Enhancing the Survivability of Intrusion Detection Agents through Port Switching and Peer-to-peer Replication

Salvador Mandujano, PhD
Instituto Tecnológico y de Estudios Superiores de Monterrey, Monterrey, Mexico, smv@itesm.mx

Abstract
Security applications such as intrusion detection software often lack a security-conscious design that supports their vigilance goal. Similarly, software generation tools and libraries typically lack security constructs that support the development more robust systems. The latter is the case of agent-generation frameworks, which are rarely designed to guarantee agents a safe, continuous functioning hereby limiting their operational potential. Some intrusion detection architectures have contemplated the use of software agents and have inherited challenges from both, the design branch and the agent-support software branch. This paper sheds some light on the interleaving of survivability features into intrusion detection agents. It contemplates two aspects essential to agentry, namely communication confidentiality and agent availability. We discuss the details of a prototype that implements a linear-time port-switching communication protocol aimed at protecting exchanges between agents, and a peer-to-peer replication and location model to strengthen the availability of agents. We discuss the design and the specifics of the integration of these two techniques and present experimental results that show an increased security-level at a low performance cost.

Keywords
Agent survivability, intrusion detection agents, replication

1 Introduction

Several years ago, the evolution of software found a new paradigm. Object-centered systems gave birth to code and data structures called *agents* that collectively accomplished a common goal (Weiss, 1999). The basic capabilities of such entities made of this approach an enticing model for developing applications requiring remote communication, cooperative work, or distributed storage. The status quo of computer networks, including the Internet, points at the utilization of highly-distributed software that helps efficiently and meaningfully collect and analyze large amounts of information. *Multiagent Systems* (MAS) are a natural alternative to these problems.

The area of intrusion detection has also looked at this approach looking for potential solutions to some prevailing challenges. This type of applications reports the occurrence of anomalous activity by inspecting a number of evidence sources. The inspection and correlation tasks necessary for intrusion detection can be independently handled by autonomous entities and thus different architectures to integrate them have been proposed using agents (Allen et al., 2000; Balasubramaniyan et al., 1998; Zamboni and Spafford, 2000). The design of some of these architectures did not contemplate security, which impacted the robustness of the protection mechanisms itself.

A key element in software design that is not easily obtainable is reliability. Due to the number of system components – in software and hardware – that need to interact effectively in order to provide a service, systems have become more vulnerable to attack. Larger systems usually imply more lines of code, which increase the number of potential bugs. The complexity of systems also makes configuration and maintenance harder as they demand increased expertise from administrators who are responsible for setting up and supporting these systems. A whole platform can be compromised from a single configuration or coding error.

An agent-based service in operation could also be tricked into performing malicious activities. Agents could be shut down or their *knowledge bases* (KB) could be altered or deleted. Agent generators, libraries, and development environments speed up the development of this kind of software providing a good number of features that enrich it functionally. However, even when these systems usually include security routines to protect information exchanges with cryptographic algorithms, they do not prevent developers from creating flawed designs. An agent generated using these technologies is not automatically secure. In fact, the basic agent structures that serve as primitives to generate more specialized agents cover just the most elementary points of security such as encapsulation and identification (not authentication) through class interface definitions and agent name servers (Brugali and Sycara, 2000).

Security mechanisms are frequently proposed for these systems but just a few projects have actually implemented them. This paper discusses the integration of survivability mechanisms into intrusion detection agents. Security applications must be secure by themselves in order to aid other resources in the preservation of their confidentiality, availability, and integrity features. We propose the inclusion of these features as a default in all agents produced by automated tools as a way of improving, from the very beginning, their protection level. The rest of this paper is divided into five sections. Section 2 describes the motivation for studying agent survivability and points out the security challenges faced by MAS. Section 3 covers related projects in the field of agent security. The proposed solution techniques are explained in Section 4, and Section 5 outlines the implementation details of the prototype. Finally, Section 6 describes the experiments and results. Conclusions appear at the end, in Section 7.

2 Background

Security has been a challenge to agent-based software for a long time. These are some of the most relevant aspects that need to be taken into consideration when dealing with the safe operation of this type of applications.

2.1 Security challenges in MAS

Given the lack of security in agent frameworks, once the code of an agent has been produced, it will have to be secured in a separate effort in order to guarantee communication privacy and continuous availability. There are two main threats that have been identified and studied by the MAS community: *malicious agents* and *malicious hosts* (both of which are exacerbated by mobility that adds a number of important challenges due to the fact that software no longer execute at a single, fixed location (Schneider, 1997.)) An agent can communicate with its peers through message passing, blackboard systems, mailboxes, etc. (Lynch, 1996). At any given time, a service request or reply from a peer agent could be incorrect. The agent could have been subverted or sabotaged by an attacker and, therefore, its behavior can no longer be deemed trustable. If a legitimate agent submits a request for data to an untrusted agent, the answer could contain false information leading to undesirable actions. The same would occur if an agent name server is compromised and its tables are modified. References returned to clients could be pointing at incorrect resources.

The problem of *malicious agents* has been widely studied and there exist solution proposals to many of its challenges (Pleisch and Schiper, 2000). Authentication methods, cryptography, and privilege systems have been created to guarantee that the interaction between agents is safe and evolves as planned. Of higher complexity is the problem of *malicious hosts* (Guan, 2000). How can someone prevent a host from reading or modifying the KB or even the code of an agent it is executing? A mobile agent that travels from host *M* into host *N* carries data from *M* and other previously visited hosts. In principle, the system administrator at *N* has access to the contents of the agent. Most operating systems support this superuser access level and so it is hard to limit what administrators are able to do and see. Techniques for the execution of encrypted code (Sander and Tschudin, 1998) and watermarking have been developed to deal with this problem. Malicious hosts and agents can cause severe damage to the operation of an agent-based system. As long as these problems remain unsolved, the agent paradigm will be unable to fully develop and succeed in commercial and other practical applications (Borselius, 2002).

Agent-based intrusion detection architectures pose similar challenges. Arranged in a hierarchical or networked fashion, autonomous agents are in charge of capturing, filtering, correlating, and, in some cases, reacting to security-relevant information (Janakiraman et al., 2003; Zamboni and Spafford, 2000). Single point-of failure, limited resilience, and the use of no cryptographic method are typical flaws in the design of these architectures. Of particular interest to us is the *robustness* of agents. Our intention is to generate agents with a generic design and inner structure that allows them to provide, from launch time, continuous operation by tolerating events such as system failures and intentionally induced faults, a.k.a. *attacks*. *Fault-tolerance* deals with failures originated from the day-to-day operation of a system. These failures are typically unintentional,and probabilistic methods exist to detect and correct them (Pleisch and Schiper, 2000). At a higher level of complexity appear failures caused intentionally by humans. As opposed to system failures, these are carefully crafted and concealed so that they are difficult to detect, difficult to prevent, and difficult to recover from.

Probabilistic fault-tolerance methods are not enough to characterize the way attackers behave, though. *Survivability* studies this kind of intentionally induced faults. For the purpose of this paper we will define survivability as the *capability a system has to provide a subset of basic services even in the presence of attack or failure*. In the case of security software agents, we consider the development of survivable systems able to stand attacks from malicious entities. In particular, we focus on the confidentiality and availability of agents without making any assumptions concerning the probabilistic independence of this type of faults.

2.2 Security risks to individual agents

Programming bugs and configuration errors increase the possibilities of having a system compromised. Agents may be attacked separately with the intention of, for instance, disabling them, sniffing on their communications, keeping them at a particular location, or modifying the contents of their KB's. All this could severely disrupt the activity of an agent by making it untrustable to others. The features an agent offers to others can be aimed by an attacker. Agents are *autonomous* by definition and, as such, should be able to execute without the intervention of a central entity (Weiss, 1999). An agent is designed to have enough knowledge as to decide what to do at any time. An agent should also display *situatedness* through which it perceives the environment it inhabits and is able to change it through actions. An agent is *social* by being capable of communicating and cooperating with other agents in order to solve common and individual goals. *Rational* or *goal-based behavior* is also expected: an agent will never act in a manner that impedes the achievement of its goal. *Mobility* is highly desirable as it gives agents the possibility of suspending execution, moving to a different host, and resuming operations over there.

As components of a MAS, or working independently, attackers who try to circumvent the security of a system can compromise intrusion detection agents as well. The following types of attack could be implemented to alter the integrity and functioning of an agent platform (Sander and Tschudin, 1998):

1. **Privacy comprises**. An agent in execution can store private information which is only shared through predefined interfaces. However, the machine that hosts the agent has total access to its resources. Some pieces of information could be illegally read by the host with the intention of benefiting other applications or competing agents. Unencrypted communications among agents can be easily observed. Confidential data such as trajectory plans, digital money, and the contents of a KB could also be copied by an intermediate server that receives the agent if the information is not protected. Whether it is a peer-to-peer or a centralized message-passing scheme, sniffing programs can always be installed to observe network traffic at strategic locations.

2. **Integrity compromise**. If any external party gets to modify the data or the code of an agent, this can no longer be considered stable nor being consistent with its design specification. Similarly, if an agent interpreter or other agents detect that an agent has experienced any sort of modification, they can no longer trust the information received from it as the agent itself may not be working to fulfill its objectives. The cooperation potential of the agent will also be limited, as agents will look for alternate agents instead of interacting with an agent whose integrity has been compromised (Yee, 1999). This will cause suboptimal collaboration as a result of a reduced number of trusted agents.

An attacker can also remove the KB of agents and they will "forget" everything they have captured and will have to be start up from scratch in order to collect data that helps them make decisions again and proceed toward their goal. A mobile agent could be tampered with so as to trick it into performing abnormal actions. Once received by a host, it could act as a Trojan horse able to plant backdoors or stealing information to send it later to a predetermined location (Borselius, 2002).

3. **Availability compromise**. The availability of an agent platform can be compromised in several ways such as service flooding, also known as *Denial of Service* (DoS), the actual deletion of agents, or via restricted communications imposed by the host entity (either it be an executing virtual machine or a server). The processing and storage capabilities of agents can also be reduced and agents can mistakenly perceive that a certain resource has been exhausted. In any of those cases, the agent will be unable to provide the services expected from it. By having its availability affected, an agent will miss opportunities to cooperate with others, to supply information, and, at the very least, to timely serve requests from its peers.

3 Related projects

Two important implementation concerns of fault-tolerant agents have been discussed by Schneider (Schneider, 1997): 1) agent authentication, and 2) policy definition and enforcement in the context of voting systems. As a way to achieve fault tolerance, agent replication has been proposed. Intermediate agent states between a source and a destination are replicated in order to have redundant communication paths between the hosts. In order to achieve security, messages are transmitted using threshold secret sharing so that colluding nodes are unable to intercept communications.

Replication for fault-tolerant agents is studied in (Fedoruk and Deters, 2002). Redundancy by replication leads to additional complexity and overhead can be minimized through proxy-based schemes such as transparent replication. This technique allows agents to communicate with replicated clones through single points of contact. The DARX framework (Martin et al., 2001) proposes replication not only for agents but also for other objects. Critical-level tasks within a multiagent system are able to switch between active and passive replication at execution time. This increases the processing needs from the system but has the advantage of detecting faults more accurately.

In (Dasgupta et al., 2000), they analyze the malicious-agent and the malicious-host problem in the light of a multiagent e-commerce system called MAgNET. Some sellers and buyers may attempt, for instance, to read private information or to alter the behavior of their peer agents in order to gain some advantage during negotiations. The system is based on ASDK aglets for providing its core functionality. The proposed solution involves methods to authenticate agents and servers before any interaction is started. MAgNET implements privilege controls to define the operations permitted to each aglet. These methods involve electronic signatures and certificates through which all parties are authenticated. For the malicious-host problem, a server is unable to execute the code carried by an aglet unless it has suitable privileges to do it. State appraisal (Farmer et al., 1996) studied this latter problem and implements authentication and authorization granting by inspecting the state of an agent that requests access to a server. Mobile agents may travel across networks where their integrity gets exposed. Users and hosts are able to prevent attacks from rogue agents by carefully granting the minimum set of privileges an agent requires reaching its goal and a group of state-appraisal functions provides an interface for analyzing the agent's state and requesting permits that protect interpreters from executing code that may have been modified by another host.

The Sanctuary project (Yee, 1999) was developed with the intention to create a secure infrastructure for mobile agents. This architecture presents several advances in agent execution, privacy and integrity through cryptographic support. Being distributed function evaluation unfeasible for practical purposes, and costly for protecting an agent from the executing host, this project implements *secure co-processors*. These are trusted environments where agent code can be safely run with the guarantee that the results have been correctly computed. Software approaches to ciphered computation of mobile code can be found in (Sander and Tschudin, 1998). With these methods an agent is able conceal executable code from the host and also to sign a document without disclosing its private key. It is true that: "There is no intrinsic reason why programs have to be executed in clear-text form… (This assumption) is wrong because it tacitly assumes that a mobile agent consists of clear-text data and clear-text programs" (Sander and Tschudin, 1998). Even when this approach has been only tested with simple functions, it reduces the set of potential attacks to denial of service and random modifications to agent programs or its output.

4 Solution

We focus on two of the security problems already discussed. The proposed model protects the communications and KB's of intrusion detection agents by guaranteeing they will be able to provide a minimum level of functionality even in the presence of attack or failure. Unlike some of the projects mentioned before, agents are not assumed to fail or be attacked independently from each other. This section outlines the security problems these models address and describes the ideas behind them.

4.1 Peer-to-peer agent replication

In the event of attack, a host may reach an unstable state at which it is no longer safe for agents. Either through self-checking routines that evaluate the integrity of the agent, or through external intrusion detection mechanism that inform about the security level of the host, the components of a MAS can be warned about the likelihood or actual existence of an attack. An adversary may compromise all or some system resources depending on the authorization level he gets. Once inside, he will be able to access the code and/or KB of agents for reading or writing. Any of the types of system compromise mentioned in Section 2.2 may seriously affect the integrity and confidentiality of agents. Considering the fact that an agent's integrity is priority over availability since no trusted service can be really provided by a flawed entity, an agent may sacrifice some of its availability with the intention of preserving integrity and confidentiality.

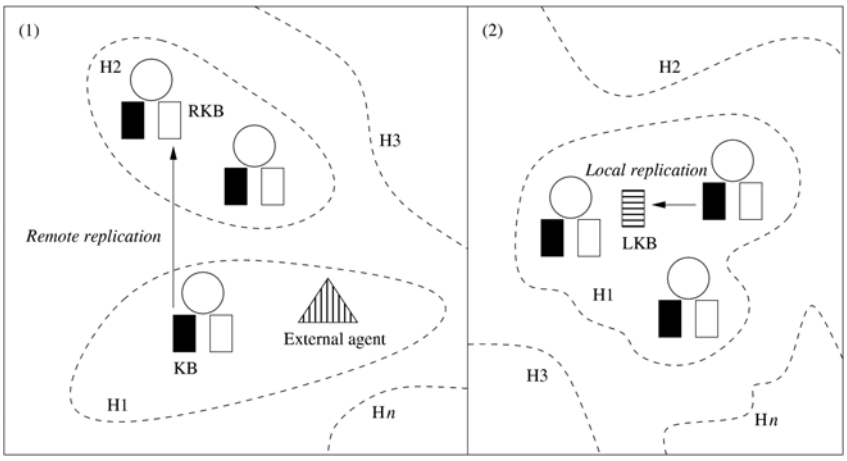

Figure 1. (1) Remote replication of a KB. (2) Local replication of a KB. ***Hi*** **denotes a host;** ***LKB*** **a local copy of a KB; and** ***RKB*** **a remote copy of a KB.**

This solution is possible when working with autonomous entities. It would not apply to systems that cannot move or cannot suspend their operation. The agent paradigm makes it possible allowing an agent to protect its KB by suspending execution and resuming at a safe, remote location and informing its peers about the change (Sander and Tschudin, 1998). Unlike many MAS, we use a mechanism for agent location that is not centralized. There is no agent name-server in charge of storing references to all available agents. Instead, this is done by storing on each participant the location of all agents (bear in mind that the number of agents that comprise an intrusion detection platform is typically small). At the cost of space and performance, it is possible to eliminate the single-point-of-failure that a central name server would represent and which constitutes a typical risk of hierarchical models. The information about what agent is located where, is not only stored in a distributed fashion but also it is redundantly maintained on every agent so that a compromised peer can recover as soon as possible from an attack by contacting a peer.

If the file system of a host *H* is not compromised, an agent that perceives an unsecure state on the host may replicate its state to disk using encryption, then suspend execution, and finally resume at the same location when the security level of *H* come back to normal (Figure 1). When the agent is unsure about the security state of the host, it will be necessary to replicate remotely. In the case of a deterministic remote replication model, an attacker could know beforehand which are the hosts that store the KB's of the replicated agents. (Pseudo)randomized methods prevent an attacker from observing a transfer between predetermined locations. This technique is complemented by a dynamic port allocation model (see Section 4.2) that fragments and ciphers the replication stream deterring an attacker from capturing an exchange.

Using random information from the system and its users, an agent is selected from the pool of authenticated agents – authentication, location, and replication are performed in a peer-to-peer fashion (Figure 1). This agent will store the KB and execution state of the agent that perceives an unsafe state at its hosts. All agents are capable of replication and keep record of the agent to which they send their state in order to recover it in the future. For a number *n* of servers hosting a number *m* of agents, this replication model escalates well even for large values. The time needed to select a remote replication agent grows *O(m)* as agents store reference to all available agents.

Servers that have been compromised are not considered for remote replication and, as long as there is one available host running, its agents will be located. If an intrusion detection agent used *proxy-controlled replication* (i.e., a model where a coordination agent is in charge of managing replication requests from

client agents; Figure 2) it could backup its state to a peer agent but the strength of the mechanisms is questionable. At least it suffers from single-point-of-failure that makes it less desirable for security purposes. A peer-to-peer mechanism provides increased reliability leveraging the natural robustness of MAS when organized non-hierarchically.

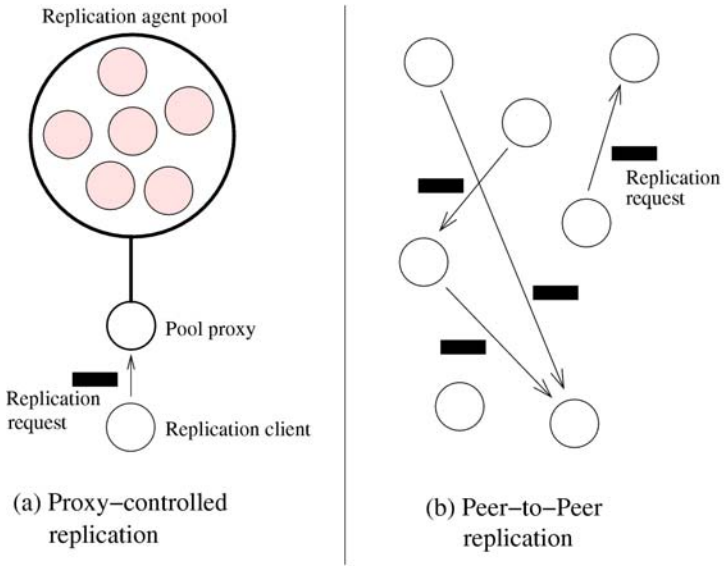

Figure 2. Proxy-controlled replication and peer-to-peer replication

4.2 Channel switching

Network daemons usually work at a predetermined port listening for service requests (although requests from clients are usually generated within a broad range of port numbers). *Port-scan attacks* probe ports trying to identify what type of services they provide. Once this is done, they look for vulnerabilities present on those services and, if they find one, they are in a position to exploit them (Klevinsky, 2002). DoS attacks typically start as port-scans which, after identifying flawed services, attempt to flood the system with an overwhelming number of requests. This same sort of attack can be launched to disrupt the operation of a MAS as well. We propose a model to minimize the likelihood of an agent being compromised while it communicates with others.

In order deter this sort of attacks, we use a technique on both, client and servers, commonly used with radio equipment: *channel switching* (services like FTP use switching but only at one side of the connection; this switching order can be known a priori; Northcutt and Novak, 2002). Only at start-up will agents listen on a well-known port number s_0. The requests for service on that port do not allocate significant system resources. This preliminary port is used exclusively to agree on an actual communication port s_1 for the exchange (s_i 6= s_i + 1). The receiver proposes a valid port number that is sent to the other party over a secure channel using public-key encryption. They immediately switch to that port for information exchange.

Since the overhead of negotiating random port numbers is considerable, our model handles sets of *k* port numbers (see Figure 3) that are used for one communication round only. The sender creates and sends the set $Ps_1 = [s_1, s_2, \ldots, s_k]$ to be used in the first exchange. It also sends message m_1 to the default port r_0 of the receiver. The receiver defines its initial set of random port numbers $Pr_1 = [r_1, _2, \ldots, r_k]$ as well as which it sends to port s_0. Port switching is maintained during the session until new port numbers need to be agreed upon. In order to circumvent this protection mechanism, a port-scan will have to be run over and over again and, by the time it succeeds, the communication will have most likely moved to a different

port already. The value of the information transmitted through a session must be such as to make unlikely for an attacker to integrate the packets observed at different ports and to arrange them in a way that makes sense (remember also that all exchanges are encrypted). The frequency of switching along with encryption makes it a more secure mechanism.

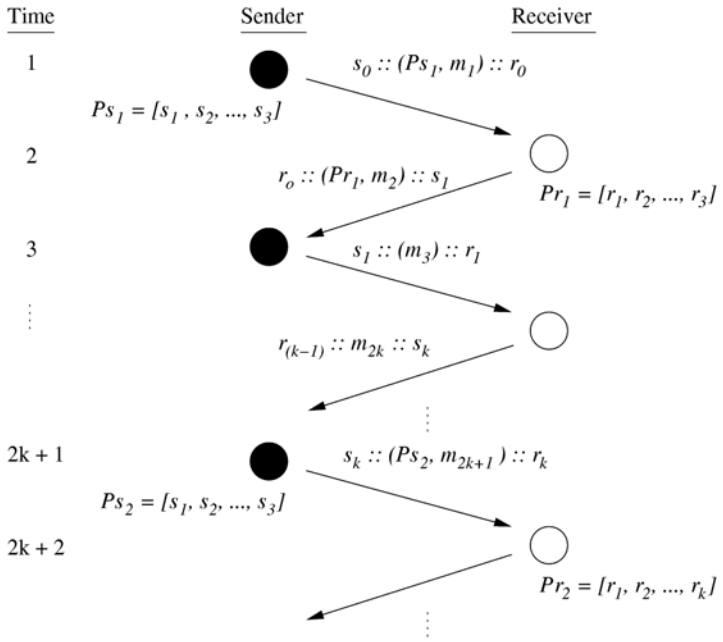

Figure 3. Channel switching for communication between two agents

5 Implementation details and experimental results

All agents are equipped with client and server capabilities. Every agent contains a key-pair to cipher and decipher messages. At start up, the administrator is asked to enter a pass-phrase that will protect the private key on the file system. The public key is broadcast to all agents so that secure communication can take place. It is assumed that, when the host is at a stable state, primary memory is safe and keys can be uploaded into it in order to perform encryption. Once this is done, the data structures that contain the keys are overwritten on disk (not just de-allocated in memory) with blank padding. The name and location of all available agents are stored into every agent and are updated with messages concerning new locations or service suspension. Remote replication agents are selected randomly and they can store the KB of more than one agent. In the implementation, agents can only serve one replication request at a time.

TCP sockets are used to develop the port allocation scheme. A data structure contains the series of random port numbers that will be used at each exchange. The number of ports *k* stored by this structure may vary (see results in Figure 5). A small *k* indicates port numbers will have to be negotiated very often during a session. A large *k* reduces the time spent in negotiation of port numbers, but represents a bigger risk since, in the event an intruder captures port numbers, more exchanges could be compromised.

The implementation of the experimental prototype was done on a RedHat Linux-9.0 box using the *gcc* compiler and coded entirely in C. The testing facility is a set of Linux machines running kernels 2.1 and above on Red Hat 7.3, RedHat 8.0, RedHat 9.0, Mandrake 8.0, and Debian Linux installations. Public-key

encryption was implemented using the Open SSL library in PEM format for RSA. Using dynamic compilation, agents are around 23 Kb in size. If static compilation is used, agents become larger (540–600 Kb) but have the advantage of not using external libraries that could have been tampered with. To obtain randomness, users are asked to enter some input at the time an agent is to be created (this randomness is also used to generate RSA pairs) and is combined with some reads of the */etc/urandom* device that uses the state of RAM, registers, and other system components to produce entropy.

6 Experiments and results

The objective of the evaluation phase was to analyze the performance of the model after having integrated both security mechanisms, as well as to detect improvement areas. Given that these schemes propose a trade-off between performance and security, it is important to quantify the overhead of the implementation. In particular, the test scenario tries to compare 1) replication times using plain-text exchanges, 2) replication times using encryption, 3) transfer times using one single communication port, and 4) transfer times using randomized port switching. Two main experiments were prepared: one to observe the behavior of the replication model, and one to evaluate how the dynamic port allocation scheme worked.

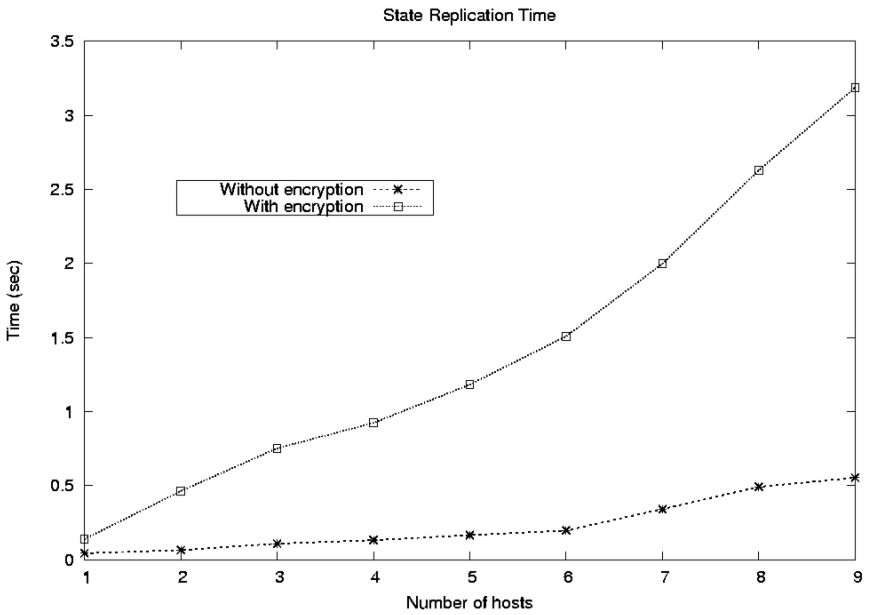

Figure 4. Replication time with and without RSA encryption

6.1 Replication results

In order to evaluate the behavior of the replication model, we used nine hosts. First, we performed state replication in plain text with a variable number of hosts $n \in \{1, 2, ..., 9\}$. Figure 4 shows the results after 10 executions with each configuration. We then tested the model enabling RSA encryption in order to evaluate the impact of cryptography on the replication model (replication with encryption was performed with the same number of hosts n). For each n, the experiment was repeated 10 times as well. The figure shows average times from these executions corresponding to different values of n. As the number of hosts increases, so does replication time. For plain-text replication, times are short as no time is spent encrypting the state of the agent using the public key of the recipient (public-key encryption is typically expensive). It is interesting to observe that, even when the number of hosts increased, replication times with encryption did not present any dramatic growth. The cost of implementing replication with or

without encryption is low considering the experiment performs remote replication on remote hosts for all *n > 1* shown in the plot.

6.2 Randomized port switching results

Several experiments were conducted in order to determine the performance cost of the dynamic port allocation scheme. The variable with the most weight is the number of ports included in a packet. This variable k defines a trade-off between confidentiality level and performance. Given that all exchanges are peer-to-peer, the experiments consisted of two agent instances exchanging its KB's. Figure 5 shows the results. KB's were transferred in pieces of 256 Kb in size and transmission times were taken for one-way trips only. The experiment was performed using the following values of *k: 1, 25,* and *50*. Each experiment was repeated 20 times.

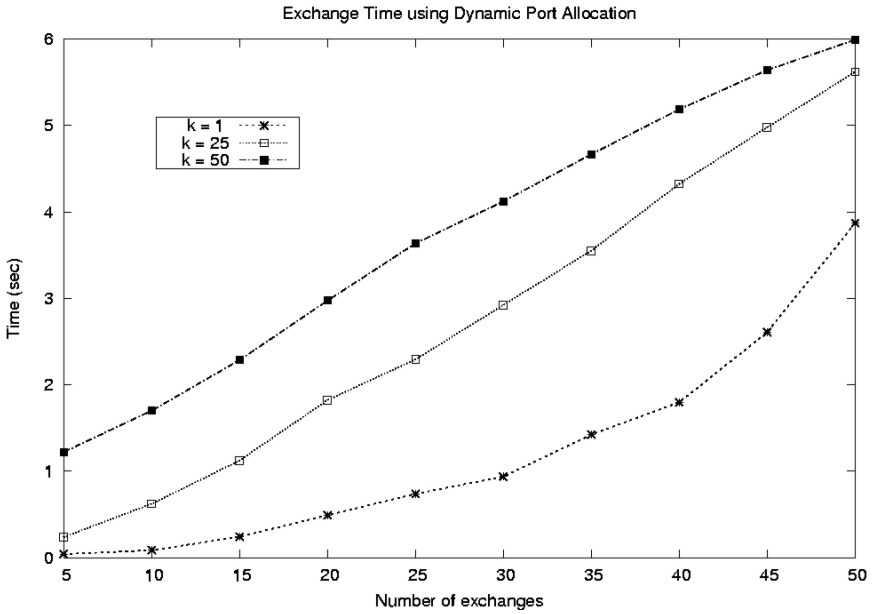

Figure 5. Exchange times using different packet sizes (k**) for channel switching**

From the above plot we conclude that the performance overhead of the port-switching model is considerable. The cost of opening and closing TCP connections at different locations increases transfer times as resources and data structures need to be allocated before exchanging any packet. The benefit, however, is that packet sniffing is significantly more difficult and that any data chunk captured from this communication scheme will have to be put together with others in order for it to be meaningful (the intruder will have to capture and interleave data addressed to different ports; moreover, all exchanges are encrypted).

No sequence number for packets is provided in the protocol as the switching is entirely managed internally by the agents. Furthermore, encryption deters confidentiality attacks and, since communications do not develop at a fixed standard port all the time, an attacker will have to first find the port sequence to which a socket is bound and those port numbers change permanently using random input.

7 Conclusions

We have presented two techniques that contribute to the survivability of security agents making no assumption regarding the probabilistic independence of failure or attack. Through local and remote state

replication, agents are able to protect their execution state and to escape subversion attempts. Using a randomized channel switching reinforced with encryption, agents can deter flooding attempts and sniffing. The results show that the combination of these techniques is a viable way to prevent common confidentiality and integrity attacks launched toward agents, and that this solution scales well to a large number of security agents.

References

Borselius, N. "Mobile agent security". *Electronics & Communication Engineering Journal*, 14(5):211–218, October 2002.

Brugali, D. and Sycara,K. "Towards agent oriented application frameworks". *ACM Computing Surveys*, 32(1):21–26, 2000.

Dasgupta, P., Moser, L. E., and Melliar-Smith, P.M. "The security architecture for Magnet: a mobile agent e-commerce system". In *3rd International Conference on Telecommunications and E-commerce*, pages 289–298, Dallas, TX, November 2000.

Farmer, W.M, Guttman, J.D., and Swarup, V. "Security for mobile agents: Authentication and state appraisal". In *4th European Symposium on Research in Computer Security*, pages 118–130, Rome, Italy, 1996.

Fedoruk, A. and Deters, R. "Improving fault-tolerance by replicating agents". In *1st International Joint Conference on Autonomous Agents and Multi-Agent Systems, AAMAS*, pages 737–744, Bologna, Italy, July 15–19 2002.

Guan, X., Yang, Y., and You J. "POM – A mobile agent security model against malicious hosts". In *4th International Conference on High-Performance Computing in the Asia-Pacific Region*, volume 2, pages 1165–1166, May 2000. Beijing, China.

Janakiraman, R., Waldvogel, M., and Zhang, Q. "Indra: A peer-to-peer approach to network intrusion detection and prevention". In *Proceedings of IEEE WETICE 2003*, June 2003.

Klevinsky, T.J., Laliberte, S., and Gupta, A. *Hack I.T. – Security through Penetration Testing*. Addison-Wesley Longman, 1st edition, February 2002.

Marin, O., Sens, P., Briot, J.P., and Guessoum, Z. "Towards adaptive fault tolerance for distributed multiagent systems". In *ERSADS 2001*, pages 195–201, Bertinoro, Italy, 2001.

McHugh, J. "Intrusion and intrusion detection". *CERT Coordination Center, Carnegie Melon University, Springer–Verlag*, July 2001.

Northcutt, S. and Novak, J. *Network Intrusion Detection*. New Riders, Indianapolis, IN, 3rd edition, September 2002.

Pleisch, S. and Schiper A. "Modeling fault-tolerant mobile agent execution as a sequence of agreement problems". In *9th IEEE Symposium on Reliable Distributed Systems (SRDS2000)*, pages 11–20, Nurnberg, Germany, 2000. IEEE Computer Society.

Sander, T. and Tschudin, C.F. "Protecting mobile agents against malicious hosts". *Lecture Notes in Computer Science, LNCS*, 1419:44–49, February 1998.

Schneider, F.B. “Towards fault-tolerant and secure agentry”. In M. Mavronicolas and P. Tsigas, editors, *11th International Workshop on Distributed Algorithms, WDAG*, Berlin, Germany, 1997. Springer Verlag.

Weiss, G. *Multiagent Systems: A Modern Introduction to Distributed Artificial Intelligence*. MIT Press, Cambridge, MA, 1st edition, 1999.

Yee, B.S. “A sanctuary for mobile agents”. In *Secure Internet Programming: Security Issues for Distributed and Mobile Objects, Springer Verlag, Lecture Notes in Computer Science, LNCS*, pages 261–273, Berlin, Germany, 1999.

Zamboni, D.M. and Spafford, E.H. “Intrusion detection using autonomous agents”. *Computer Networks – Elsevier*, 34(04):547–570, October 2000.

Biographical Information

Salvador Mandujano is a recent PhD graduate from ITESM. He did research with the Center for Intelligent Systems and the Information Security group at the Monterrey campus and is currently working for Intel in Hillsboro, USA.

Authorization and Disclaimer