Securing Agents against Malicious Host in an Intrusion Detection System

Rafael Páez[1], Joan Tomàs-Buliart[1], Jordi Forné[1], and Miguel Soriano[1,2]

[1] Technical University of Catalonia, Telematics Engineering, Jordi Girona 1-3, Barcelona, Spain
[2] Centre Tecnològic de Telecomunicacions de Catalunya
Parc Mediterrani de la Tecnologia (PMT), Av. Canal Olímpic S/N, 08860 - Castelldefels, Barcelona, Spain
{rpaez,jtomas,jforne,soriano}@entel.upc.edu

Abstract. In an agent's environment, the most difficult problem to solve is the attack from a platform against the agents. The use of software watermarking techniques is a possible solution to guarantee that the agents are properly executed. In this paper we propose these techniques in an Intrusion Detection System (IDS) based on agents. To achieve this goal, we propose to embed a matrix of marks in each transceiver of the IDS. Moreover, we include obfuscation techniques to difficult a possible code analysis by an unauthorized entity.

Keywords: Multiagent systems, Mobile agents, Intrusion Detection Systems, IDS, *Watermarking*, *Fingerprinting.*

1 Introduction

The security of systems based on software has become in an important subject because most of them must control critical infrastructures like centres for disasters prevention, intelligent buildings, planes' functions automation, etc. So, many human lives and huge amount of money could depend on the confidentiality, integrity and availability of these critical systems. There are several tools to achieve these security requirements such as firewalls, honeynets, honeypots, intrusion detection systems, etc. Due to the high dependability of the systems in this type of tools, they become objectives susceptible of being attacked and therefore in critical systems that also need to be protected.

Particularly, the Intrusion Detection Systems (IDS) have as a goal to detect suspicious activities and prevent from possible intrusions in a network or system at the moment when happen. Therefore, it is important to keep in mind the integrity of the information, authentication and access control. The different entities that compose the IDS need to be communicated among them and cooperate to achieve the system's goal. So, the use of agents inside IDS has been proposed because they can perform simple tasks, that joining them resolve complex works [1].

On the other hand, one of the reasons that has jeopardised the generalized use of the mobile agents is precisely their security. In this paper we focus our attention in the

J. Lopez and B. Hämmerli (Eds.): CRITIS 2007, LNCS 5141, pp. 94–105, 2008.

agent's protection as part of an IDS. In particular, we extend the work previously presented in [14] in order to make the system more resistant against replay attacks.

The rest of the paper is structured as follows. Firstly, in section 2 we present the required background and the previous work by the authors: the Cooperative Itinerant Agents (CIA). Then, section 3 identifies the risk of replay attacks to the CIA scheme, and outlines the proposed solution, that combines the use of a matrix of marks (section 4) and code obfuscation techniques (section 5). Next, in section 6 we describe the algorithm proposed to embed the mark in the agent an, finally, we conclude in section 7.

2 Background and Previous Work

In an IDS based on autonomous agents it is necessary to combine different tools to guarantee the required security level. We propose to use software fingerprinting and software obfuscation techniques. Likewise, we have analyzed possible threats to provide a solution.

2.1 Intrusion Detection Systems

An Intrusion Detection System (IDS) tries to detect and alert about suspicious activities and possible intrusions in a system or particular network. An intrusion is an unauthorized or non wished activity that attacks confidentiality, integrity and/or availability of the information or computer resources. In order to reach its goal an IDS monitors the traffic in the network or gets information from another source such as log files. The IDS analyzes this information and sends an alarm to the system administrator. The system administrator decides to avoid, correct or prevent the intrusion.

The basic architecture of an IDS is conformed by the data collection module, detection module and response module [2]. Inside the data collection module is located the event generator sub module which can be the operating system, the network or a particular application. The events generator sends the packets to the events collection sub module which relates the data and sends the information to the detection sub module. The analyzers or sensors which filter the information and discard irrelevant data are located inside the detection sub module. Finally, the data are sent to the response sub module. The response module decides if send an alarm to the system administrator basing on predefined policies.

2.2 Software Watermarking and Fingerprinting

Watermarking techniques have been basically used in the protection of digital contents. With these techniques, some information (usually called mark), is embedded into a digital content like video, audio, software, etc. The main objective is to keep this information imperceptible in all copies of the content that we protect in such a way that we can later demand the authorship rights over these copies. In software watermarking, the mark must not interfere with the software functionalities. The mark can be: static, when it is introduced in the source code, or dynamic, when it is stored in the program execution states.

In the same way software fingerprinting techniques appeared. The aim of this kind of watermarking is to identify the author of copies, as in watermarking scenarios, and also identify the original buyer of each copy. In other words, a different mark is embedded in every copy before distribution. The main attack to fingerprinting schemes is the collusion attack, meaning that, some malicious users compare their copies and they can try to construct a new copy with a corrupted mark which can not blame any of them.

In the scenario presented in this paper, the fingerprinting techniques are used to include a different matrix of marks in each copy. As a consequence of using fingerprinting and watermarking techniques, this inclusion will be imperceptible against inspection attacks and it provides a consistent tamperproof protection.

2.3 IDS Based on Autonomous Agents

According to [4], [5] and [6], the mobile agents are suitable to IDS since they offer scalability, resilience to failures, code independency, network traffic reduction, facility to perform previous proves to the agents in a independent manner before deploying them to the system, among others.

The architecture for IDS based on autonomous agents is built by the following components:

- *Monitors*: They are data processing entities and the main controllers of the system. Monitors have an overall vision of the state of the network and can detect suspicious activities. They can also raise alarms and are hierarchically connected to other monitors. In addition, monitors offer an interface that allows the user to interact with the system.
- *Transceivers*: They control all the agents of the host and can process data sent to them from the host. A transceiver communicates with the monitor on which it depends, within the hierarchical structure. Moreover, it can start, stop or eliminate the agents that are dependent on it.
- *Agents*: They can be distributed at points within the network and monitor that particular traffic. An agent is a separate process that stores states, carries out simple or complex actions and exchanges data with other entities. Each agent generates a report and sends it to the transceiver but it cannot generate an alarm.
- *Filters*: They make a selection of data and send the registers to the agents that correspond to the given selection criteria. There is only one filter for each data origin and the agents can be subscribed to each one of the different filters.

AAFID system [3] includes a user interface like component of its architecture. User interfaces use APIs exported by the monitor, to ask for information and to provide instructions.

2.4 Risks in an IDS Based on Agents

The internal security of an IDS based on autonomous agents is an important factor to keep in mind, therefore it is necessary to protect the access to the platform and to the agents to ensure the privacy and the integrity of the data exchanged among them.

Although the mobile agents offer many advantages, because of their nature they also incur risks. Possible threats are the following: agent against the platform, platform against the agents, agents against other agents and other entities against the agent's system. There are different solutions to reduce these risks [7], [9]. In this work we analyze the threats of the platform against the agents to offer a possible solution.

2.4.1 Platform against Agents

Particularly, the threats from platform against agents are the more difficult to prevent. This is because the platform has access to the data, code and results of the agents located on it. In this way, if a host is malicious, it can perform an active or passive attack.

In the case of a passive attack, the host obtains secret information as electronic money, private keys, certificates or secrets that the agent utilizes for his own requests of security. On the other hand, to perform an active attack, the host would be able to corrupt or to modify the code or the state of the agents. A malicious host can also carry out a combination of passive and active attacks, for example, by analyzing the operation of the agent and applying reverse engineering to introduce subtle changes, so the agent shows malicious behaviour and reports false results. Our proposal is focused on verifying the integrity of the agents, transceivers or monitors in runtime.

2.5 Cooperative Itinerant Agents (CIA)

The CIA security scheme [14], which consists in using itinerant cooperative agents, was proposed to verify the integrity of the monitors and transceivers in an IDS based on autonomous agents. The system works in the following way: Each monitor generates a transceivers agent for each host in the network segment that controls. Similarly, the monitor embeds a fingerprint mark onto each transceiver and keeps a copy. The transceivers generate information collection agents that are located in the lower level of the IDS infrastructure. The monitor generates a cooperative itinerant agent with a previously defined itinerary. Thus, the agent travels through the network segment that is controlled by the monitor that generated this agent.

In Fig. 1 the process of CIA agents is illustrated. The network segment controlled by a CIA corresponds to the underlying level of an issuer monitor within the tree infrastructure. Every time that the CIA arrives at a host, it requests its fingerprint mark via the corresponding transceiver; subsequently the agent forwards the response to the monitor. The monitor verifies that the mark requested is correct by comparing it against each mark belonging to its set of marks. If the mark does not match any of the marks in the set, then it is assumed that the agent was manipulated, so the monitor will be able to act. This consists in eliminating the suspicious transceiver or isolating the malicious host.

The monitors' verification is performed in a similar way to that of the transceivers; when the CIA moves from one host to another, it reports its new destination to the monitor and continues its itinerary, repeating the process in each host. Each CIA can be configured to monitor entities with given profiles, for example, to verify transceivers located in a rank of directions or to verify monitors only. The agent can be programmed to cover either a previously defined route or a random one.

A special case of verification occurs when monitors are located at a high level within the hierarchy but there is no upper entity to verify them. In this situation, a cross verification of the monitors in the same level must be performed. This means that there must be at least two monitors in the root level. Each monitor within the upper level then generates a cooperative agent to verify the integrity of the neighbouring monitors.

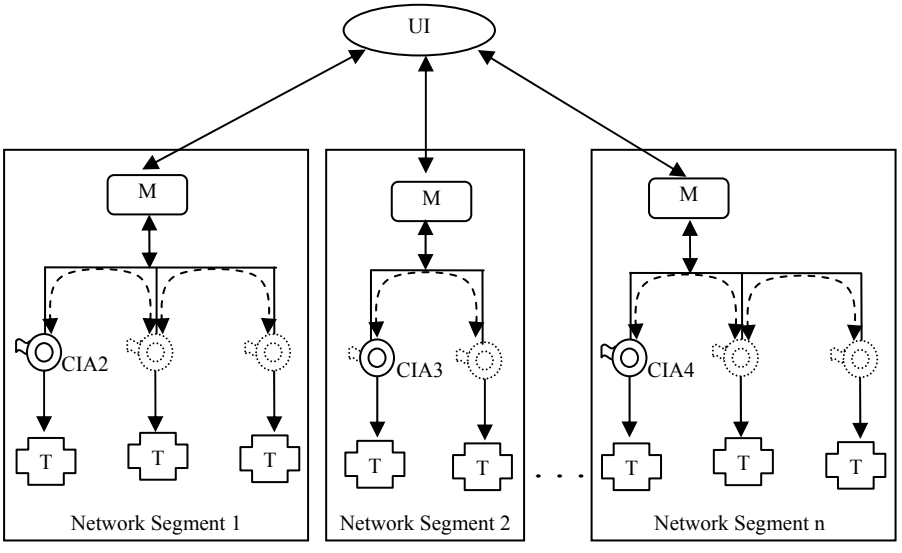

Fig. 1. Transceivers verification by cooperative agents

3 Protecting Agents against Replay Attacks

Because the malicious host has access to the code and state of the agent, the agent is exposed to such attacks, which is a disadvantage of the CIA security scheme. If the host performs an active attack, the monitor will be able to detect it because it will either not receive a notification on time or it will receive an incorrect notification. However, if the host performs a passive attack, it can, for example, detect and copy the response sent by the transceiver or CIA. Thus, when the agent is verified again, a malicious host can replace the response to deceive the monitor (i.e., perform a replay attack).

In order to avoid this kind of attack we propose using a particular mark. This mark is a matrix that identifies each monitor and transceiver in the IDS. The matrix is in turn split into various submarks; when a CIA arrives at a host it requests a set of submarks through a particular function. The corresponding entity (monitor or transceiver) responds with the result of another function. In this way, if a malicious host intercepts the communication, the information obtained cannot later be used to deceive the verifier. We also propose using software obfuscation techniques to prevent or hinder the process of reverse engineering or code analysis by a malicious host (see section 5).

4 Using a Matrix of Marks

Software can be identified by a mark; thus it is possible to prove not only its integrity but also to reclaim copyright. This mark should be embedded in a place known by the verifier and must seem like part of the results, that is, it should be imperceptible and resistant to transformation attacks. However, the mark should not influence in the operation of the marked code. The marks are static when they are stored in the application code and dynamic when they are constructed at runtime and stored in the dynamic state of the program [11], [13].

To identify transceivers and monitors from an IDS based on agents, we propose a matrix be used as a mark. The matrix has a fixed, previously determined dimension: m x n. Each cell contains a prime value; prime values are considered submarks. To request the mark we use a cooperative itinerant agent (CIA). The CIA uses the following function to request a set of submarks from the IDS entity:

$$f_1(x) = \{(x * p + r_1), (x * q + c_1), (x * r + r_2), (x * s + c_2)\} \quad (1)$$

where x is an integer greater than or equal to zero and y represents the module used by the agent; it masks the values and is known by the transceiver. The values p, q, r and s are random prime numbers that change every time the agent uses the function. The values (r_1, c_1), (r_2, c_2) correspond to the number of row and column numbers of two prime numbers of the matrix (row1, column1, row2, column2), which are chosen randomly every time the agent attempts to verify a transceiver.

The submarks requested by the CIA are located in the cells (r_1, c_1) y (r_2, c_2). The transceiver receives the four parameters and applies the corresponding modular reduction (mod x) to obtain the coordinates of the matrix. In this way, the transceiver obtains the values w_1 and w_2 located in these positions and multiplies them. The transceiver applies the following function:

$$f_2(y) = \{(y * t) + (w1 * w2)\} \quad (2)$$

where t is a random prime number that changes every time the transceiver uses the function and y is an integer that represents the previously established module that is to be used in the function. This module is known by the monitor and is used to mask the submarks. Thus, if the function's result is obtained by a malicious user, no information will be revealed. The CIA receives the result and forwards it to the monitor. The monitor applies the reverse process using the module (y) and compares this result against a second result. This second result is obtained by applying the same module to the result of w1 * w2. If the operation does not match, it means that the transceiver has been modified and should therefore be considered malicious. Otherwise, it is highly probable that the agent has not been modified.

4.1 Example

Next we explain the process of marking a transceiver. The first step is to issue a matrix to be used as a mark with a size calculated by m x n (each matrix is unique to each transceiver). Each cell contains different prime numbers (w_j) and these are

considered submarks. A module (x) must be assigned to be used with Function 1, which is fixed during the process. In contrast, various random values must be issued by the monitor each time the CIA begins the verification process; these values are issued to multiply the module and to select two positions in the matrix.

In this example, we use the following matrix of marks and values:

- *Matrix of marks*

Firstly, a monitor generates a transceiver and assigns a matrix to mark it. The monitor stores a copy of the matrix. The matrix remains fixed and when a CIA arrives at a host, the monitor gives it the masked coordinates, where the values to be verified are located.

	0	1	2	3
0	68813	79687	36599	98663
1	59879	16993	98689	36997
2	79657	11383	35729	21991
3	78643	41299	86323	59693

- *Fixed values used in the process*:

Variable	Value	Description
x	17	Module of the function $f_1(x)$
y	53	Module of the function $f_2(y)$

- *Random values issued every time that the CIA arrives at a host*:

The values (r_1, c_1) = (0, 2) and (r_2, c_2) = (2, 3); correspond to the positions of the values *36599* and *21991* in the matrix of marks that we are using.

Variable	Value	Descripción
p	37	Random values, which multiply to the module of function (1).
q	13	
r	7	
s	23	
t	53	Random values, which multiply to the module of function (2).
r_1	0	Row of the matrix where the first submark (w_1) to be verified is located ($r_1 < m$: where m corresponds to the number of rows in the matrix).
c_1	2	Column of the matrix where the first submark (w_1) to be verified is located ($c_1 < n$: where n corresponds to the number of columns in the matrix).
r_2	2	Row of the matrix where the second submark (w_2) to be verified is located ($r_2 < m$: where m corresponds to the number of rows in the matrix).

c_2	3	Column of the matrix where the second submark (w_2) to be verified is located ($c_2 < n$: where n corresponds to the number of columns in the matrix).

The monitor computes $f_1(x)$ and sends it to the CIA:

$$f_1(x) = \{(x * p + r_1), (x * q + c_1), (x * r + r_2), (x * s + c_2)$$
$$f_1(x) = \{17 * 37 + 0, 17 * 13 + 2, 17 * 7 + 2, 17 * 23 + 3\}$$
$$f_1(x) = \{629, 223, 121, 394\}$$

The CIA forwards the parameters to the transceiver, which uses the values and applies the reversed process to obtain the positions of the matrix where the requested values are located:

$$C = \{\text{mod}_x(629), \text{mod}_x(223), \text{mod}_x(121), \text{mod}_x(394),\}$$
$$C = \{\text{mod}_{17}(629), \text{mod}_{17}(223), \text{mod}_{17}(121), \text{mod}_{17}(394)\}$$
$$C = \{0, 2, 2, 3\}$$

these values correspond to the row 0, column 2 and row 2, column 3 of the matrix; in these positions the values 36599 and 21991 are located. The transceiver uses function $f_2(y)$. In this example, we use $y = 53$ and $t = 3571$.

$$f_2(y) = \{(y * t) + (w1 * w2)\}$$
$$f_2(y) = \{\ 3571 * 53 + (\ 36599 * 21991)\ \}$$
$$f_2(y) = \{805037872\}$$

The transceiver sends the response to the CIA, which then forwards it to the monitor. The monitor applies the reverse process and obtains the module of the received value:

$$S_1 = \{\ \text{mod}_y(f(y))\ \}$$
$$S_1 = \{\ \text{mod}_{53}(805037872)\ \}$$
$$\mathbf{S_1 = \{43\}}$$

The monitor obtains the requested submarks directly from the matrix and multiplies them to apply the corresponding module to the result:

$$S_2 = \{\ \text{mod}_{53}\ (36599 * 21991)\ \}$$
$$S_2 = \{\ \text{mod}_{53}\ (804848609)\ \}$$
$$\mathbf{S_2 = \{43\}}$$

Afterwards, the monitor verifies that the module sent by the transceiver is equal to the module of the result obtained by multiplying the requested values (36599, 21991), that is, that S1 = S2. In our example, the comparison is correct, so there is a high probability that the transceiver's integrity has not been modified.

- *Protection against replay attacks*:

In this next example, we use a matrix with a size of four rows by four columns; in this way the possible combinations of obtaining the same pair of submarks is given by:

$$\frac{n!}{w!(n-w)!} = \frac{16!}{2!(16-2)!} = 120$$

Thus, by using a matrix with 16 positions and requesting only two marks, there is a low probability of requesting the same pair of values (1/120) again. For simplicity's sake, we chose a matrix with 16 positions and low values as a mark to fill the matrix that is to be used as a module in Functions 1 and 2. Similarly, we chose only two submarks to verify the transceiver's integrity. However, it is possible to request more submarks, in which case the monitor must provide the positions of the values to be used in Function 1 and the transceiver must use the values in Function 2 to multiply them.

In spite of using a module's function to mask the results, so as to prevent a malicious host from being able to deduce the operations performed by the transceiver or monitor, it is possible that the submarks will be detected when a CIA requests them (although the probability of requesting the same combination of marks is very low). This drawback can be solved using one-time submarks, meaning that each time a submark is requested it must be blocked when the verification process finishes. Consequently, the CIA cannot request a mark twice, and when all the submarks of a given entity have been requested by a CIA, it notifies the monitor. The monitor will issue another transceiver with its corresponding matrix of marks to replace the previous one.

Table 1 shows the possible of matrix positions 16, 32, 64 and 128 when 2, 3 or 4 submarks are chosen to verify the integrity of a transceiver.

The previous verification process of a transceiver or monitor can be considered a Zero-Knowledge Proof (ZKP). The ZKP is an interactive protocol between two parts: one part provides the proof (*prover*) and the other part verifies it (*verifier*). The *prover* must convince the *verifier* that it knows the solution to a given theorem without revealing any type of additional information [10]. The *verifier* can request information about the solution several times; these questions will be different and random to avoid a sequence, so it is impossible to memorize a response. Similarly, if an attacker intercepts the exchanged messages between the prover and verifier, no confidential information will be revealed.

Table 1. Matrix positions vs set of possible verification submarks

No. positions	2 submarks	3 submarks	4 submarks
16	120	560	1,820
32	496	4,960	35,960
64	2,016	41,664	635,376
128	8,128	341,376	10,668,000

In our case, the transceiver must solve function (2) with the parameters sent by the monitor through the CIA, and the monitor must verify whether the transceiver's response is correct. There is a low probability of correctly guessing the response without knowing both the protocol operation and the operations of the corresponding

functions. Moreover, each time that the CIA requests a mark, the monitor sends different and random matrix positions. Another additional measure to protect the entity integrity of the IDS from malicious hosts is using obfuscation techniques to prevent code analysis or reverse engineering.

5 Code Obfuscation

Code obfuscation is a technique used to alter the structure of a program to obscure its reading and make it harder for unauthorized users to understand its operation. From the computer's point of view, it is simple translation to be performed and the compiler can easily process the obfuscated code. In order to perform code obfuscation, it is necessary to use an obfuscator program that transforms the application into another application that is functionally identical to the original. The obfuscator program might, for example, insert irrelevant code into loops, enter unnecessary calculations, or perform data consultations that will not be used. This technique is appropriate for protecting secret marks but it cannot prevent reverse engineering, because a programmer with enough knowledge and time could recover the algorithms and data structures of the analyzed code [11]. In this case, one strategy is to discourage the unauthorized entities by making the information analysis more expensive than the information itself.

There are several algorithms for performing code obfuscation [11], [12] and they can be applied to both static and mobile agents. In this way, if a malicious user analyzes the code of an obfuscated agent, it will not be able to understand it easily and the process will be very expensive. In our example, if an attacker obtains the used functions or the values returned by an agent, this information will not reveal important data because the function results are masked by different functions.

6 Mark Embedding

The algorithm used to embed the mark is proposed in previous literature [8]. It provides copyright protection and is used to distinguish copies of specific software: in other words, it provides the system with fingerprinting properties. This algorithm uses branch functions to generate the appropriate fingerprinting in run time. The original formulation can be summarized as two processes (embedding and extracting):

$$embed(P,AM, key_{AM}, key_{FM}) \rightarrow P', FM$$
$$recognize(P0, key_{AM}, key_{FM}) \rightarrow AM, FM$$

The first process requires the input of the program (P), the Authorship Mark (AM) and copyright and fingerprinting keys. The copyright key will be the same for all copies, but the fingerprinting key will be different. As result of this process, the correctly marked program (the monitors and transceivers in our scenario) and the corresponding fingerprinting code. On the other hand, the *recognize* function requires the input of a piece of marked code (or a function of the program) and two keys. In the output, this function retrieves the Authorship Mark (AM) and Fingerprint.

In the scenario presented in this paper, the algorithm is used to embed the matrix presented in section 4. The aim of the mark embedding process is to obtain a system

that retrieves a value from two input values (in our case, to obtain a value in the matrix from values *f* and *c*). In the embedding process we can use *f* as key_{AM}, *c* as key_{FM}, and *P* as the pointer for the piece of source code that will embed the value in row *f* and in column *c* in the mark matrix. The embedding process must be done for each value in the matrix. The *recognize* function will be adapted in the same way. The new formulation can be summarized as two processes (embedding and extracting):

$$embed(P,AM, f, c) \rightarrow P', M(f,c)$$
$$recognize(P0, f, c) \rightarrow M(f,c)$$

where *M(f* and *c)* is the value in row *f* and in column *c* of the mark matrix, and *P* and *P0* are pointers to a piece of the program.

7 Conclusions

Attacks from malicious host against agents are considered one of the most difficult problems to solve and there is no a way to avoid them. To offer a determined security level in an IDS based on agents, it is necessary to combine different techniques to at least detect a possible attack, even though it cannot be avoided. The drawback of sending an agent to a host is that the host could be malicious and attack the agent, because it has complete access, not only to the agent data but also to the code. With our proposal of using a matrix of marks and subsequently asking for a set of submarks to be verified, it is possible to determine whether an agent has been modified and whether it was executed correctly within a certain period of time. By masking the results of the functions used by the mobile IDS entities (transceivers, monitors), no information will be revealed if an attacker gets as far as obtaining the functions or the results received and used by them.

Code obfuscation techniques hinder the efforts of a malicious user and make it very difficult to understand a transceiver or monitor operation. In addition, even if the malicious user was able to understand the code or obtain information, the process would be so expensive that the attack would not be justified.

Acknowledgments. This work has been supported in part by the projects TSI2005-07293-C02-01 (SECONNET), TSI2007-65393-C02-02 (ITACA) and CONSOLIDER CSD2007-00004 "ARES", funded by the Spanish Ministry of Science and Education, as well as the grant to consolidated research groups, Generalitat de Catalunya, 2005 SGR 01015.

References

[1] Nwana, H.S.: Software Agents: An Overview. Knowledge Engineering Review 11(3), 1–40 (1996)

[2] Goyal, B., Sitaraman, S., Krishnamurthy, S.: Intrusion Detection Systems: An overview. SANS Institute 2001, as part of the Information Security Reading Room (2003)

[3] Balasubramaniyan, J.S., Garcia-Fernandez, J.O., Isacoff, D., Spafford, E., Zamboni, D.: An Architecture for Intrusion Detection using Autonomous Agents. In: Proceedings of 14th Annual Computer Security Applications Conference, pp. 13–24 (1998)

[4] Jansen, W., Mell, P., Karygiannis, T., Marks, D.: Mobile Agents in Intrusion Detection and Response. In: Proc. 12th Annual Canadian Information Technology Security Symposium, Ottawa (2000)
[5] Lange, D., Oshima, M.: Programming and deploying java mobile agents with agle. Addison-Wesley, Reading (1998)
[6] Denning, D.E.: An intrusion detection model. IEEE Transactions on Software Engineering 13(2), 222–232 (1987)
[7] Jansen, W.A.: Countermeasures for mobile agent security, Computer communications. Special Issue on Advanced Security Techniques for Network Protection 25(15), 1392–1401 (2002)
[8] Myles, G., Jin, H.: Self-validating branch-based software watermarking. In: Barni, M., Herrera-Joancomartí, J., Katzenbeisser, S., Pérez-González, F. (eds.) IH 2005. LNCS, vol. 3727, pp. 342–356. Springer, Heidelberg (2005)
[9] White, J., Niinimäki, M., Niemi, T.: The 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid, Japan (2003)
[10] De Santis, A., Persiano, G.: Zero-knowledge proofs of knowledge without interaction Foundations of Computer Science. In: Proceedings of 33rd Annual Symposium, October 24-27, 1992, pp. 427–436 (1992) Digital Object Identifier 10.1109/SFCS.1992.267809
[11] Collberg, C.S., Thomborson, C.: Watermarking, Tamper-Proofing, and Obfuscation – Tools for Software Protection Software Engineering. IEEE transactions 28(8), 735–746 (2002)
[12] Linn, C., Debray, S.: Obfuscation of executable code to improve resistance to static disassembly. In: Proceedings of the 10th ACM Conference on Computer and Communications Security (CCS), October 2003, pp. 290–299 (2003)
[13] Collberg, C., Myles, G.R., Huntwork, A.: Sandmark-A tool for software protection research. Security & Privacy Magazine 1(4), 40–49 (2003)
[14] Páez, R., Satizábal, C., Forné, J.: Cooperative Itinerant Agents (CIA): Security Scheme for Intrusion Detection Systems. In: International Conference on Internet Surveillance and Protection, 2006. ICISP (2006) Digital Object Identifier 10.1109/ICISP.2006.6. ISBN: 0-7695-2649-7