo Tecnológico. The authors thank the contribution of the TeleCARE consortium members.

References

1. W. Jansen, T. Karygiannis: Mobile Agent Security. NIST Special Publication 800-19, October 1999.
2. L. M. Silva, P. Simoes, G. Soares, P. Martins, V. Batista, C. Renato, L. Almeida, N. Stohr: JAMES: A Platform for Moble Agents for the Management of Telecommunication Networks. Proceedings Intelligent Agents for Telecommunication Applications, IATA'99, Stockholm, Sweden, August 1999.
3. A. Puliafito, O. Tomarchio: Design and Development of a Practical Security Model for Mobile Agent System. 7th IEEE International Symposium on Computers and Communications, ISCC'02.
4. S. Fischmeister, G. Vigna, R. A. Kemmerer : Evaluating the Security of Three Java-Based Mobile Agent Systems. 5th IEEE International Conference on Mobile Agents (MA'01). Atlanta, Georgia, USA, December 2001.
5. R. Wahbe, S. Lucco, T. Anderson: Efficient Software-Based Fault Isolation, October 1998.
6. W. Farmer, J. Guttman, V. Swarup: Security for Mobile Agents: Authentication and State Appraisal. Proceedings of the 4th European Symposium on Research in Computer Security (ESORICS), September 1996.
7. J. J. Ordille: When Agents Roam, Who Can You Trust? Proceedings of the First Conference on Emerging Technologies and Applications in Communications, May 1996.
8. D. Chess, B. Grosof, C. Harrison, D. Levine, C. Parris, G. Tsudik: Itinerant Agents for Mobile Computing. IEEE Personal Communications, vol.2, no. 5, October 1995.
9. G. Necula, P. Lee: Safe Kernel Extensions Without Run-Time Checking. Proceedings of the 2nd Symposium on Operating System Design and Implementation (OSDI), October 1996.
10. G. Vigna: Protecting Mobile Agents Through Tracing. Proceedings of the 3rd ECOOP Workshop on Mobile Object Systems, June 1997.
11. J. Riordan, B. Schneier: Environmental Key Generation Towards Clueless Agents. Mobile Agents and Security, Springer-Verlag, Lecture Notes in Computer Science No. 1419, 1998.
12. T. Sander, C. Tschdin: Protecting Mobile Agents Against Malicious Hosts. Mobile Agents and Security, Springer-Verlag, Lecture Notes in Computer Science No. 1419, 1998.
13. F. Hohl: Time Limited Blackbox Security: Protecting Mobile Agents From Malicious Hosts. Mobile Agents and Security, Springer-Verlag, Lecture Notes in Computer Science No. 1419, 1998.
14. D. B. Lange, M. Oshima: Programming and Deploying Java Mobile Agents with Aglets. Addison-Wesley, 1998. ISBN 0-201-32582-9.

15. J. Altmann, F. Gruber, L. Klug, W. Stockner, E. Weippl: Using Mobile Agents in Real World: A Survey and Evaluation of Agent Platforms. 5th International Conference on Autonomous Agents, 2nd Workshop on Infrastructure for Agents, MAS, and Scalable MAS at Autonomous Agents. Montreal, Canada, May 2001.
16. W. Wong: Stunnel: SSLing Internet Services Easily. SANS Institute, November 2001.
17. A. Pankratov: Trivial Denial of Service Attack against TCP-based VPN. Security Focus, April 2003.
18. N. Rinsema: Secure (and free) IP Tunneling using Zebedee. SANS Institute, June 2001.
19. M. Zalewski: Strange Attractors and TCP/IP Sequence Number Analysis. BlindView Corporation, 2001.
20. T. Ylonen, T. Kivinen, M. Saarinen, T. Rinne, S. Lehtinen: SSH Protocol Architecture, September 2002.
21. Netscape Technologies, Inc.: Introduction to SSL.
22. Author names: Performance Evaluation of Cryptographic Protocols for Mobile Multi-Agent Systems. To be published, 2004.
23. J. K. Ousterhout: Scripting: Higher-Level Programming for the 21st Century. IEEE Computer, March 1998.
24. J. K. Ousterhout, J. Y. Leavy, Brent B. Welch: The Safe-TCL Security Model. Sun Microsystems Technical Report SMLI TR-97-60.
24. Schneier: Applied Cryptography. Protocols, Algorithms and Source Code in C 2nd Ed. pp 336. John Wiley and Sons, Inc. 1996.