Platform of Intrusion Management

Design and Implementation

by

Joaquín García Alfaro

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Computer Science

Supervised by

Joan Borrell Viader
Frédéric Cuppens

Abstract

Since computer infrastructures are currently getting more vulnerable than ever, traditional security mechanisms are still necessary but not sufficient. We need to design effective response techniques to circumvent intrusions when they are detected. We present in this dissertation the design of a platform which is intended to act as a central point to analyze and verify network security policies, and to control and configure – without anomalies or errors – both prevention and detection security components. We also present in our work a response mechanism based on a library that implements different types of counter-measures. The objective of such a mechanism is to be a support tool in order to help the administrator to choose, in this library, the appropriate counter-measure when a given intrusion occurs. We finally present an infrastructure for the communication between the components of our platform, as well as a mechanism for the protection of such components. All these approaches and proposals have been implemented and evaluated. We present the obtained results within the respectives sections of this dissertation.

This thesis has mainly been funded by the Agency for Administration of University and Research Grants (AGAUR) of the Ministry of Education and Universities (DURSI) of the Government of Catalonia (reference number 2003FI00126). The research was jointly carried out at the *Universitat Autònoma de Barcelona* and at the *Ecole Nationale Supérieure des Télécommunications de Bretagne*.

Keywords: Security policies, intrusion detection, response, counter-measures, event correlation, communication *publish/subscribe*, access control, components protection.

Résumé

Aujourd'hui les systèmes informatiques sont plus vulnérables aux activités malveillantes qu'auparavant. C'est pour cela que l'utilisation des mécanismes de sécurité traditionnaux est encore nécessaire mais pas suffisante. Nous devons élaborer des méthodes efficaces de détection et de réponse aux attaques afin d'arrêter les événements détectés. Nous présentons dans cette thèse la conception d'une architecture générale qui agira en tant que point central pour analyser et vérifier des politiques de sécurité réseaux, et pour contrôler et configurer – sans anomalies ou erreurs de configuration – des composants de sécurité préventifs et de détection. Nous présentons également un mécanisme de réponse basé sur une bibliothèque de différents types de contremesures. L'objectif de ce mécanisme est d'aider l'administrateur à choisir dans cette bibliothèque la contremesure la mieux adaptée quand une intrusion est détectée. Nous finissons par la présentation d'une infrastructure pour la communication des composants de notre plateforme, et d'un mécanisme pour la protection des composants de celle-ci. Toutes les propositions et approches introduites dans cette thèse ont été implémentées et évaluées. Nous présentons les résultats obtenus dans les sections respectives de cette dissertation.

Cette thèse a été principalement financée par l'Agence de Gestion d 'Aides Universitaires et de Recherche (AGAUR) du Département d'Universités, Recherche et Société de l'Information (DURSI) de la Généralité de Catalogne (numéro de référence 2003FI00126). Les travaux ont été réalisés conjointement à l'Université Autonome de Barcelone et à l'Ecole Nationale Supérieure des Télécommunications de Bretagne.

Mots clés: Politiques de sécurité, détection d'intrusion, contremesures, corrélation d'événements, communication *publish/subscribe*, contrôle d'accès, protection des composants.

Resumen

Puesto que los sistemas informáticos son cada vez más vulnerables a actividades deshonestas, los mecanismos tradicionales de seguridad son todavía necesarios, pero no suficientes. Es necesario elaborar nuevos métodos de detección y de respuesta de manera que sea posible detener acciones de ataque tan pronto como sean realizadas. En esta tesis se presenta el diseño de una arquitectura de carácter general que pretende ser utilizada tanto para la realización de tareas de análisis y verificación de políticas de seguridad en red, como para controlar y configurar – sin anomalias ni errores de configuración – componentes de seguridad preventivos y de vigilancia. Se presenta también en esta tesis un mecanismo de respuesta basado en librerías de contramedidas. El objetivo de este mecanismo es ayudar al administrador a escoger posibles respuesta tan pronto como las acciones de ataque vayan siendo detectadas. Por último, se introduce también en esta tesis el diseño de una infraestructura para la comunicación entre los componentes de nuestra plataforma, y un mecanismo para la protección de dichos componentes. Todas las proposiciones y propuestas han sido implementadas y evaluadas a lo largo de nuestro trabajo. Los resultados obtenidos son presentados en las respectivas secciones de esta disertación.

Esta tesis ha sido principalmente financiada por la Agencia de Gestión y Ayudas Universitarias y de Investigación (AGAUR) del Departamento de Universidades, Investigación y Sociedad de la Información (DURSI) de la *Generalitat de Catalunya* (num. de referenci 2003FI00126). El trabajo ha sido conjuntamente realizado en la *Universitat Autònoma de Barcelona* y la *Ecole Nationale Supérieure des Télécommunications de Bretagne*.

Palabras clave: Políticas de seguridad, detección de intrusos, contramedidas, correlación de eventos, comunicación *publish/subscribe*, control de acceso, protección de componentes.

Acknowledgements

This work has been supported by the Spanish Government Commission CICYT, through the projects TIC2001-5108-E and TIC2003-02041, and by the Ministry of Education and Universities (DURSI) of the Government of Catalonia, with its grants 2001SGR-219, 2003-FI00126, 2004ZAOCE1263, 2005BE77, and 2006BE569.

My first thanks go to my family, who always gave me support and encouragement. My thanks also go to my advisers for giving me the chance to work under their direction and expert tutelage. I also wish to thank my close friends and colleagues, for giving me their kindness, enthusiasm, and support during my entire academic career. I would like to list here a long list of names and situations. But, since I am afraid I might forget some of them, and rather than offend anyone, I will settle for offending everyone. Finally, I would like to specially thank Katell Himeur, my significant other, without whose help this dissertation would never have been completed.

Contents

Chapter 1

Introduction

"Fiery the angels rose, and as they rose deep thunder roll'd;
Around their shores: indignant burning with the fires of Orc."
– WILLIAM BLAKE

This dissertation is the final result of four years of Ph.D. studies in Computer Science at the *Universitat Autònoma de Barcelona* (UAB, Campus of Bellaterra, Spain) and the *Ecole Nationale Supérieure des Télécommunications de Bretagne* (ENST Bretagne, Campus of Rennes, France), under the co-supervision of Professor Dr. Joan Borrell (UAB) and Professor Dr. Frédéric Cuppens (ENST Bretagne).

I obtained in Summer 2000 the degree of *Bachelor of Science* (B.Sc.) in *Computer Management Engineering*, and in Summer 2002 the degree of *Master of Science* (M.Sc.) in *Computer Engineering* – both at the UAB and with two extraordinary awards for excellent performance as undergraduate student. In September 2004 I presented my DEA (*Diploma of Advanced Studies*) thesis at the *Information and Communications Engineering Department* (dEIC) of the UAB, with the final score of *A with honors*.

I started the Ph.D. program of Computer Science of the UAB in September 2002 and became a teacher assistant at the *Information and Communications Engineering Department* (dEIC), both at the UAB. In January 2003 I obtained a Catalan Government Department DURSI grant of four years to fund my Ph.D. studies (reference number 2003FI126). As a part of my Ph.D. program I took courses on advanced techniques in cryptography and coding, heuristic methods in optimization, advanced network security techniques, and other subjects. As teacher assistant in the Computer Science graduate studies of the UAB, I taught several subjects and courses, such as Computer Networks, GNU/Linux administration, and Computer Network Security. It is worthy of note that some texts related to the disciplines I taught were edited by the *Fundació Universitat Oberta de Catalunya* [Herrera et al., 2004b, Herrera et al., 2004c]. An extended version of these documents, [Herrera et al., 2004a], was later licensed as a free GPL document.

In February 2004 I obtained a Catalan Government Department DURSI grant (reference number 2004ZAOCE1263) to collaborate with Professor Dr. Frédéric Cuppens in the *Department of Networks, Security, and Multimedia* (RSM) at the Rennes campus of ENST Bretagne. In September of 2004, after having demanded the co-supervision of my Ph.D. thesis between the UAB and the ENST Bretagne, I extended my collaboration with the group of Professor Cuppens at the campus of Rennes. Since then, I have carried out three internships at the ENST Bretagne. These internships, and other related research activities, have been partially funded by the Catalan Government Department DURSI, with its grants 2001SGR-219, 2005BE77, and 2006BE569; the Spanish Government Commission CICYT, through its grants TIC2001-5108-E and TIC2003-02041; and the French ministry of research, under the project *ACI DESIRS*.

Background and Motivation

The research done during these four years of Ph.D. studies has mainly been about computer and network security technologies. Despite the recent advances in this field, such as firewalls, encrypted communications, and authentication mechanisms, there may always be errors or flaws that can be exploited for unauthorized parties. Moreover, the proliferation of Internet access to most network devices, the increased mobility of these elements, and the introduction of network-enabled applications have rendered traditional network-based security infrastructures vulnerable to a new generation of attacks.

The use of distributed and coordinated techniques in these kind of attacks is getting more common among the attacker community, since it opens the possibility to perform more complex tasks, such as *coordinated port scans* and *distributed denial of service* (DDoS) attacks. These techniques are also useful to make their detection more difficult and, normally, these attacks will not be detected by exclusively considering information from isolated sources of the network. Different events and specific information must be gathered from all of these sources and combined in order to identify the attack. Information such as suspicious connections, initiation of processes, addition of new files, sudden shifts in network traffic, and so on, have to be considered.

Intrusion attacks, as defined in [Cuppens and Ortalo, 2000], are those combination of unauthorized actions performed by a malicious adversary to violate the security policy of a target computer system or a network domain. Therefore, one may define the detection process of intrusion attacks as the sequence of elementary actions that should be performed in order to identify and respond to those unauthorized actions, being the *intrusion detection systems* (IDSs) the most important component to perform such a process.

The initial motivation of our work was the design of a policy-based framework for managing detection and prevention of intrusion attacks on distributed heterogeneous systems. This framework takes the inspiration of policy management approaches, such as the one proposed in [Moore et al., 2001], where the *policy decision point* is a network policy server responsible for supplying policy information for network devices and applications, and the *policy enforcement point* are network security components, such as *firewalls* and *intrusion detection systems* (IDSs), in charge of both detecting and reacting to intrusion attacks. Thus, it was decided to model intrusions in our work as prohibited activities that lead to security policy violations. This way, it would be feasible to specify security requirements and define mechanisms to translate these requirements into concrete filtering rules and detection signatures (to automatically configure firewalls and intrusion detection systems, for instance). It was hence expected to design a general framework to manage and reason with both preventive and detective security components. Furthermore, it was also expected to provide through our approach the necessary mechanisms to guarantee the interoperability and protection of those security components used in a network security infrastructure.

The starting point of our work was the recognition process of the intruder's intentions proposed in [Cuppens and Miège, 2002]. We further extend those results in a number of ways. First, in [García et al., 2006f, García et al., 2006d] we proposed a set of algorithms to analyze and rewrite network security policies in order to properly deploy security mechanisms over multi-component setups free of anomalies and misconfiguration. This approach allows security officers to verify and validate the correctness of a distributed network security policy in an efficient way – i.e., assuring the security officers that the new configuration is free of policy anomalies. Second, we proposed in [García et al., 2005e, García et al., 2005a] a decentralized exchange of audit information between those security components. Through the use of a publish/subscribe model, our approach allows such an exchange across multiple nodes of a cooperative network. Third, we presented in [Cuppens et al., 2006a, García et al., 2004b] a response approach for the selection of counter-measures and reporting of diagnoses. This approach could be further used, for example, to properly reconfigure the global security policy by means of our algorithms, guaranteeing that the deployment of the new security policy – once reconfigured – continues free of anomalies. We finally proposed in [García et al., 2006b, García et al., 2005b] the use of a protection model based on a kernel based access control, which allows us the protection of the set of components of the complete framework.

Contributions

Some parts of this dissertation have been partly published at national and international conference proceedings, national and international journals, and JCR publications. We list in the following these contributions: [García et al., 2004a; García et al., 2004b; García et al., 2004c; Cuppens et al., 2005a; Cuppens et al., 2005b; Castillo et al., 2005a; Castillo et al., 2005b; García et al., 2005a; García et al., 2005b; García et al., 2005c; García et al., 2005d; Cuppens et al., 2006a; Cuppens et al., 2006b; García and Barrera, 2006; García et al., 2006a; García et al., 2006b; García et al., 2006c; García et al., 2006d; García et al., 2006e; García et al., 2006f].

It is also worthy of note that the work done during these four years of Ph.D. has included the collaboration on the development of many software tools, including the implementation of an audit console for the discovering of policy anomalies (cf. Chapter 4, Section 4.4); a set of libraries for the exchange of IDMEF messages based on a publish/subscribe model (cf. Chapter 5, Section 5.3); a set of sensors and attack scenarios for the Cooperative Intrusion Detection Framework, CRIM, developed at the ENST-Bretagne (cf. Chapter 6, Section 6.4); and a kernel based access control for the protection of processes and resources of the platform (cf. Chapter 7, Section 7.2).

Organization

This dissertation is organized as follows. Chapter 2 introduces the basic concepts and properties of computer security, and surveys traditional mechanisms than can be used to enforce the security of a network system. We also discuss on this chapter the necessity of complementary mechanisms, and we then introduce the *intrusion detection systems* (IDSs) as the most important component to perform such a process. Chapter 3 presents related work that falls into similar research of our dissertation. The remainder of chapters is then dedicated to present our main contributions. More specifically, Chapter 4 presents our set of algorithms for the deployment and analysis of network security policies. Chapter 5 introduces our decentralized infrastructure to share messages and audit information between the components of our platform. Chapter 6 describes our approach for the correlation of this audit information and the selection of counter-measures. Chapter 7 overviews the use of a security mechanism to handle the protection of the security components of our framework. Chapther 8 finally concludes this dissertation and gives an outlook on future work.

Chapter 2

Security in Computer Networks

”The security systems have to win every time, the attacker only has to win once.”
– DUSTIN DYKES (THE ART OF INTRUSION)

Computer and network security is the field of computer science that concerns the control and protection of data transmission over networked systems [Stallings, 1995]. Therefore, we assume that the security of network communications must guarantee that the stream of data flowing from a source object to a destination entity is restricted to only those parties that are authorized to have access. Unauthorized actions like *interruption*, *interception*, or *injection* of information must be avoided.

Interruption refers to those actions against computer networks such that an unauthorized party makes unavailable the source of the communication (or even the communication channel itself) in order to prevent the flow of data getting the receiver (e.g., denial of service attacks); *interception* refers to those actions in which an unauthorized entity gets access to the flow of data in order to *modify* or *disclose* such an information (e.g., passive or active wiretapping, eavesdropping, sniffing, or snooping attacks); and *injection* refers to those actions in which an unauthorized entity inserts information into the system without having the source’s object doing anything (e.g., replay and spoofing attacks).

Those three examples of unauthorized actions listed above violate, respectively, the main security properties that any computer system must guarantee – often referred in the literature as *confidentiality*, *integrity*, and *availability* [Stallings, 1995]. *Confidentiality* concerns the prevention and detection of unauthorized disclosure of information (e.g., passive interception of information with a further disclosure of such an information); *integrity* concerns the prevention and detection of unauthorized modification of information (e.g., active interception of information with a further modification of the intercepted information); and *availability* concerns the prevention and detection of unauthorized withholding of information or resources (e.g., interruption of service).

Although these three security requirements must be considered as mandatory, we may also consider some other security properties, such as *non-repudiability* and *authentication*. *Non-repudiability*, for instance, prevents that neither the source's object nor the destination's entity may deny the transmitted flow of data. This way, once a message has been transmitted, the source's object can prove that such a message has been received by the destination's entity; and the destination's entity can prove that the message it received has been sent by the source's object. *Authentication*, on the other hand, ensures that the origin of an information is correctly identified, with an assurance that the identity is not false – i.e., it ensures that the information is authentic.

In order to ensure the security requirements of a computer network, a *security policy* must be defined by the security officer of such a network. A *security policy* is a set of rules stating what is permitted and what is not permitted in a system during normal operation [International Organisation for Standardization, 1989]. Thus, in the security policy is defined the complete set of security requirements for the system. As we can see in Figure 2.1, before to define the set of security requirements into the security policy, a previous *analysis of threats* must be done.

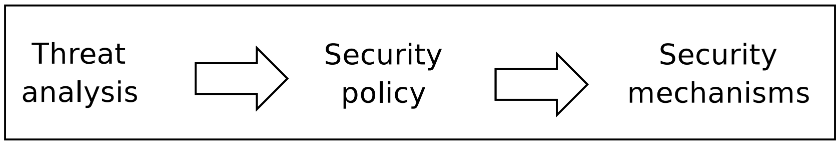


Figure 2.1: The role of the security policy.

The *analysis of threats* is the process where all the possible risks related to the networked system are identified, and a list containing these threats and the severity of each threat is reported. This list is then used to establish the security policy, i.e., the list of rules that state the security requirements to protect the network system. Once defined the security policy, the security officer must decide which *security mechanisms* to use to enforce the security policy. *Security mechanisms* are the technical solutions used to implement the security policy in the networked system, i.e., to defend the network against unauthorized actions that attempt to violate its security policy.

Because this dissertation is focused on security mechanisms management, we oversee in the rest of this chapter such mechanisms. We first present in the following section the use of cryptography and firewalls as traditional mechanisms to prevent unauthorized parties from violating the security policy of the network. We then shortly discuss the necessity of complementary measures to these traditional mechanisms, in order to detect and report those actions that bypass (or that attempt to bypass) the security policy of the system. We finally present in the last section the use of intrusion detection systems (IDSs) as a complementary third building block (together with firewalls and cryptography) to guarantee the security of computer network systems.

2.1 Traditional Security Mechanisms

Cryptography

Cryptography is the field of mathematics and computer science that concerns the managing of information to keep messages secure [Schneier, 1996]. It allows us to modify a flow of information in a way that, even if an unauthorized party can access to such an information, it will be unusable for its use. To do so, the original piece of information is transformed into a format that hides its substance during a process called encryption. This transformation is lossless, since the original message must be recovered under all circumstances. Once transformed, the new piece of information may be transported over an untrusted communication channel and resist to actions that try to violate its confidentiality and integrity – it cannot guarantee, however, the availability of the information, since an unauthorized entity may still interrupt the communication between both source and destination. The operation of turning back the transformed information into the original one is called decryption.

In order to perform the decryption process, modern cryptographic algorithms assume the possession of a secret piece of information (often referred as *key* in the literature). Depending on the keys that are used, one can distinguish the following two approaches: *symmetric* and *asymmetric* cryptography. Symmetric cryptography, on the first hand, requires that the sender and the receiver agree on a single key. This key must be safely exchanged before the encryption process starts, and should remain secret during the whole process. Asymmetric cryptography, on the other hand, elegantly solves the key exchange problem by using two different keys, one called the public key, the other one called the private key. More specifically, the receiver generates its public/private key pair and announce its public key. Anyone can obtain the public key and use it to encrypt messages that only the receiver, with the corresponding private key, is able to decrypt. Furthermore, asymmetric cryptography can be easily used to include authentication during the communication process. In this way, a sender can sign a message by encrypting it with its own private key; and anyone with access to the corresponding sender's public key can verify the signature [Diffie, 1988].

A drawback when using asymmetric cryptography is that we must prove the authenticity of the public key. Otherwise, an attacker could be able to offer its own public key to the sender to perform a man-in-the-middle attack and read, or even modify, encrypted communications. In order to solve this problem, we can use *digital certificates*. Digital certificates bind together a public key, additional information of its owner (such as its name, organization, and so on), and provide the related data signed by a trusted third party – also known as *certification authority* or *CA*. This CA guarantees that the public key belongs to such a person or institution. In turn, this CA is usually verified by a higher level CA in a hierarchical fashion, such as the public key infrastructure (PKI). In the PKI, for example, one can verify the authenticity of a public key by verifying the authenticity of the involved chain of CAs until the hierarchy's top – a special self-signed CA [Schneier, 1996].

Firewalls

Similarly to firewalls in building construction, a computer firewall is defined as a network security device that can perform access control at network level – i.e., to limit and regulate the access to critical resources in network systems [Bellovin and Cheswick, 1994]. Therefore, firewalls are security mechanisms that allow the prevention of certain attacks before they can actually reach and affect the target. This is done by identifying or authenticating the party that requests a resource and checking its permissions against the rights specified for the demanded object in the security policy. To do so, firewalls are typically deployed to filter traffic between different zones of the network, as well as to police their incoming and outcoming interaction with other networks (e.g., Internet). Though firewalls may also implement other functionalities not related with the security policy, such as Network Address Translation (NAT), it is not the purpose of this section to cover these functionalities.

A single firewall is not always suitable for the protection of a complete system. Networks often consist of several nodes which need to be publicly accessible for untrusted parties, and private nodes that should be completely protected against connections from the outside. Those networks would benefit from a separation between these two groups. Otherwise, if an unauthorized party compromises a publicly accessible node behind a single firewall, all the other machines can also be attacked from this open point within the network. To prevent this, one can use several firewalls and the concept of a demilitarized zone in between. Hence, the security administrator may partition, for instance, its network into three different zones: a demilitarized zone (DMZ for short), a private network and a zone for security administration. In this case, one firewall separates the outside network from the segment with the publicly accessible nodes; and a second firewalls separates this area from the rest of the network. The second firewall can be configured in a way that denies all incoming connection attempts. Thus, even if an intruder compromises a host in the first segment, he is now unable to immediately attack the rest.

According to [Stallings, 2002], one may classify firewalls into the following three categories: packet-filtering router, application-level gateway, and circuit-level gateway. A packet-filtering router, on the first hand, is a forwarding device that applies a set of filtering rules to each incoming IP packet flowing in both in- and out-going directions. Through these filtering rules, it decides whether accept (i.e., forward) or deny such a packet. Each filtering rule typically specifies a *decision* (e.g., accept or deny) that applies to a set of *condition* attributes, such as protocol, source, destination, and so on. Filtering rules are organized in a list with a certain default policy enabled. Every incoming packet is compared to the rules starting at the head of the list until the first list entry matches. In this case, the corresponding decision is taken. When no matching rule can be identified, the default policy is consulted. It can either be an open or close default policy. When a default close policy is enabled, the packet is simply dropped; otherwise it is accepted and forwarded. A close policy demands the security officer to explicitly specify the publicly accessible services. An open policy, on the other hand, requires the security officer to explicitly specify each known threat for the network and then deny associated connections.

An application-level gateway, also called proxy server, acts as a relay on the application level. The user contacts the gateway which in turn opens a connection to the intended target (on behalf of the user). It completely separates the inside and outside networks at the network level and allows authentication of the user who requests a connection and session-oriented scanning of the exchanged traffic up to the application level data. This feature makes application gateways more secure than packet filters and offers a broader range of log facilities. The overhead of such a setup, however, may cause performance problems under heavy load. To solve this disadvantage, circuit-level gateway may be used as a hybrid variant between packet filters and proxy servers.

This third type of firewall (i.e., circuit-level gateways) can act as a stand-alone system, similar to a packet filter at the network level. It can also act as an application-level gateway performing specialized functions for certain applications. Hence, one may use it to authorize connections to a specific target machine (as a proxy server does), but perform network-level filtering (without examining the contents) once the connection has been set up. The security function of a circuit-level gateway consists of determining which connections are allowed. Circuit-level gateways are often used in those situations in which one may trust internal users. Thus, the circuit-level gateway is configured to act as a proxy server on incoming connections, and as a packet filter on outgoing data.

2.2 Necessity of Complementary Technologies

Although the use of cryptography and firewalls allows us to minimize the number of potential targets, it cannot defend those data or resources that must be publicly accessed (i.e., data or resources that can neither be completely protected by cryptography nor by firewalls). This fact leads to a situation where it is still possible for unauthorized parties to get protected components through vulnerabilities within unprotected resources (and so evading firewalls and cryptography). Indeed, although the installation of updates and patches to those unprotected components can minimize their vulnerabilities, there may always be errors or flaws that can be exploited for unauthorized parties.

The Morris worm incident, for instance, showed in 1988 the possibility of attacking the availability of the majority of components of Internet by exploiting known vulnerabilities in Unix sendmail, Finger, rsh/rexec and weak passwords [Spafford, 1991]. The Mitnick attack, on the other hand, showed in 1994 the possibility of breaking the confidentiality and integrity of information. Two different attack mechanisms were used. IP source address spoofing and TCP sequence number prediction were used to gain initial access to a diskless workstation being used mostly as an X terminal. After root access had been obtained, an existing connection to another system was hijacked [Northcutt, 2002].

Although network security technologies have efficiently evolved since then, the number of vulnerabilities and attack tools continues to increase [Householder et al., 2002]. Since 1999, with the advent of distributed tools, attackers have been able to manage and coordinate attacks across many Internet systems. In year 2000, for example, a Distributed Denial of Service (DDoS) attack stopped several commercial sites, including Yahoo and CNN, from functioning normally, although they were protected by firewalls and cryptography-based mechanisms. Today, coordinated tools scanning for potential victims and compromising vulnerable systems are more active than ever. This situation shows the inadequacy of the use of cryptography- and prevention-based mechanisms as single techniques to guarantee the security of a networked system, and leads to the necessity of additional defense mechanisms to cope with attacks when this first line of defense (i.e., cryptography and firewalls) has been evaded.

Intrusion detection systems (IDSs) were originally proposed in 1980 as a complement to traditional security mechanisms. In [Anderson, 1980], an intrusion is defined as a violation of the security policy of a system. Similarly, intrusion detection techniques are defined as those mechanisms that are developed to detect the violation of the system security policy. Before the formal definition of intrusion detection and intrusion detection systems, network administrators performed detection activities by monitoring system activities and looking for intrusive or unusual actions. Although this early form of intrusion detection was effective enough at the time, it was ad hoc and not scalable [Kemmerer and Vigna, 2002]. Current intrusion detection techniques are also based on the assumption that intrusive and unusual activities are clearly different from normal and authorized activities. Thus, intrusive and unusual activities may be detectable and reported [Denning, 1987].

The use of intrusion detection systems is currently considered as a third line of defense for computer and network systems. Hence, intrusion detection must be considered as a complementary approach to traditional security mechanism such as cryptography and firewalls, not as a replacement one. The use of IDSs along with firewalls, for example, can considerably enhance the protection of networked systems by preventing and detecting those actions that bypass (or that attempt to bypass) the security policy of a given system. In the following section we present a more detailed overview of intrusion detection systems.

Intrusion Detection Systems

An intrusion detection system (IDS) can be defined as the tools, methods, and resources to help identify, assess, and report unauthorized activity against a target network. According to [Debar et al., 1999], an IDS has to fulfill the requirements of accuracy (it must not confuse a legitimate action with an intrusion), performance (its performance must be enough to carry out real-time intrusion detection), completeness (it should not fail to detect an intrusion), fault tolerance (the IDS must itself be resistant to attacks) and scalability (it must be able to process the worst-case number of events without dropping information).

Currently, there are a great number of publications related to the design and implementation of intrusion detection systems that detect and prevent intrusion attacks. Despite the differences between all these contributions and the different mechanisms used to accomplish the basic requirements pointed out above, we can identify the following components in most IDS' architectures (cf. Figure 2.2): *sensors*, *analyzers*, *managers*, and *effectors*. These components are explained in detail in the following sections.

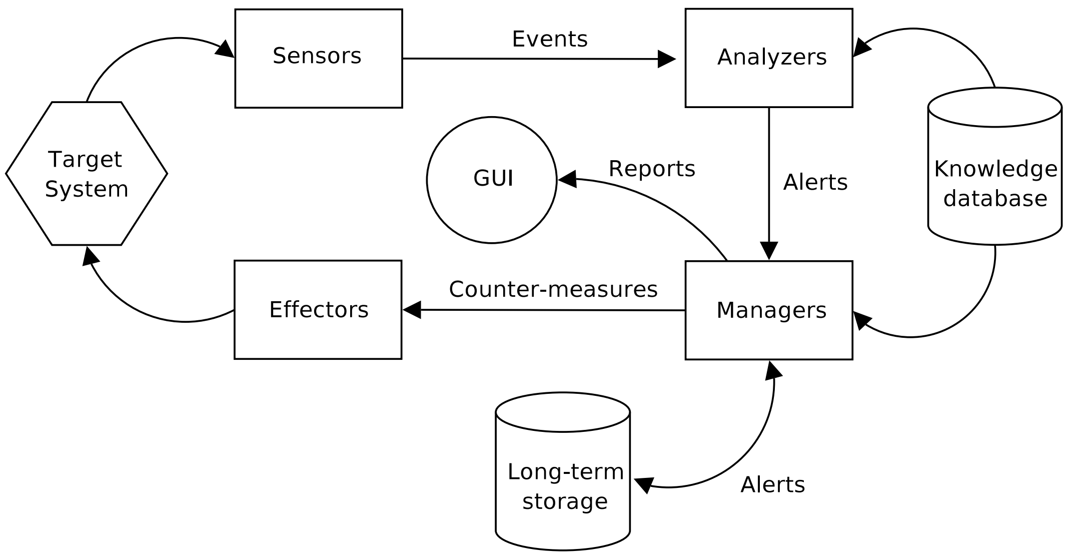


Figure 2.2: Components of a standard intrusion detection system.

Collection of Audit Information

Sensors are the components of an IDS in charge of collecting audit information. They gather data from the system that is being protected and thus generating events by preprocessing the collected data. These components are generally classified in the literature by the location where they are placed. The two main categories by means of this classification holds as host- and network-based sensors.

Host-based sensors (often referred in the literature as host-based IDSs or HIDSs) generate events by using information produced by the host operating systems, such as hosts audit trails, shell command history, or system calls. In UNIX-like operating systems, for example, audit trails may be directly extracted by the syslog's audit service. The syslog facility [Lonvick, 2001] is a de facto standard for forwarding log messages in UNIX-like operating systems. It allows security administrators to log valuable information that may be used for intrusion detection purposes. Shell command history, on the second hand, allows security administrators to manage the stream of commands, and their arguments, executed by unauthorized users or programs [Lee et al., 1999]. The sequences of system calls executed when running processes, on the third hand, may also be used by security administrators in order to catch valuable information. In [Hofmeyr et al., 1998], for example, a sensor for detecting intrusions by tracing sequences of system calls is presented.

As a specific case of host-based sensors, we can also consider those sensors that collect information not just a host operating level, but also at application level. This information at application level is usually obtained either using syslog facility (already pointed out above) or by implementing specific audit mechanisms within the audited applications. In [Almgren and Lindqvist, 2001], for instance, the authors present an application-based sensor integrated as an extension mechanism of the Apache web server to report audit information regarding the behavior of the web server.

Network-based sensors (often referred in the literature as network-based IDSs or NIDSs) collect information from network traffic in order to gather information that may point out to unauthorized actions, such as buffer overflows, stealth port scans, CGI attacks, SMB probes, OS fingerprinting attempts, and much more [Roesch, 1999]. They are not only limited to inspecting incoming network traffic. Often, valuable information about an ongoing intrusion can be retrieved from outgoing or local traffic as well. Some attacks might even be staged from the inside of the monitored network or network segment, and are therefore not regarded as incoming traffic at all. Network-based sensors are very easy to deploy, and have a minimal impact – compared to host-based sensors – on the monitored hosts. Nonetheless, changes in network technology (such as encrypted communications, switched networks, and high-speed links) may impair the usefulness of network-based sensors.

Analysis of Audit Information

Analyzers are the components of an IDS responsible for processing the audit information (i.e., events) gathered by their associated sensors. Through this audit information, analyzers infer possible violations of the security policy in form of alarms or alerts (i.e., audit data at a higher level of abstraction). These components are generally classified in the literature by the processing method in which they perform such a detection. The two main categories by means of this classification holds as *misuse*- and *anomaly-based analyzers*.

The first category, *misuse-based analyzers*, attempts to identify unauthorized activity by searching for specific known patterns (e.g., known attacks or system vulnerabilities), called signatures, in their input stream. Thus, any action that conforms to the pattern of a known attack or vulnerability is considered intrusive. The audit information collected by the associated sensors is compared with the content of a database of signatures and, if a match is found, an alert is generated. Events that do not match any of the attack models are considered part of legitimate activities. Several approaches have been proposed for performing misuse-based detection, such as [Kumar and Spafford, 1994, Ilgun et al., 1995, Mounji, 1997, Porras and Neumann, 1997, Lindqvist and Porras, 1999, Roesch, 1999].

The second category, *anomaly-based analyzers*, attempts to identify certain deviations from the expected behavior of a subject (e.g., a user, an application, a host, or a network) that indicate hostile activities against the protected network. The expected behavior may be learned by observing the subject under normal operations, or specified based on a priori knowledge. Numerous attempts have been made to build anomaly detection models, including machine learning and data mining approaches [Teng et al., 1990, Fox et al., 1990], statistical approaches [Javits and Valdes, 1993, Anderson et al., 1995b], and specification-based approaches [Ko et al., 1997, Sekar et al., 2002].

Similarly to virus analyzers, the advantage of misuse-based analyzers is its reliable detection of known attacks or vulnerabilities. Hence, most commercial intrusion detection systems tend to include a misuse-based approach for the analysis of the audited information. Nevertheless, it is important to combine both approaches, in order to detect those attacks that can remain undetected by slightly deviating its pattern.

Correlation and Response Mechanisms

A recent trend in intrusion detection is the cooperation of different security components in order to increase the detection rate. The idea is to use multiple sensors and analyzers within the intrusion detection system. Then, these low level components are intended to report their events and alarms to higher level components which, in turn, correlate the analysis results and the corresponding alerts. The cooperation between those components may be achieved by complementing the coverage of different sensors – i.e., combining multiple host- and network-based sensors – and complementing the analysis with different detection techniques – i.e., by combining anomaly- and misuse-based analyzers.

Though this approach effectively increases the detection's rate, it also increases the number of alerts to process, as well as the rate of false alarms. Managers are the components in charge of managing this process, by using higher-level descriptions of the policy violations, and achieving a more global view of the system and its security issues. During the execution of the correlation process, the initial set of alerts is first normalized and pre-processed. The resulting alerts are then fused and aggregated into new alerts, and finally correlated in order to reconstruct and verify possible attacks scenarios.

Managers may also be responsible for reacting on detected security violations by selecting appropriate counter-measures to react to such violations, and reporting these counter-measures to a set of effectors. Effectors are the components in charge for initiating actions to neutralize the effects of the violation once detected. These actions can either be automated, acting against the attacker when the intrusion is still in progress, or involve human interaction by raising, for example, alarms or notifications to warn a system administrator. Managers may finally perform some other administrative functions such as sensor and analyzer configuration, data consolidation, reporting, and so on. By implementing a database manager, for example, it is possible to store all the data generated by the IDS to enable the correlation of alerts which occurred at different time periods.

Since this dissertation is focused on intrusion management by means of component co-operation and correlation of alerts, the following chapter is dedicated to a more detailed introduction to these issues by overviewing some related work in the field.

Chapter 3

Overview of Related Research

”I just read books! We read everything that’s published in the world ...
I look for leaks, I look for new ideas ... We read adventures and
novels and journals. I... I... Who’d invent a job like that?”
– JOE TURNER (THREE DAYS OF THE CONDOR)

The work we present in this dissertation falls into the research domain of cooperation of network security mechanisms over distributed environments, including distribution and analysis of security policies, exchange of information, correlation of the audit data, and protection of network security components. In this chapter, we survey work which has been previously done in those areas of research. We first examine on Section 3.1 existing approaches to get a distributed network security policy composed of multiple network security components, free of anomalies and misconfiguration. We then deal in Section 3.2 with existing work for the collaboration and exchanging of information between different network security components. We give in Section 3.3 an outlook on related work on the area of correlation of information reported by collaborative network security components. We finally conclude this chapter in Section 3.4 by overviewing two main approaches for protecting security resources on modern operating systems.

3.1 Deployment of Components Free of Anomalies

As we pointed out in the previous chapter, once the security officer of a network system has performed the analysis of threats and has specified a network security policy, he may implement such a policy by means of security mechanisms. Generally, this deployment consists in distributing the security rules expressed in this policy over different network security components of the system – such as firewalls, intrusion detection systems (IDSs), proxies, etc. – both at application, system, and network level. This implies cohesion of the security functions supplied by these components. In other words, security rules deployed over the different components must be consistent, not redundant and, as far as possible, optimal.

An approach based on a formal security policy refinement mechanism (using for instance abstract machines grounded on set theory and first order logic) ensures cohesion, completeness and optimization as built-in properties [Cuppens et al., 2004]. Unfortunately, in most cases, such an approach has not a wide follow and the policy is more often than not empirically deployed based on security administrator expertise and flair. It is then advisable to analyze the security rules deployed to detect, i.e., verify, and correct some policy anomalies – often referred in the literature as *intra- and inter-configuration anomalies* [Hamed and Al-Shaer, 2006]. These anomalies might be the origin of security holes and/or heaviness of intrusion prevention and detection processes. Firewalls and network intrusion detection systems (NIDSs) are the most commonly used network security components and, in our work, we focus particularly on their security rules.

As we overviewed in Chapter 2, firewalls are prevention-based devices ensuring access control at network level. They manage the traffic between the public network and the private network zones on one hand and between private zones in the local network in the other hand. The undesirable traffic is blocked or deviated by such a component. NIDSs are detection devices ensuring a monitoring role. They are components that supervise the traffic and generate alerts in the case of suspicious traffic. The attributes used to block or to generate alerts are almost the same.

Main research on firewalls and NIDSs is frequently focused on their design [Guttman, 1997, Bartal et al., 1999, Gouda and Liu, 2004], their analysis mechanisms [Frantzen et al., 2001, Kamara et al., 2003], and their packet classification mechanisms [Srinivasan et al., 1999, Eppstein and Muthukrishnan, 2001]. The challenge, when these two kinds of components coexist in the security architecture of an computer network is then to avoid inter-configuration anomalies. Nevertheless, none of these approaches address the proper management of anomalies due to conflicts between multiple-component setups.

For our work, we define the security rules of both firewalls and NIDSs as filtering and alerting rules, respectively. In turn, both filtering and alerting rules are specific cases of a more general configuration rule, which typically defines a $decision$ (such as $deny$, $alert$, $accept$, or $pass$) that applies over a set of $condition$ attributes, such as ***protocol***, ***source***, ***destination***, ***classification***, etc. We define a general configuration rule as follows:

$$R_i : \{condition_i\} \rightarrow decision_i$$

where i is the relative position of the rule within the set of rules, $\{condition_i\}$ is the conjunctive set of condition attributes such that $\{condition_i\}$ equals $C_1 \wedge C_2 \wedge ... \wedge C_p$ – being p the number of condition attributes of the given rule – and $decision$ is a boolean value in $\{true, false\}$. Let us notice that the decision of a filtering rule will be $true$ whether it applies to a specific value related to ***deny*** the traffic it matches, and will $false$ whether it applies to a specific value related to ***accept*** the traffic it matches. Similarly, the decision of an alerting rule will be $true$ whether it applies to a specific value related to ***alert*** the traffic it matches, and will be $false$ whether it applies a value related to ***pass*** the traffic.

We presented in [Cuppens et al., 2005a, Cuppens et al., 2005b] an audit process to manage intra-firewall policy anomalies, in order to detect and remove anomalies within a given firewall. This audit process is based on the existence of relationships between the condition attributes of the filtering rules, such as coincidence, disjunction, and inclusion, and proposes a transformation process which derives from an initial set of rules – with potential policy anomalies – to an equivalent one which is completely free of errors. The resulting rules are moreover completely disjoint, i.e., the ordering of rules is no longer relevant.

In [García et al., 2006f, García et al., 2006d], we extended our proposal to a distributed setup where both firewalls and NIDSs are in charge of the network security policy. This way, assuming that the role of both prevention and detection of network attacks is assigned to several components, our objective is to avoid intra and inter-component anomalies between filtering and alerting rules . The proposed approach is based on the similarity between the parameters of a filtering rule and those of an alerting rule. This way, we can check whether there are errors in those configurations regarding the policy deployment over each component which matches the same traffic. We refer the reader to Chapter 4 for a more detailed description of our approach.

Some other proposals, such as [Adiseshu et al., 2000, Gupta, 2000, Gouda and Liu, 2004, Al-Shaer et al., 2005], also provide means to directly manage the discovery of anomalies from the components' configuration. For instance, the authors in [Adiseshu et al., 2000] consider that, in a configuration set, two rules are in conflict when the first rule in order matches some packets that match the second rule, and the second rule also matches some of the packets that match the first rule. This approach is very limited since it just detects a particular case of wrongly defined rules in a single firewall configuration, i.e., just ambiguity within a intra-firewall configurations could be detected. It does not provide, furthermore, detection on more complex scenarios, i.e., inter-firewall configurations, where more than one component is intended to perform network access control.

In [Gupta, 2000], two cases of anomalies are considered. First, a rule R_j is defined as backward redundant iff there exists another rule R_i with higher priority in order such that all the packets that match rule R_j also match rule R_i. Second, a rule R_i is defined as forward redundant iff there exists another rule R_j with the same decision and less priority in order such that the following conditions hold: (1) all the packets that match R_i also match R_j; (2) for each rule R_k between R_i and R_j, and that matches all the packets that also match rule R_i, R_k has the same decision as R_i.

Although this approach seems to head in the right direction, we consider it as incomplete, since it does not detect all the possible cases of intra-component anomalies (as the ones defined in our work). For instance, given the set of rules shown in Figure 3.1(a), since R_2 comes after R_1, rule R_2 only applies over the interval $[51, 70]$ – i.e., R_2 is redundant to R_3, since, if we remove this rule from the configuration, the same policy is applied by rule R_3. The detection proposal, as defined in [Gupta, 2000], cannot detect the redundancy of rule R_2 within the configuration of such a given firewall. Furthermore, neither [Gupta, 2000] nor [Gouda and Liu, 2004] provide detection on multiple-component configurations.

$$\begin{array}{l} R_1 : s \in [10, 50] \rightarrow true \\ R_2 : s \in [40, 70] \rightarrow false \\ R_3 : s \in [50, 80] \rightarrow false \end{array} \qquad \begin{array}{l} R_1 : s \in [10, 50] \rightarrow false \\ R_2 : s \in [40, 90] \rightarrow false \\ R_3 : s \in [30, 80] \rightarrow true \end{array}$$

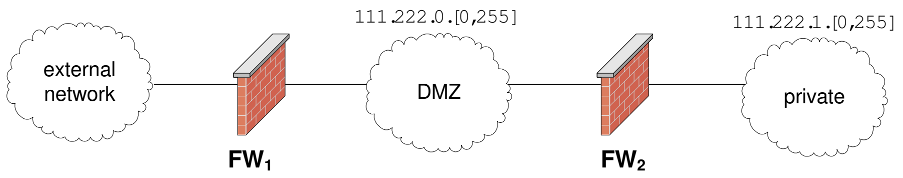
(a) Set of rules A (b) Set of rules B

Figure 3.1: Example of two firewall configurations.

To our best knowledge, the approach presented in [Al-Shaer et al., 2005] propose the most efficient set of techniques and algorithms to detect policy anomalies in both single and multi-firewall configuration setups. In addition to the discovery process, their approach also attempts an optimal insertion of arbitrary rules into an existing configuration, through a tree based representation of the filtering criteria. Nonetheless, and even though the efficiency of their proposed discovering algorithms and techniques is very promising, we also consider this approach as incomplete.

On the one hand, their intra- and inter-component discovery approach is not complete since, given a single- or multiple-component security policy, their detection algorithms are based on the analysis of relationships between rules two by two. This way, errors due to the union of rules are not explicitly considered (as our approach does). For example, the set of rules shown in Figure 3.1(b), may lead their discovery algorithms to inappropriate decisions (since the approach defined in [Al-Shaer and Hamed, 2004] cannot detect that rule R_3 will be never applied due to the union of rules R_1 and R_2). Though in [Al-Shaer et al., 2005] the authors pointed out to this problematic, claiming that they break down the initial set of rules into an equivalent set of rules free of overlaps between rules, no specific algorithms have been provided for solving it in [Al-Shaer and Hamed, 2004, Al-Shaer et al., 2005, Hamed and Al-Shaer, 2006].

On the other hand, their inter-component discovery approach considers as anomalies some situations that, from our point of view, must be suited to avoid inconsistent decisions between components used in the same policy to control or survey to different zones. For instance, given the scenario shown in Figure 3.2 their algorithms will inappropriately report a redundancy anomaly between filtering rules $FW_1\{R_1\}$ and $FW_2\{R_1\}$. This is because rule $FW_1\{R_1\}$ matches every packet that also $FW_2\{R_1\}$ does. As a consequence, [Al-Shaer and Hamed, 2004] considers rule $FW_2\{R_1\}$ as redundant since packets denied by this rule are already denied by rule $FW_1\{R_1\}$. However, this conclusion is not appropriate because rule $FW_1\{R_1\}$ applies to packets from the external zone to the private zone whereas rule $FW_2\{R_1\}$ applies to packets from the DMZ zone to the private zone. So, rule $FW_2\{R_1\}$ is useful and cannot be removed.



$FW_1\{R_1\} : p = tcp \wedge s \in any \wedge d \in 111.222.1.0/24 \wedge dport = 80 \rightarrow deny$

$FW_2\{R_1\} : p = tcp \wedge s \in 111.222.0.0/24 \wedge d \in 111.222.1.0/24 \wedge dport = 80 \rightarrow deny$

Figure 3.2: Example of a distributed control access scenario.

Though in [Al-Shaer and Hamed, 2004, Al-Shaer et al., 2005] the authors claim that their analysis technique marks every rule that is used on a network path, no specific algorithms have been provided for doing so. The net advantage of our approach over their approach is that it includes a model of the traffic which flows through each component. We consider this is necessary to draw the right conclusion in this case. Furthermore, although in [Al-Shaer et al., 2005] the authors consider their work as sufficiently general to be used for verifying many other filtering based security policies such as intrusion detection and prevention systems, no specific mechanisms have been provided for doing so.

3.2 Exchange of Audit Data between Components

Once verified and deployed a distributed network security policy, one may consider the exchange of information between the set of components implemented in such a policy. Traditional client/server solutions can be used in order to deploy multiple sensors at each host of the protected network. Thus, those sensors can locally collect audit data and forward it to a central point where it can be further analyzed. Early intrusion detection systems, such as DIDS [Snapp et al., 1991], and STAT [Ilgun et al., 1995], use this approach to process their data in a central node.

DIDS (Distributed Intrusion Detection System), for instance, is one of the earliest systems referred in the literature for using this approach of monitoring [Snapp et al., 1991]. The main components of DIDS are a central analyzer component (called DIDS director), a set of host-based sensors installed on each monitored host within the protected network, and a set of network-based sensors installed on each broadcasting segment of the target system. The communication infrastructure between the central analyzer and the distributed sensors is bidirectional. This way, although the sensors are most of the time sending asynchronously their reports to the central analyzer, it is also possible that the director directly requests them for more details.

NetSTAT [Vigna and Kemmerer, 1998, Vigna and Kemmerer, 1999], on the other hand, is an application of STAT (State Transition Analysis Technique) [Ilgun et al., 1995] to network-based detection. Indeed, in turn, it is the evolution of NSTAT [Kemmerer, 1997]. Based on the attack scenarios and the network fact modeled as a hyper-graph, NetSTAT automatically chooses places to probe network activities and applies an analysis of state transitions. This way, it is possible to decide what information it is necessary to collect within the protected network. Similarly to DIDS, and although NetSTAT collects network events in a distributed way, it analyzes them in a centralized fashion.

The main limitation of both DIDS and NetSTAT is that their exchange of audit data can quickly become a bottleneck – due to saturation problems associated with the service offered by their centralized analyzers. Their monitoring schemes are straightforward as they simply push the data to a central node and perform the computation there. Both approaches try to reduce the audit data before to send it to the central analysis unit – they try to select interesting parts of the audit stream and compress it. Unfortunately, an efficient data reduction scheme capable of forwarding only relevant data for arbitrary threat scenarios is very difficult to realize when too many sensors are deployed. Hence, when too many sensors are deployed, the central host is simply overloaded. Furthermore, the use of a single analyzer also induces a fault tolerance problem. If such a single analyzer crashes or becomes the victim of a denial of service (DoS) attack, the whole system is completely blinded.

To solve these disadvantages, some other results like GrIDS [Staniford-Chen et al., 1996], EMERALD [Porras and Neumann, 1997], and AAfID [Spafford and Zamboni, 2000], propose the use of layered structures where data is locally pre-processed and filtered, and further analyzed by intermediate components in a hierarchical fashion. The computational and network load is distributed over multiple analyzers and managers, distributed over different domains to analyze. The analyzers and managers of each domain perform their detection for just a small section of the whole network. Then, they forward the processed information to the entity which is on the top of the hierarchy – i.e., a master node – which finally analyzes all the reported incidents of the system.

GrIDS (Graph-based Intrusion Detection System for large networks) is an evolution of DIDS [Snapp et al., 1991] that aims at large distributed systems. It performs detection of distributed scans and worms by aggregating computer and network information into activity graphs [Staniford-Chen et al., 1996]. In contrast to the centralized approach of DIDS, GrIDS allows the construction of activity graphs that only represent hosts and the network activity between them. Each node of the graph represents a single host or a group of nodes, and the edges represent network traffic between nodes. The audit data of GrIDS is collected by means of both host- and network-based sensors, and then forwarded to the graph manager, which further feeds the collected information into the graph. The whole system deploys several graphs and several graph managers in a hierarchical fashion, in order to increase the scalability of the whole system. Therefore, each manager controls just a subset on the whole graph. Unfortunately, little details were provided regarding the communication infrastructure for the exchange of information between components.

Similarly, EMERALD (Event Monitoring Enabling Responses to Anomalous Live Disturbances) extends the work of IDES (Intrusion Detection Expert System) [Lunt et al., 1990] and NIDES (Next-Generation Intrusion Detection Expert System) [Anderson et al., 1995a] by implementing a recursive framework in which generic building blocks can be deployed in a hierarchical fashion [Porras and Neumann, 1997]. It combines host- and network-based sensors, as well as anomaly- and misuse-based analyzers. EMERALD is focused on the protection of large-scale enterprise networks that, in turn, are divided into independent domains – each one of them with their own security policy. Unfortunately, and although the authors were claiming to a very efficient communication infrastructure for the exchanging of information between components, few details were provided regarding the implementation and performance of such an infrastructure.

AAfID (Architecture for Intrusion Detection using Autonomous Agents), on the other hand, also presents a hierarchical approach to combat the limitations of centralized proposals and, particularly, to resist to denial of service attacks [Spafford and Zamboni, 2000]. It consists of four main components (called agents, filters, transceivers, and monitors) organized in a tree structure, where child and parent components communicate with each other. Regarding the communication subsystem of AAfID, it exhibits a very simplistic design and does not seem to be resistant to a denial of service attack. Furthermore, and although the set of agents may communicate with each other to agree upon a common suspicion level at every host, all the relevant data is simply forwarded to monitors (via transceivers) and require for human interaction in order to detect distributed intrusions.

To our best knowledge, and although those hierarchical approaches may mitigate some weaknesses present in centralized schemes, they still cannot avoid bottlenecks, scalability problems, and fault tolerance issues due to vulnerabilities at the root level. First, and although the analyzers of those hierarchical variants attempt to pre-filter the information within small domains, the massive amount of audit data forwarded to the higher level components is very hard to manage even at those layers. Second, if the root domain component crashes or becomes unavailable, the detection process will not properly be concluded. In order to solve these difficulties with both central and hierarchical data analysis, a decentralized scheme free of dedicated processing nodes is necessary.

Alternative approaches such as Micael [Queiroz et al., 1999], IDA [Asaka et al., 1999], Sparta [Kruegel and Toth, 2002] and MAIDS [Helmer et al., 2002], propose the use of mobile agent technology to gather the pieces of evidence of an attack (which are scattered over arbitrary locations). The authors of those approaches justify the use of mobile agent technology by the usual reasons of *overcoming network latency, reducing network load or allowing autonomous and asynchronous execution*. While these reasons are perfectly valid, in most of those approaches the use of agent technology and mobility is unnecessary and counterproductive. Mobile agents are used in those designs mainly as data containers (a task that can be performed more efficiently by using a simple message passing). They introduce, moreover, additional security risks and cause a performance penalty without providing any clear advantage. Furthermore, none of the proposals seems to have a definitive implementation or any industrial application.

In contrast to those centralized, hierarchical, and mobile agent-based proposals, we presented in [García et al., 2005e, García et al., 2005a] a decentralized message passing design which tries to eliminate the limitations and disadvantages studied and overviewed above. Our message passing design proposes an exchange of audit information across multiple nodes of a cooperative network through the use of a publish/subscribe model. We refer the reader to Chapter 5 for more information about such a work.

3.3 Merging and Correlation of Audit Information

The rise of cooperative frameworks to implement a distributed security policy leads to the reasoning on audit information held by multiple security components. The merging and correlation of this information allows us to fulfill different goals, such as removal of redundancy and scenario detection. The managing of this process has been extensively discussed in recent literature, such as [Valdes and Skinner, 2001, Debar and Wespi, 2001, Julisch, 2002, Cuppens and Miège, 2002, Ning et al., 2002]. Nevertheless, the goals aimed by those proposals are different and need to be explained.

For the discussion in this section, we assume that the process of merging and correlating audit information receives as input a stream of alerts from different network security components (such as NIDSs, firewalls, and so on). Despite the differences between the proposals studied in the literature, we can identify the whole process as a set of phases that transform the audit information (i.e., alerts) into a more complete view of occurring or attempted attack scenarios. The main objective is to produce, at the end of the entire process, intrusion reports that should advise the security officer about possible counter-measures – in order to react to the detected activity. In Figure 3.3 we give a graphical representation of such a process. The different steps shown in the figure, and the operations performed within each phase, are explained in the following sections.

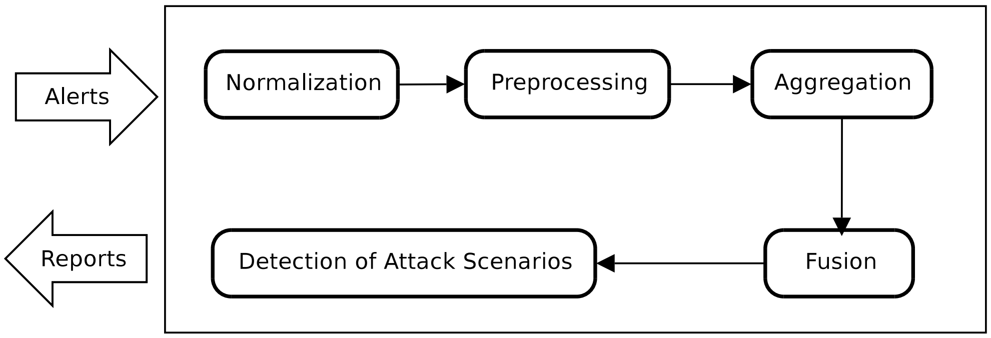


Figure 3.3: Merging and Correlation process overview.

Normalization and Preprocessing of Alerts

An alert is an abstraction of an event that can refer to one or more unauthorized actions and which removes the irrelevant details of those actions. It is often defined in the literature as a list of pairs of attributes with their corresponding value sets, where each attribute describes a certain property (or feature) of the action that this alert refers to (e.g., classification of a given attack). Each attribute has a type (e.g., string or integer) and a set of values associated with it. The value set can be empty when the attribute does not apply to the action (e.g., in case of attributes specifying network-level properties for a host-based activity) or when no information has been supplied by the component that generated the alert.

Before to combine the set of alerts reported from different security components, it is necessary to *normalize* such alerts – possibly encoded in different formats – into a standardized format. This normalization process may guarantee that the syntax and semantics of the resulting alerts is understood by the components involved in the correlation process. In order to do so, some specifications have been proposed by the intrusion detection community. The Common Intrusion Specification Language (CISL), for instance, was proposed to allow the components of the Common Intrusion Detection Framework (CIDF) to exchange data in semantically well-defined ways [Feiertag et al., 1999].

Although the objective of CISL's authors was to standardize their work, it was not well received by the industry community – probably due to the potential complexity of its procedures and expressions. But, some of the concepts and ideas proposed by the CIDF community motivated the creation of the IETF's Intrusion Detection Exchange Format Working Group (IDWG) to accomplish an industry standard. Their main goal is the Intrusion Detection Message Exchange Format (IDMEF) [Debar et al., 2006], which provides a standard representation for the exchange of alerts between different security components.

Once normalized the stream of alerts – into the IDMEF format, for example – one may perform a preprocessing of those resulting alerts. This preprocessing must guarantee that the complete set of attributes necessary to compare alerts (e.g., classification and name of the activity, timestamps, source and target, and so on) are included in those alerts.

Aggregation and Fusion of Alerts

After the normalization and preprocessing phases, the stream of alerts collected by the set of security components is in a suitable manner and a new process to merge similar alerts is performed – i.e., to reduce information before to start the correlation process.

This new process is often split in two stages (aggregation and fusion). The task of the first phase is the clustering of alerts that belong to the same activity occurrence – but detected by different components. To do so, the aggregation process performs an analysis of similarity between the attributes of each alert to compare, for example, timestamps, source and target

IP address, etc. Once compared the necessary set of attributes of two alerts, the similarity value of each comparison is generally computed to finally decide whether the two alerts are related or not to the same activity. If so, these two alerts are grouped in the same cluster, and then merged into a new alert, often referred in the literature as *meta-alert*, during the alert fusion process. A meta-alert is similar to a normal alert but its attributes are derived from the attributes of the merged alerts. Each meta-alert also contains references to all those the alerts that were merged to produce the meta-alert during the alert fusion process. Furthermore, a meta-alert can be then merged with other alerts or meta-alerts in a hierarchical manner, where the most recent meta-alert is the root node in the hierarchy, and all successor nodes can be accessed by following the references to the merged alerts. We refer the reader to [Kruegel et al., 2005] for further information.

The deployment of the complete process of merging differs among the studied literature. In [Valdes and Skinner, 2001], for example, a probabilistic similarity function is defined for each attribute. The authors then propose how to obtain an overall similarity value by combining similarity functions and using a probabilistic expectation of similarity to perform the fusion of alerts. A different approach is presented in [Debar and Wespi, 2001], where similar alerts are previously associated through the use of *a priori* definitions. A clustering method is presented in [Julisch, 2002], where an off-line process determines the causes for what a set of alerts must be fused. Finally, the use of boolean predicates is proposed by both [Cuppens, 2001] and [Ning et al., 2002], where expert rules are defined to determine whether two alerts are similar and may be aggregated and fused.

Detection of Attack Scenarios

During this phase, a correlation process is performed in order to link those alerts or meta-alerts that refer to unauthorized actions (or attacks) launched by an attacker against a given target. Several approaches have been proposed in order to represent these unauthorized actions. Among them are languages based on transition of states such as STAT [Ilgun, 1993]; colored petri nets such as CPA [Kumar and Spafford, 1994]; rule-based languages such as RUSSEL [Mounji et al., 1995] and P-BEST [Lindqvist and Porras, 1999]; languages based on finite state machines such as JFSM [Wu et al., 1999]. The use of formalisms for modeling dynamic systems, such as Chronicles, have also been proposed to perform alert correlation in [Morin et al., 2002]. The representation of unauthorized actions proposed in [Cuppens and Miège, 2002] is based on LAMBDA [Cuppens and Ortalo, 2000], an attack description language based on logic, and whose scenarios steps represent the attacker's actions (see Chapter 6 for a more detailed description of this language).

Early approaches directly perform the correlation stage at the merging process. The probabilistic approach presented in [Valdes and Skinner, 2001], for example, tries to detect the attack scenario as soon as they derive the similarity between rules – computed during the merging of alerts. Similarly, in [Debar and Wespi, 2001] the correlation process is also performed during the aggregation phase, where the different alerts are getting linked by means of their concept of duplicity of alerts and consequences.

Alternative approaches present a more defined separation of the merging and correlation of alerts. In [Cuppens and Miège, 2002], for example, the use of correlation rules declared by means of boolean predicates, is proposed. They first specify logic links between the influence of an attack performed against the target, and the necessary conditions to perform this attack. Hence, the pre-conditions to perform an attack are compared to the post-conditions of previously detected alerts. If the result is positive, alerts are correlated. The result is a graph representing the attack scenario (see Chapter 6 for a more detailed description of this approach). In [Ning et al., 2002], on the other hand, the authors present a similar proposal, also based on boolean predicates, and where those predicates are also used to represent prerequisites and consequences of actions that may point to an attack scenario.

Reports and Response Mechanisms

The output of the correlation process provides a set of attack scenarios that should advise the security officer about the intruder's activity. Nevertheless, just detecting attack scenarios does not prevent the intruder from reaching his objective. An additional mechanism is often necessary to decide when to execute a counter-measure once the scenario has been partially observed.

Most of the studied approaches propose few response mechanisms in addition to common intrusion reports. There is only a small variety of response techniques and the decision criteria that are used to activate the response remains often simplistic. Security officers, moreover, generally distrust at using the most interesting responses such as automatic reconfiguration of the network security policy. The main reason against the use of more elaborated response mechanisms is mainly due to the lack of confidence in the capabilities of the detection systems to take the right decision. Security administrators may also fear of not controlling the consequences of the automation of those counter-measures. Hence, the objective of most responses consists in stopping an ongoing attack. More elaborate reactions that are effective to automatically correct the detected vulnerabilities, remain marginal.

In [Cuppens et al., 2006a] we presented an intrusion reaction approach based on a library that implements different types of counter-measures once an attack scenario is detected. This proposal is based on a logical representation of both unauthorized actions and counter-measures, and further extends the recognition process of the intruder's intentions presented in [Cuppens and Miège, 2002]. Hence, when an attack scenario is identified, it can anticipate on the objective that the intruder attempts to achieve and on the future attacks that the intruder will perform to achieve it. We refer the reader to Chapter 6 for further information of this proposal and its implementation.

3.4 Protection of Network Security Components

Current research in network security components, such as firewalls and intrusion detection systems (IDSs), is mainly focused on improving classification, processing, detection, and reaction mechanisms, without taking into consideration their own security. The protection of these components is a serious and important problem which must be solved. Otherwise, if a remote adversary manages to compromise the security of these components, he may obtain the control of the system itself. These components, as opposite to other network elements, are almost always working with special privileges in order to execute their tasks. This fact may lead remote attackers to acquire these privileges in an unauthorized manner.

For instance, the existence of programming errors within its internal code, the manipulation of their resources (such as processes, configuration files, log files, and so on) in an unappropriated manner, or the increase of user privileges by looking for errors at operating system level, are some examples in which a remote adversary can bypass traditional security policy controls and get the control of a security component.

There are two main approaches to safely execute processes with special privileges on modern operating systems. A first approach is to apply a kernel-based access control to the outcoming system calls. A second approach is the creation of restricted environments, in which the processes will be executed and controlled outside the trusted system space.

Regarding the first approach, we presented in [García et al., 2005b, García et al., 2006b] a kernel based access control method which intercepts and cancels forbidden system calls launched by a remote attacker. This way, even if the attacker gains administration permissions, he will not achieve his purpose. To solve the administration constraints of our approach, we use a smart-card based authentication mechanism for ensuring the administrator's identity. Through the use of a cryptographic protocol, the protection mechanism verifies administrator's actions before holding him the indispensable privileges to manipulate a component. Otherwise, the access control enforcement will come to its normal operation. We refer the reader to Chapter 7 for more information regarding our proposal and implementation.

The proposals closest to ours are the protection mechanisms presented in [Ott, 2002] and [Loscocco and Smalley, 2001] for the creation of enhanced access control mechanisms integrated in the kernel of the GNU/Linux operating system. The main goal behind these two proposals is to reinforce the complete system by controlling the system calls and ensuring which process or user does the system call and against what it will be done. The ability to control the access to the resources allows to protect the security components and to avoid that nobody (including an attacker with administrator privileges) can disable them. Nevertheless, both approaches differ from ours in a number of ways. First, and to our best knowledge, neither [Loscocco and Smalley, 2001] nor [Ott, 2002] do not address the management of administration constraints, as our proposal does through the two-factor authentication mechanism we present in Chapter 7, Section 7.3. Second, our approach, entirely based on the *Linux Security Modules* (LSM) framework [Wright et al., 2002], guarantees the compatibility with previous applications and kernel modules without the necessity of modifications. However, both [Loscocco and Smalley, 2001] and [Ott, 2002] require the rewriting of some features of the original Linux kernel to properly work. This situation may force to recompile existing code and/or modules in order to obtain the new security features. Although it exists a LSM-based prototype for the approach presented in [Loscocco and Smalley, 2001], it does not seem to be actively maintained for the current Linux-2.6 kernel series.

Regarding the second approach, we find in [Hope, 2002] a protection mechanism for the creation of restricted environments within Unix setups. The authors in [Hope, 2002] present the use of a special system call to restrict the access to a specific area of the file system. This specific area is intended just for the processes that are executed under each restricted environment. Then, this system call properly changes the root directory to the given path. This way, the process remains in a safe space from where it is not possible to escape – even if the component is compromised, the whole system will remain safe since the illicit activities are caught within the replicated file system.

This proposal requires, however, a replicated file system tree for each environment. Hence, the administrator in charge of the system must reproduce the original file system tree to include, for example, shared libraries or configuration files, and copy them to the new path. Other disadvantage of this proposal is that it does not guarantee the correct execution flow of a process, i.e., the behavior of a process can be modified by using, for example, a buffer overflow. Hence, the attacker can overwrite the configuration or logs files of such a process by simply using an arbitrary code execution attack – since these files remain in the same environment of the protected security component process.

Extended versions of the previous model, such as [Herzog and Shahmehri, 2002], may also offer support for access control to resources and guarantee the integrity of the security component's resources. Nonetheless, these extended proposals do not protect from vulnerabilities placed outside the trusted environment. A simple bug in a privileged service, or even the use of stolen passwords, may lead the attacker from the external environment to attack the component and its resources.

Chapter 4

Management of Anomalies on Distributed Network Security Policies

"I know I've made some very poor decisions recently, but I can give you my complete assurance that my work will be back to normal."
– HAL (2001: A SPACE ODYSSEY)

The use of *firewalls* and *network intrusion detection systems* (NIDSs) is the dominant method to survey and guarantee the security policy in current computer networks. Firewalls are traditional security components which provide means to filter traffic within computer networks, as well as to police the incoming and outcoming interaction with the Internet. On the other hand, NIDSs are complementary components used to enhance the visibility level of the system as a whole, pointing out to malicious traffic. To deploy the configuration of both firewalls and NIDSs, it is necessary to translate the rules of the network security policy into a set of filtering and alerting rules. The existence of anomalies between those rules in distributed multi-component scenarios, is very likely to degrade the network security policy. The discovering and removal of these anomalies is a serious and complex problem to solve. In this chapter, we present a set of algorithms for such a management.

The remaining of this chapter is organized as follows. Section 4.1 starts by introducing a network model that is further used in Section 4.2 and Section 4.3 when presenting, respectively, our intra and inter-component anomaly's classifications and algorithms. Section 4.4 overviews a first implementation of our algorithms in order to validate its performance over real multi-component scenarios.

4.1 Network Model and Topology Properties

The purpose of our network model is to determine which components within the network are crossed by a given packet, knowing its source and destination. It is defined as follows. First, and concerning the traffic flowing from two different zones of the distributed policy scenario, we may determine the set of components that are crossed by this flow. Regarding the scenario shown in Figure 4.1, for example, the set of components crossed by the network traffic flowing from zone $external\ network$ to zone $private_3$ equals [C_1,C_2,C_4], and the set of components crossed by the network traffic flowing from zone $private_3$ to zone $private_2$ equals [C_4,C_2,C_3].

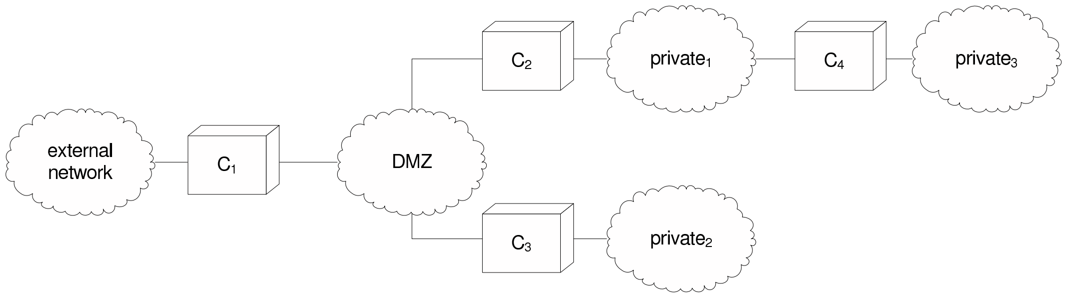


Figure 4.1: Simple distributed policy setup.

Let C be a set of components and let Z be a set of zones. We assume that each pair of zones in Z are mutually disjoint, i.e., if $z_i \in Z$ and $z_j \in Z$ then $z_i \cap z_j = \emptyset$. We then define the predicate $connected(c_1, c_2)$ as a symmetric and anti-reflexive function which becomes $true$ whether there exists, at least, one interface connecting component c_1 to component c_2. On the other hand, we define the predicate $adjacent(c, z)$ as a relation between components and zones which becomes $true$ whether the zone z is interfaced to component c. Referring to Figure 4.1, we can verify that predicates $connected(C_1, C_2)$ and $connected(C_1, C_3)$, as well as $adjacent(C_1, DMZ)$, $adjacent(C_2, private_1)$, $adjacent(C_3, DMZ)$, and so on, become $true$. We then define the set of paths, P, as follows. If $c \in C$ then $[c] \in P$ is an atomic path. Similarly, if $[p.c_1] \in P$ (be "." a concatenation functor) and $c_2 \in C$, such that $c_2 \notin p$ and $connected(c_1, c_2)$, then $[p.c_1.c_2] \in P$. This way, we can notice that, concerning Figure 4.1, $[C_1, C_2, C_4] \in P$ and $[C_1, C_3] \in P$.

Let us now define a set of functions related with the order between paths. We first define functions $first$, $last$, and the order functor between paths. We define function $first$ from P in C such that if p is a path, then $first(p)$ corresponds to the first component in the path. Conversely, we define function $last$ from P in C such that if p is a path, then $last(p)$ corresponds to the last component in the path. We then define the order functor between paths as $p_1 \leq p_2$, such that path p_1 is shorter than p_2, and where all the components within p_1 are also within p_2. We also define the predicates $isFirewall(c)$ and $isNIDS(c)$ which become $true$ whether the component c is, respectively, a firewall or a NIDS.

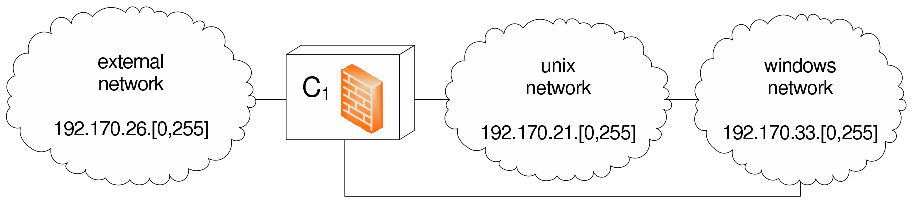
Two additional functions are $route$ and $minimal\_route$. We first define function $route$ from Z to Z in 2^P, such that $p \in route(z_1, z_2)$ iff the path p connects zone z_1 to zone z_2. Formally, we define that $p \in route(z_1, z_2)$ iff the predicates $adjacent(first(p), z_1)$ and $adjacent(last(p), z_2)$ become $true$. Similarly, we then define $minimal\_route$ from Z to Z in 2^P, such that $p \in minimal\_route(z_1, z_2)$ iff the following conditions hold: (1) $p \in route(z_1, z_2)$; (2) There does not exist $p' \in route(z_1, z_2)$ such that $p' < p$. Regarding Figure 4.1, we can verify that the $minimal\_route$ from zone $private_3$ to zone $private_2$ equals $[C_4, C_2, C_3]$, i.e., $minimal\_route(private_3, private_2) = \{[C_4, C_2, C_3]\}$.

Let us finally conclude this section by defining the predicate *affects*(Z, A_c) as a boolean expression which becomes $true$ whether there is, at least, an element $z \in Z$ such that the configuration of z is vulnerable to the attack category $A_c \in V$, where V is a vulnerability set built from a vulnerability database, such as CVE/CAN [MITRE Corporation, 2005] or OSVDB [Open Security Foundation, 2005].

4.2 Intra-component Classification and Algorithms

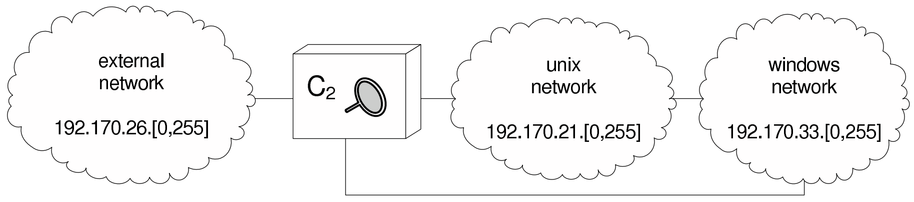
In this section we present our set of intra-component audit algorithms, whose main objective is the complete discovering and removal of policy anomalies that could exist in a single component policy, i.e., to discover and warn the security officer about potential anomalies within the configuration rules of a given component.

Let us start by classifying the complete set of anomalies that can occur within a single component configuration. An example for each anomaly will be illustrated through the sample scenario shown in Figure 4.2.



$C_1\{R_1\}$: {tcp, 192.170.26.[10,20]:any, 192.170.26.[50,60]:any} → false
$C_1\{R_2\}$: {tcp, 192.170.26.[0,255]:any, 192.170.33.[0,255]:any} → false
$C_1\{R_3\}$: {tcp, 192.170.21.[1,30]:any, 192.170.26.[20,45]:any} → true
$C_1\{R_4\}$: {tcp, 192.170.21.[20,60]:any, 192.170.26.[25,35]:any} → false
$C_1\{R_5\}$: {tcp, 192.170.21.[30,70]:any, 192.170.26.[20,45]:any} → false
$C_1\{R_6\}$: {tcp, 192.170.21.[15,45]:any, 192.170.26.[25,30]:any} → true

(a) Example scenario of a filtering policy.



$C_2\{R_1\}$: {tcp, 192.170.26.[0,255]:any, 192.170.33.[0,255]:any, $payload_1$, winworm} → true
$C_2\{R_2\}$: {tcp, 192.170.26.[0,255]:any, 192.170.21.[0,255]:any, $payload_2$, winworm} → true
$C_2\{R_3\}$: {tcp, 192.170.33.[0,255]:any, 192.170.21.[0,255]:any, $payload_3$, unixworm} → true
$C_2\{R_4\}$: {tcp, 192.170.26.[1,30]:any, 192.170.21.[20,45]:any, $payload_4$, unixworm} → true
$C_2\{R_5\}$: {tcp, 192.170.26.[20,60]:any, 192.170.21.[25,35]:any, $payload_5$, unixworm} → true
$C_2\{R_6\}$: {tcp, 192.170.26.[10,40]:any, 192.170.21.[25,30]:any, $payload_6$, unixworm} → true

(b) Example scenario of an alerting policy.

Figure 4.2: Example of filtering and alerting policies.

Intra-Component Shadowing A configuration rule R_i is shadowed in a set of configuration rules R whether such a rule never applies because all the packets that R_i may match, are previously matched by another rule, or combination of rules, with higher priority. Regarding Figure 4.2, rule $C_1\{R_6\}$ is shadowed by the overlapping of rules $C_1\{R_3\}$ and $C_1\{R_5\}$.

Intra-Component Redundancy A configuration rule R_i is redundant in a set of configuration rules R whether the following conditions hold: (1) R_i is not shadowed by any other rule or set of rules; (2) when removing R_i from R, the security policy does not change. For instance, referring to Figure 4.2, rule $C_1\{R_4\}$ is redundant, since the overlapping between rules $C_1\{R_3\}$ and $C_1\{R_5\}$ is equivalent to the police of rule $C_1\{R_4\}$.

Intra-Component Irrelevance A configuration rule R_i is irrelevant in a set of configuration rules R if one of the following conditions holds:

(1) Both source and destination address are within the same zone. For instance, rule $C_1\{R_1\}$ is irrelevant since the source of this address, $external\ network$, as well as its destination, is the same.

(2) The component is not within the minimal route that connects the source zone, concerning the irrelevant rule which causes the anomaly, to the destination zone. Hence, the rule is irrelevant since it matches traffic which does not flow through this component. Rule $C_1\{R_2\}$, for example, is irrelevant since component C_1 is not in the path which corresponds to the minimal route between the source zone $unix\ network$ to the destination zone $windows\ network$.

(3) The component is a NIDSs, i.e., the predicate $isNIDS(c)$ (cf. Section 4.1) becomes $true$, and, at least, one of the condition attributes in R_i is related with a classification of attack A_c which does not affect the destination zone of such a rule – i.e., the predicate affects(z_d, A_c) becomes $false$. Regarding Figure 4.2, we can see that rule $C_2\{R_2\}$ is irrelevant since the nodes in the destination zone $unix\ network$ are not affected by vulnerabilities classified as $winworm$.

Intra-Component Algorithms

Our proposed audit process is a way to alert the security officer in charge of the network about these configuration errors, as well as to remove all the useless rules in the initial firewall configuration. The data to be used for the detection process is the following. A set of rules R as a list of initial size n, where n equals $count(R)$, and where each element is an associative array with the strings $condition$, $decision$, $shadowing$, $redundancy$, and $irrelevance$ as keys to access each necessary value.

For reasons of clarity, we assume one can access a linked-list through the operator R_i, where i is the relative position regarding the initial list size – $count(R)$. We also assume one can add new values to the list as any other normal variable does ($element \leftarrow value$), as well as to remove elements through the addition of an empty set ($element \leftarrow \emptyset$). The internal order of elements from the linked-list R keeps with the relative ordering of rules.

Algorithm 1: `exclusion`(B,A)

```
C[condition] ← ∅;
C[shadowing] ← false;
C[redundancy] ← false;
C[irrelevance] ← false;
C[decision] ← B[decision];
forall the elements of A[condition] and B[condition] do
    if ((A_1 ∩ B_1) ≠ ∅ and (A_2 ∩ B_2) ≠ ∅
    and ... and (A_p ∩ B_p) ≠ ∅) then
        C[condition] ← C[condition] ∪
        {(B_1 − A_1) ∧ B_2 ∧ ... ∧ B_p,
        (A_1 ∩ B_1) ∧ (B_2 − A_2) ∧ ... ∧ B_p,
        (A_1 ∩ B_1) ∧ (A_2 ∩ B_2) ∧ (B_3 − A_3) ∧ ... ∧ B_p,
        ...
        (A_1 ∩ B_1) ∧ ... ∧ (A_{p−1} ∩ B_{p−1}) ∧ (B_p − A_p)};
    else
        C[condition] ← (C[condition] ∪ B[condition]);
return C;
```

Algorithm 2: `testIrrelevance`(c, r)

```
$z_s \leftarrow$ source ($r$);
$z_d \leftarrow$ dest ($r$);
if ($z_s = z_d$) and ($\neg r[decision]$) then
    warning ("$First\ case\ of\ irrelevance$");
else if $z_s \neq z_d$ then
    $p \leftarrow$ minimal_route ($z_s$,$z_d$);
    if $c \notin p$ and ($\neg r[decision]$) then
        warning ("$Second\ case\ of\ irrelevance$");
    else if ($\neg$empty ($r[A_c]$)) and ($\neg$affects$(z_d, r[A_c])$) then
        warning ("$Third\ case\ of\ irrelevance$");
    else return $false$;
return $true$;
```

Algorithm 3: `testRedundancy`(R,r)

```
$i \leftarrow 1$;
$temp \leftarrow r$;
while $\neg test$ and ($i \leq count(R)$) do
    $temp \leftarrow$ exclusion$(temp, R_i)$;
    if temp[condition] = $\emptyset$ then
        return $true$;
    $i \leftarrow (i + 1)$;
return $false$;
```

Algorithm 4: `intra-component-audit`(c, R)

```
begin
    $n \leftarrow count(R)$;
    /*Phase 1*/
    for $i \leftarrow 1$ to $(n-1)$ do
        for $j \leftarrow (i+1)$ to $n$ do
            if $R_i[decision] \neq R_j[decision]$ then
                $R_j \leftarrow$ exclusion ($R_j$,$R_i$);
                if $R_j[condition] = \emptyset$ then
                    warning (“$Shadowing$”);
                    $R_j[shadowing] \leftarrow true$;
    /*Phase 2*/
    for $i \leftarrow 1$ to $(n-1)$ do
        $R_a \leftarrow \{r_k \in R \mid n \geq k > i$ and
            $r_k[decision] = r_i[decision]\}$;
        if testRedundancy ($R_a$,$R_i$) then
            warning (“$Redundancy$”);
            $R_i[condition] \leftarrow \emptyset$;
            $R_i[redundancy] \leftarrow true$;
        else
            for $j \leftarrow (i+1)$ to $n$ do
                if $R_i$[decision]=$R_j$[decision] then
                    $R_j \leftarrow$exclusion ($R_j$,$R_i$);
                    if ($\neg R_j[redundancy]$ and
                    $R_j$[condition] = $\emptyset$) then
                        warning (“$Shadowing$”);
                        $R_j[shadowing] \leftarrow true$;
    /*Phase 3*/
    for $i \leftarrow 1$ to $n$ do
        if $R_i[condition] \neq \emptyset$ then
            if testIrrelevance (c,$R_i$) then
                $R_j[irrelevance] \leftarrow true$;
                $r[condition] \leftarrow \emptyset$;
end
```

Each element $R_i[condition]$ is a boolean expression over p possible attributes. To simplify, we only consider as attributes the following ones: $szone$ (source zone), $dzone$ (destination zone), $sport$ (source port), $dport$ (destination port), $protocol$, and $attack\_class$ – or A_c for short – which will be empty whether the component is a firewall. In turn, each element $R_i[decision]$ is a boolean variable whose values are in $\{true, false\}$. Finally, elements $R_i[shadowing]$, $R_i[redundancy]$, and $R_i[irrelevance]$ are boolean variables in $\{true, false\}$ – which will be initialized to $false$ by default.

We split the whole process in four different algorithms. The first algorithm (cf. Algorithm 1) is an auxiliary function whose input is two rules, A and B. Once executed, this auxiliary function returns a further rule, C, whose set of condition attributes is the exclusion of the set of conditions from A over B. In order to simplify the representation of this algorithm, we use the notation A_i as an abbreviation of the variable $A[condition][i]$, and the notation B_i as an abbreviation of the variable $B[condition][i]$ – where i in $[1, p]$.

The second algorithm (cf. Algorithm 2) is a boolean function in $\{true, false\}$ which applies the necessary verifications to decide whether a rule r is irrelevant for the configuration of a component c. To properly execute such an algorithm, let us define $source(r)$ as a function in Z such that $source(r) = szone$, and $dest(r)$ as a function in Z such that $dest(r) = dzone$.

The third algorithm (cf. Algorithm 3) is a boolean function in $\{true, false\}$ which, in turn, applies the transformation *exclusion* (Algorithm 1) over a set of configuration rules to check whether the rule obtained as a parameter is potentially redundant.

The last algorithm (cf. Algorithm 4) performs the whole process of detecting and removing the complete set of intra-component anomalies. This process is split in three different phases. During the first phase, a set of shadowing rules are detected and removed from a top-bottom scope, by iteratively applying Algorithm 1 – when the decision field of the two rules is different. Let us notice that this stage of detecting and removing shadowed rules is applied before the detection and removal of proper redundant and irrelevant rules.

The resulting set of rules is then used when applying the second phase, also from a top-bottom scope. This stage is performed to detect and remove proper redundant rules, through an iterative call to Algorithm 3 (i.e., *testRedundancy*), as well as to detect and remove all the further shadowed rules remaining during the latter process. Finally, during a third phase the whole set of non-empty rules is analyzed in order to detect and remove irrelevance, through an iterative call to Algorithm 2 (i.e., *testIrrelevance*).

Applying the Intra-Component Algorithms

In the following we give an outlook on applying our set of intra-component algorithms over some representative examples. Let us start applying the function *exclusion* (Algorithm 1) over a set of two rules R_i and R_j, each one of them with two condition attributes – $szone$ and $dzone$ – and where rule R_j has less priority than rule R_i. In this first example,

$$R_i[condition] = (szone \in [80, 100]) \wedge (dzone \in [1, 50])$$
$$R_j[condition] = (szone \in [1, 50]) \wedge (dzone \in [1, 50])$$

since $(szone \in [1, 50]) \cap (szone \in [80, 100])$ equals $\emptyset$, the condition attributes of rules R_i and R_j are completely independent. Thus, the applying of $exclusion(R_j, R_i)$ is equal to $R_j[condition]$.

The following three examples show the same execution over a set of condition attributes with different cases of conflict. A first case is the following,

$$R_i[condition] = (szone \in [1, 60]) \wedge (dzone \in [1, 30])$$
$$R_j[condition] = (szone \in [1, 50]) \wedge (dzone \in [1, 50])$$

where there is a main overlap of attribute $szone$ from $R_i[condition]$ which completely excludes the same attribute on $R_j[condition]$. Then, there is a second overlap of attribute $dzone$ from $R_i[condition]$ which partially excludes the range $[1, 30]$ into attribute $dzone$ of $R_j[condition]$, which becomes $dzone$ in $[31, 50]$. This way, $exclusion(R_j, R_i) \leftarrow \{(s \in [1, 50]) \wedge (dzone \in [31, 50])\}$. For reasons of clarity, we do not show the first empty set corresponding to the first overlap. If shown, the result should become as follows: $exclusion(R_j, R_i) \leftarrow \{\emptyset, (szone \in [1, 50]) \wedge (dzone \in [31, 50])\}$.

In this other example,

$$R_i[condition] = (szone \in [1,60]) \wedge (dzone \in [20,30])$$
$$R_j[condition] = (szone \in [1,50]) \wedge (dzone \in [1,50])$$

there are two simple overlaps of both attributes $szone$ and $dzone$ from $R_i[condition]$ to $R_j[condition]$, such that $exclusion(R_j, R_i)$ becomes $\{(szone \in [1,50]) \wedge (dzone \in [1,19]), (szone \in [1,50]) \wedge (dzone \in [31,50])\}$.

A more complete example is the following,

$$R_i[condition] = (szone \in [10,40]) \wedge (dzone \in [20,30])$$
$$R_j[condition] = (szone \in [1,50]) \wedge (dzone \in [1,50])$$

where $exclusion(R_j, R_i)$ becomes $\{(szone \in [1,9]) \wedge (dzone \in [1,50]), (szone \in [41,50]) \wedge (dzone \in [1,50]), (szone \in [10,40]) \wedge (dzone \in [1,19]), (szone \in [10,40]) \wedge (dzone \in [31,50])\}$.

Regarding a full exclusion, let us show the following example,

$$R_i[condition] = (szone \in [1,60]) \wedge (dzone \in [1,60])$$
$$R_j[condition] = (szone \in [1,50]) \wedge (dzone \in [1,50])$$

where the set of condition attributes of rule R_i completely excludes the ones of rule R_j. Then, the applying of $exclusion(R_j, R_i)$ becomes an empty set (i.e., $\{\emptyset, \emptyset\} = \emptyset$). Hence, on a further execution of Algorithm 4 the shadowing field of rule R_j (initialized as $false$ by default) would become $true$ (i.e., $R_j[shadowing] \leftarrow true$).

In order to show the execution of Algorithm 4 over a more complete set of rules, we give an outlook of such an execution over the following set of rules:

$$R_1 : szone \in [10,50] \rightarrow true$$
$$R_2 : szone \in [40,90] \rightarrow false$$
$$R_3 : szone \in [60,100] \rightarrow false$$
$$R_4 : szone \in [30,80] \rightarrow true$$
$$R_5 : szone \in [1,70] \rightarrow false$$

We start by showing the initial step within the first phase of Algorithm 4, where i equals 1, and applied over the previous set of filtering rules. Let us notice that on this first step, the execution of function *exclusion*, with rules R_2 and R_1, since their decision is different, becomes the range $[51, 90]$. Similarly, the execution of function *exclusion*, with rules R_5 and R_1 becomes the range $\{[1, 9], [51, 70]\}$. The result of this first step is the following:

$$\begin{aligned}
&R_1 : szone \in [10, 50] \rightarrow true\\
&R_2 : szone \in [51, 90] \rightarrow false\\
&R_3 : szone \in [60, 100] \rightarrow false\\
&R_4 : szone \in [30, 80] \rightarrow true\\
&R_5 : szone \in \{[1, 9], [51, 70]\} \rightarrow false
\end{aligned}$$

Let us now move to the second step, with i equals 2. In this step, the range of rule R_4 decreases since the execution of function *exclusion*, with rules R_2 and R_4, whose decision is different, becomes the range $[30, 50]$:

$$\begin{aligned}
&R_1 : szone \in [10, 50] \rightarrow true\\
&R_2 : szone \in [51, 90] \rightarrow false\\
&R_3 : szone \in [60, 100] \rightarrow false\\
&R_4 : szone \in [30, 50] \rightarrow true\\
&R_5 : szone \in \{[1, 9], [51, 70]\} \rightarrow false
\end{aligned}$$

At the end of the first phase, once executed both third and fourth steps, the resulting rules remain as above:

$$\begin{aligned}
&R_1 : szone \in [10, 50] \rightarrow true\\
&R_2 : szone \in [51, 90] \rightarrow false\\
&R_3 : szone \in [60, 100] \rightarrow false\\
&R_4 : szone \in [30, 50] \rightarrow true\\
&R_5 : szone \in \{[1, 9], [51, 70]\} \rightarrow false
\end{aligned}$$

Once finished the first phase and running over the first step of the second phase, i.e., i equals 1, we notice that: (1) the result of applying function *testRedundancy* with rule R_1

as parameter becomes $false$; (2) the execution of function *exclusion*, with rules R_4 and R_1, completely excludes the condition attribute of rule R_4. Hence, rule R_4, is reported as shadowed by the combination of rules R_1 and R_2, and its condition attribute becomes an empty set. Therefore, the status field $shadowing$ of rule R_4, i.e., $R_4[shadowing]$, switches its value to $true$:

$$\begin{aligned}
&R_1 : szone \in [10, 50] \rightarrow true\\
&R_2 : szone \in [51, 90] \rightarrow false\\
&R_3 : szone \in [60, 100] \rightarrow false\\
&R_4 : \emptyset \rightarrow true\\
&R_5 : szone \in \{[1, 9], [51, 70]\} \rightarrow false
\end{aligned}$$

Then, we follow now to the second step of the second phase, i.e., i equals 2, and we notice that rule R_2 disappears since the result of applying function *testRedundancy* with rule R_2 as parameter becomes $true$. Thus, the condition attribute of rule R_2 becomes an empty set, and its status field $redundancy$, i.e., $R_2[redundancy]$, switches its value to $true$:

$$\begin{aligned}
&R_1 : szone \in [10, 50] \rightarrow true\\
&R_2 : \emptyset \rightarrow false\\
&R_3 : szone \in [60, 100] \rightarrow false\\
&R_4 : \emptyset \rightarrow true\\
&R_5 : szone \in \{[1, 9], [51, 70]\} \rightarrow false
\end{aligned}$$

At the end of the following step, where i equals 3, we notice that the execution of function *testRedundancy* with rule R_3 as parameter becomes $false$. Thus, we apply function *exclusion*, with rules R_5 and R_3 as parameters. As a result of this execution, the second subrange of rule R_5 scarcely decreases from $[51, 70]$ to $[51, 59]$:

$$\begin{aligned}
&R_1 : szone \in [10, 50] \rightarrow true\\
&R_2 : \emptyset \rightarrow false\\
&R_3 : szone \in [60, 100] \rightarrow false\\
&R_4 : \emptyset \rightarrow true\\
&R_5 : szone \in \{[1, 9], [51, 59]\} \rightarrow false
\end{aligned}$$

We do not show the rest of the execution, since the resulting set of filtering rules does not modify from the previous one, which is the following:

$/*\, resulting\; rules *\,/$

$R_1 : szone \in [10, 50] \rightarrow true$
$R_3 : szone \in [60, 100] \rightarrow false$
$R_5 : szone \in \{[1, 9], [51, 59]\} \rightarrow false$

Let us recall that the following two warnings will notice the security officer to the discovering of both shadowing and redundancy anomalies, in order to verify the correctness of the whole detection and transformation process:

$/*\, warnings *\,/$

Shadowing on R_4 with R_2,R_1
Redundancy on R_2 with R_3,R_5

To conclude this section, let us finally show the warnings reported when executing Algorithm 4 over the configuration of the two components we showed in Figure 4.2.

$/*\, warnings *\,/$

First case of irrelevance on $C_1\{R_1\}$
Second case of irrelevance on $C_1\{R_2\}$
Redundancy on $C_1\{R_4\}$ with $C_1\{R_3\}$,$C_1\{R_5\}$
Shadowing on $C_1\{R_6\}$ with $C_1\{R_3\}$,$C_1\{R_5\}$
Third case of irrelevance on $C_2\{R_2\}$

Correctness of the Intra-Component Algorithms

Lemma 1 *Let $R_i : condition_i \rightarrow decision_i$ and $R_j : condition_j \rightarrow decision_j$ be two configuration rules. Then $\{R_i, R_j\}$ is equivalent to $\{R_i, R'_j\}$ where $R'_j \leftarrow exclusion(R_j, R_i)$.*

Proof of Lemma 1 Let us assume that:

$$R_i[condition] = A_1 \wedge A_2 \wedge ... \wedge A_p, \text{ and}$$
$$R_j[condition] = B_1 \wedge B_2 \wedge ... \wedge B_p.$$

If $(A_1 \cap B_1) = \emptyset$ or $(A_2 \cap B_2) = \emptyset$ or $\ldots$ or $(A_p \cap B_p) = \emptyset$ then $exclusion(R_j, R_i) \leftarrow R_j$. Hence, to prove the equivalence between $\{R_i, R_j\}$ and $\{R_i, R'_j\}$ is trivial in this case.

Let us now assume that:

$$(A_1 \cap B_1) \neq \emptyset \text{ and } (A_2 \cap B_2) \neq \emptyset$$
$$\text{and ... and } (A_p \cap B_p) \neq \emptyset.$$

If we apply rules $\{R_i, R_j\}$ where R_i comes before R_j, then rule R_j applies to a given packet if this packet satisfies $R_j[condition]$ but not $R_i[condition]$ (since R_i applies first). Therefore, notice that $R_j[condition] - R_i[condition]$ is equivalent to:

$$(B_1 - A_1) \wedge B_2 \wedge ... \wedge B_p \text{ or}$$
$$(A_1 \cap B_1) \wedge (B_2 - A_2) \wedge ... \wedge B_p \text{ or}$$
$$(A_1 \cap B_1) \wedge (A_2 \cap B_2) \wedge (B_3 - A_3) \wedge ... \wedge B_p \text{ or}$$
$$\ldots$$
$$(A_1 \cap B_1) \wedge ... \wedge (A_{p-1} \cap B_{p-1}) \wedge (B_p - A_p)$$

which corresponds to $R'_j = exclusion(R_j, R_i)$. This way, if R_j applies to a given packet in $\{R_i, R_j\}$, then rule R'_j also applies to this packet in $\{R_i, R'_j\}$. Conversely, if R'_j applies to a given packet in $\{R_i, R'_j\}$, then this means this packet satisfies $R_j[condition]$ but not $R_i[condition]$. So, it is clear that rule R_j also applies to this packet in $\{R_i, R_j\}$. Since in Algorithm 1 $R'_j[decision]$ becomes $R_j[decision]$, this enables to conclude that $\{R_i, R_j\}$ is equivalent to $\{R_i, R'_j\}$. $\square$

Theorem 2 *Let R be a set of configuration rules and let $Tr(R)$ be the resulting rules obtained by applying Algorithm 4 to R. Then R and $Tr(R)$ are equivalent.*

Proof of Theorem 2 Let $Tr'_1(R)$ be the set of rules obtained after applying the first phase of Algorithm 4.

Since $Tr'_1(R)$ is derived from rule R by applying $exclusion(R_j, R_i)$ to some rules R_j in R, it is straightforward, from Lemma 1, to conclude that $Tr'_1(R)$ is equivalent to R.

Let us now move to the second phase, and let us consider a rule R_i such that $testRedundan$-$cy(R_i)$ (cf. Algorithm 3) is $true$. This means that $R_i[condition]$ can be derived by conditions of a set of rules S with the same decision and that come after in order than rule R_i.

Since every rule R_j with a decision different from the one of rules in S has already been excluded from rules of S in the first phase of the Algorithm, we can conclude that rule R_i is definitely redundant and can be removed without changing the component configuration.

This way, we conclude that Algorithm 4 preserves equivalence in this case.

On the other hand, if $testRedundancy(R_i)$ is $false$, then transformation consists in applying function $exclusion(R_j, R_i)$ to some rules R_j which also preserves equivalence. Similarly, and once in the third phase, let us consider a rule R_i such that $testIrrelevance(c, R_i)$ is $true$.

This means that this rule matches traffic that will never cross component c, or that it is irrelevant for the component's configuration. So, we can remove R_i from R without changing such a configuration.

Thus, in this third case, as in the other two cases, $Tr'(R)$ is equivalent to $Tr'_1(R)$ which, in turn, is equivalent to R. □

Lemma 3 *Let R_i : $condition_i \rightarrow decision_i$ and R_j : $condition_j \rightarrow decision_j$ be two configuration rules. Then rules R_i and R'_j, where $R'_j \leftarrow exclusion(R_j,\ R_i)$ will never simultaneously apply to any given packet.*

Proof of Lemma 3 Notice that rule R'_j only applies when rule R_i does not apply. Thus, if rule R'_j comes before rule R_i, this will not change the final decision since rule R'_j only applies to packets that do not match rule R_i. □

Theorem 4 *Let R be a set of configuration rules and let $Tr(R)$ be the resulting rules obtained by applying Algorithm 4 to R. Then the following statements hold: (1) Ordering the rules in $Tr(R)$ is no longer relevant; (2) $Tr(R)$ is completely free of anomalies.*

Proof of Theorem 4 For any pair of rules R_i and R_j such that R_i comes before R_j, R_j is replaced by a rule R'_j obtained by recursively replacing R_j by $exclusion(R_j, R_k)$ for any $k < j$.

Then, by recursively applying Lemma 3, it is possible to commute rules R'_i and R'_j in $Tr(R)$ without changing the policy.

Regarding the second statement – $Tr(R)$ is completely free of anomalies – notice that, in $Tr(R)$, each rule is independent of all other rules.

Thus, if we consider a rule R_i in $Tr(R)$ such that $R_i[condition] \neq \emptyset$, then this rule will apply to any packet that satisfies $R_i[condition]$, i.e., it is not shadowed.

On the other hand, rule R_i is not redundant because if we remove this rule, since this rule is the only one that applies to packets that satisfy $R_i[condition]$, then configuration of the component will change if we remove rule R_i from $Tr(R)$.

Finally, and after the execution of Algorithm 4 over the initial set of configuration rules, one may verify that for each rule R_i in $Tr(R)$ the following conditions hold:

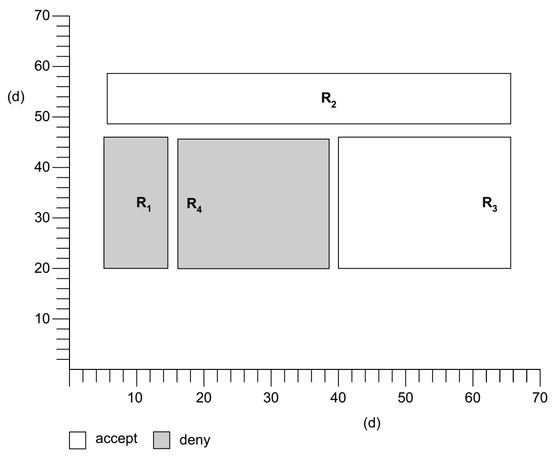
(1) $s = z_1 \cap source(r) \neq \emptyset$ and $d = z_2 \cap dest(r) \neq \emptyset$ such that $z_1 \neq z_2$ and component c is in $minimal\_route(z_1, z_2)$;

(2) if $A_c = attack\_category(R_i) \neq \emptyset$, the predicate $affects(A_c, z_2)$ becomes $true$.

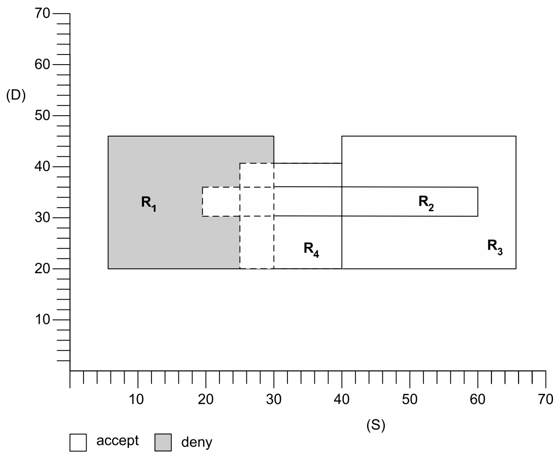
Thus, each rule R_i in $Tr(R)$ is not irrelevant. □

Complexity of the Intra-Component Algorithms

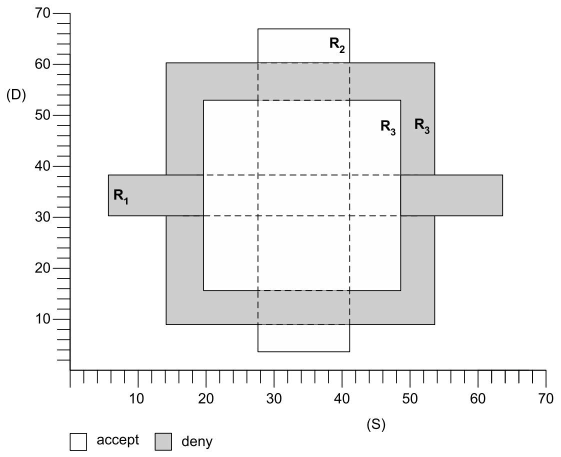
In this section, we shortly discuss the degree of computational complexity of our approach's main algorithm, i.e., Algorithm 1, with respect to the increase of the initial number of rules due to the rewriting process. Indeed, in the worst case (cf. Figure 4.3, Algorithm 1 may generate a huge number of rules. For instance, if we have two rules with p attributes, the second rule can be replaced by p new rules in the worst case, leading to $p+1$ rules.



(a) Best case example



(b) Normal case example



(c) Worst case example

Figure 4.3: Best, normal, and worst ruleset examples.

If we now assume that we have n rules ($n > 2$) with p attributes, then each rule except the first one can be replaced by p new rules in the first rewriting step of the algorithm. In the second step, the p rules that replace the second rule are combined with the p rules that replace rules 3 to n. Thus, each rule from 3 to n can be replaced by p^2 new rules. In the third step, the p^2 rules corresponding to rule 3 are combined with the p^2 rules corresponding to rules 4 to n. We can show that this may lead to p^3 new rules. And so on. Hence, in the worst case, if we have n rules ($n > 2$) with p attributes, then we can obtain $1+p+p^2+\ldots+p^{n-1}$ rules when applying Algorithm 1, that is $\frac{p^n-1}{p-1}$ rules.

Although this complexity seems very high, in all the experimentations we have done (cf. Section 4.4), we were always very far from this case. First, because only attributes source and destination may significantly overlap and exert a bad influence on the algorithm complexity. Other attributes, protocols and source and destination port numbers, are generally equal or completely different when combining configuration rules. Second, administrators generally use overlapping rules in their configurations to represent rules that may have *exceptions*. This situation is closer to the normal case presented in Figure 4.3 than to the worst case. Third, when shadowing or redundancy situations are discovered by the algorithm, some rules are removed – which significantly reduces the algorithm complexity.

Default policies

Each component implements a positive (i.e., close) or negative (i.e., open) policy. If it is positive, the default decision is to $alert$ or to $deny$ a packet when any configuration rule applies. By contrast, the negative policy will $accepts$ or $pass$ a packet when no rule applies.

After rewriting the rules with the intra-component-audit algorithms (cf. Section 4.2), we can actually remove every rule whose decision is *pass* or *accept* if the policy of this component is negative (else this rule is redundant with the default policy); and similarly we can remove every rule whose decision is *deny* or *alert* if its policy is positive. Thus, we can consider that our proposed *intra-component-audit* algorithm generates a configuration that only contains positive rules if the component default policy is negative, and negative rules if the default policy is positive.

4.3 Inter-component Classification and Algorithms

The objective of the inter-component audit algorithms is the complete detection of policy anomalies that could exist in a multi-component policy, i.e., to discover and warn the security officer about potential anomalies between policies of different components.

The main hypotheses to deploy our algorithms hold the following:

(1) An upstream traffic flows away from the closest component to the origin of this traffic (i.e., the most-upstream component [Al-Shaer et al., 2005]) towards the closest component to the remote destination (i.e., the most-downstream component [Al-Shaer et al., 2005]);

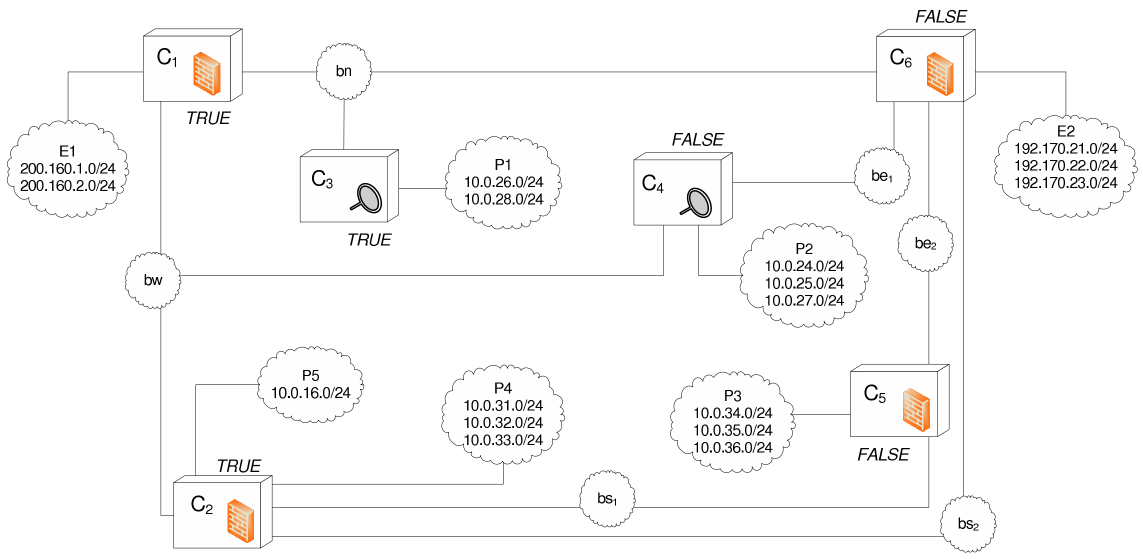
(2) Every component's policy in the network has been rewritten using the intra-component algorithms defined in Section 4.2, i.e., it does not contain intra-component anomalies and the rules within such a policy are completely independent between them.

Inter-Component Anomalies Classification

In this section, we classify the complete set of anomalies that can occur within a multi-component policy. Our classification is based on the network model presented in Section 4.1. An example for each anomaly will be illustrated through the distributed multi-component policy setup shown in Figure 4.4.

Inter-Component Shadowing A shadowing anomaly occurs between two components whether the following conditions hold: (1) The most-upstream component is a firewall; (2) The downstream component, where the anomaly is detected, does not block or report (completely or partially) traffic that is blocked (explicitly, by means of positive rules; or implicitly, by means of its default policy), by the most-upstream component.

The explicit shadowing as result of the union of rules $C_6\{R_7\}$ and $C_6\{R_8\}$ to the traffic that the component C_3 matches by means of rule $C_3\{R_1\}$ is a proper example of *full shadowing* between a firewall and a NIDS. Similarly, the anomaly between $C_3\{R_2\}$ and $C_6\{R_8\}$ shows an example of an *explicit partial shadowing* anomaly between a firewall and a NIDS.



$C_1\{R_1\}$: {tcp 200.160.1.[0,255]:any, 10.0.16.[0,255]:any} → false
$C_1\{R_2\}$: {tcp 200.160.1.[0,255]:any, 10.0.16.[0,255]:any} → false
$C_1\{R_3\}$: {tcp 192.170.21.[20,33]:any, 200.160.1.[20,30]:any} → false
$C_1\{R_4\}$: {tcp 192.170.21.[60,80]:any, 200.160.1.[20,30]:any} → false
$C_1\{R_5\}$: {tcp 10.0.33.[0,30]:any, 200.160.2.[10,30]:any} → false
$C_1\{R_6\}$: {tcp 10.0.31.[10,20]:any, 200.160.1.[0,255]:any} → false
$C_1\{R_7\}$: {tcp 10.0.33.[0,255]:any, 200.160.1.[10,12]:any} → false

$C_2\{R_1\}$: {tcp 10.0.31.[15,17]:any, 200.160.1.[0,255]:any} → false
$C_2\{R_2\}$: {tcp 10.0.32.[0,70]:any, 10.0.35.[0,255]:any} → false
$C_2\{R_3\}$: {tcp 10.0.32.[0,70]:any, 200.160.2.[0,255]:any} → false
$C_2\{R_4\}$: {tcp 10.0.33.[0,255]:any, 200.160.1.[0,255]:any} → false
$C_2\{R_5\}$: {tcp 10.0.32.[0,70]:any, 10.0.25.[0,255]:any} → false
$C_2\{R_6\}$: {tcp 200.160.[1.0,2.255]:any, 10.0.16.[0,255]:any} → false

$C_3\{R_1\}$: {tcp 10.0.31.[15,17]:any, 200.160.1.[0,255]:any, $p_{3\text{-}1}$, $c_{3\text{-}1}$} → false
$C_3\{R_2\}$: {tcp 10.0.32.[0,70]:any, 10.0.35.[0,255]:any, $p_{3\text{-}2}$, $c_{3\text{-}2}$} → false

$C_4\{R_1\}$: {tcp 10.0.32.10:any, 10.0.25.[0,255]:any, $p_{4\text{-}1}$, $c_{4\text{-}1}$} → true
$C_4\{R_2\}$: {tcp 10.0.32.[60,80]:any, 10.0.25.[0,255]:any, $p_{4\text{-}2}$, $c_{4\text{-}2}$} → true
$C_4\{R_3\}$: {tcp 192.170.22.[15,30]:any, 10.0.24.[0,255]:any, $p_{4\text{-}3}$, $c_{4\text{-}3}$} → true
$C_4\{R_4\}$: {tcp 192.170.23.[0,255]:any, 10.0.24.[0,255]:any, $p_{4\text{-}4}$, $c_{4\text{-}4}$} → true
$C_4\{R_5\}$: {tcp 192.170.21.[18,20]:any, 10.0.27.[0,255]:any, $p_{4\text{-}5}$, $c_{4\text{-}5}$} → true

$C_6\{R_1\}$: {tcp 192.170.22.[0,255]:any, 10.0.34.[0,255]:any} → true
$C_6\{R_2\}$: {tcp 192.170.23.[18,20]:any, 10.0.34.[0,255]:any} → true
$C_6\{R_3\}$: {tcp 192.170.21.[10,40]:any, 200.160.1.[0,255]:any} → true
$C_6\{R_4\}$: {tcp 192.170.21.[65,70]:any, 200.160.1.[0,255]:any} → true
$C_6\{R_5\}$: {tcp 192.170.22.[0,255]:any, 10.0.24.[0,255]:any} → true
$C_6\{R_6\}$: {tcp 192.170.23.[18,20]:any, 10.0.24.[0,255]:any} → true
$C_6\{R_7\}$: {tcp 192.170.21.[10,40]:any, 10.0.26.[0,255]:any} → true
$C_6\{R_8\}$: {tcp 192.170.21.[65,70]:any, 10.0.26.[0,255]:any} → true

$C_5\{R_1\}$: {tcp 10.0.32.10:any, 10.0.35.[0,255]:any} → true
$C_5\{R_2\}$: {tcp 10.0.32.[60,80]:any, 10.0.35.[0,255]:any} → true
$C_5\{R_3\}$: {tcp 192.170.22.[15,30]:any, 10.0.34.[0,255]:any} → true
$C_5\{R_4\}$: {tcp 192.170.23.[0,255]:any, 10.0.34.[0,255]:any} → true
$C_5\{R_5\}$: {tcp 192.170.21.[18,20]:any, 10.0.36.[0,255]:any} → true

Figure 4.4: An example for a distributed network policy setup.

On the other hand, the implicit shadowing between the rule $C_1\{R_5\}$ and the default policy of component C_2 is a proper example of *implicit full shadowing* between two firewalls. Finally, the anomaly between the rule $C_1\{R_6\}$, $C_2\{R_1\}$, and the default policy of component C_2 shows an example of an *implicit partial shadowing* anomaly between two firewalls.

Inter-Component Redundancy A redundancy anomaly occurs between two components whether the following conditions hold: (1) The most-upstream component is a firewall; (2) The downstream component, where the anomaly is detected, blocks or reports (completely or partially) traffic that is blocked by the most-upstream component.

A proper example of *full redundancy* between two firewalls is shown by rules $C_5\{R_3\}$ and $C_6\{R_1\}$; rules $C_4\{R_3\}$ and $C_6\{R_5\}$, on the other hand, show an example of *full redundancy* between a firewall and a NIDS.

Similarly, rules $C_5\{R_4\}$ and $C_6\{R_2\}$ show a proper example of *partial redundancy* between two firewalls, whereas rules $C_4\{R_4\}$ and $C_6\{R_6\}$ show an example of *partial redundancy* between a firewall and a NIDS.

Sometimes, this kind of redundancy is expressly introduced by network administrators (e.g., to guarantee the forbidden traffic will not reach the destination). Nonetheless, it is important to discover it since, if such a rule is applied, we may conclude that at least one of the redundant components is wrongly working.

Inter-Component Misconnection A misconnection anomaly occurs between two components whether the following conditions hold: (1) The most-upstream component is a firewall; (2) the most-upstream firewall permits (explicitly, by means of negative rules; or implicitly, through its default policy) all the traffic – or just a part of it – that is denied or alerted by a downstream component.

An explicit misconnection anomaly between two firewalls is shown through the rules $C_5\{R_1\}$ and $C_2\{R_2\}$ (*full misconnection*); and the rules $C_5\{R_2\}$ and $C_2\{R_2\}$ (*partial misconnection*).

An implicit misconnection anomaly between two firewalls is also shown by the rule $C_1\{R_5\}$ and the default policy of firewall C_2 (*full misconnection*); and the rules $C_1\{R_6\}$ and $C_2\{R_1\}$, together with the default policy of C_2 (*partial misconnection*).

Similarly, the pair of rules $C_4\{R_1\}$-$C_2\{R_5\}$ and the pair of rules $C_4\{R_2\}$-$C_2\{R_5\}$ show, respectively, an explicit example of full and partial misconnection anomaly between a firewall and a NIDS.

Finally, the rule $C_4\{R_5\}$ together with the negative policy of the firewall C_2 shows an example of implicit misconnection anomaly between a firewall and a NIDS.

Inter-Component Analysis Algorithms

For reasons of clarity, we split the whole analysis process in four different algorithms. The input for the first algorithm (cf. Algorithm 5) is the set of components C, such that for all $c \in C$, we note $c[rules]$ as the set of configuration rules of component c, and $c[policy] \in \{true, false\}$ as the default policy of such a component c.

In turn, each rule $r \in c[rules]$ consists of a boolean expression over the attributes $szone$ (source zone), $dzone$ (destination zone), $sport$ (source port), $dport$ (destination port), $protocol$, and $decision$ (true or false).

Algorithm 5: `inter-component-audit`(C)

```
foreach $c \in C$ do
    foreach $r \in c[rules]$ do
        $Z_s \leftarrow \{z \in Z \mid z \cap$ source $(r) \neq \emptyset\}$;
        $Z_d \leftarrow \{z \in Z \mid z \cap$ dest $(r) \neq \emptyset\}$;
        foreach $z_1 \in Z_s$ do
            foreach $z_2 \in Z_d$ do
                audit $(c,r,z_1,z_2)$;
```

Algorithm 6: `audit`(c,r,z_1,z_2)

```
foreach $p \in$ minimal_route $(z_1,z_2)$ do
    $path_d \leftarrow$ tail $(p,c)$;
    $path_u \leftarrow$ header $(p,c)$;
    if $path_d \neq \emptyset$ and $r[decision] =$”$false$”
    and isFirewall $(c)$ then
        $c_d \leftarrow$ first$(path_d)$;
        downstream $(r,c,c_d)$;
    if $path_u \neq \emptyset$ then
        $c_u \leftarrow$ last$(path_u)$;
        if isFirewall $(c_u)$ then
            upstream $(r,c,c_u)$;
```

Algorithm 7: `downstream`(r,c,c_d)

```
if $c_d[policy] = true$ then
    $R_{df} \leftarrow \{r_d \in c_d \mid r_d \backsim r \wedge r_d[decision] = false\}$;
    if $R_{df} = \emptyset$ then warning ("$Full\ Misconnection$ ");
    else if $\neg$ testRedundancy ($R_{df}$,$r$) then
        warning ("$Partial\ Misconnection$");
```

Algorithm 8: `upstream`(r,c,c_u)

```
$R_{uf} \leftarrow \{r_u \in c_u \mid r_u \backsim r \wedge r_u[decision] = false\}$;
$R_{ut} \leftarrow \{r_u \in c_u \mid r_u \backsim r \wedge r_u[decision] = true\}$;
if $r[decision] =$"$true$" then
    if testRedundancy ($R_{uf}$,$r$) then
        warning ("$Full\ Spurious$");
    else if $R_{ua} \neq \emptyset$ then
        warning ("$Partial\ Spurious$");
    else if testRedundancy ($R_{ut}$,$r$) then
        warning ("$Full\ Redundancy$");
    else if $R_{ut} \neq \emptyset$ then
        warning ("$Partial\ Redundancy$");
    else if $R_{uf} = \emptyset$ and $R_{ut} = \emptyset$
    and $c_u[policy] = false$ then
        warning ("$Full\ Misconnection$");
else
    if testRedundancy ($R_{ut}$,$r$) then
        warning ("$Full\ Shadowing$");
    else if $R_{ut} \neq \emptyset$) then
        warning ("$Partial\ Shadowing$");
    else if $R_{uf} = \emptyset$ and $c_u[policy] = true$ then
        warning ("$Full\ Shadowing$");
    else if $\neg$ testRedundancy ($R_{uf}$,$r$)
    and $c_u[policy] = true$ then
        warning ("$Partial\ Shadowing$");
```

Let us recall here the functions $source(r) = szone$ and $dest(r) = dzone$. Thus, we compute for each component $c \in C$ and for each rule $r \in c[rules]$, each one of the source zones $z_1 \in Z_s$ and destination zones $z_2 \in Z_d$ – whose intersection with respectively $szone$ and $dzone$ is not empty – which become, together with a reference to each component c and each rule r, the input for the second algorithm (i.e., Algorithm 6).

Once in Algorithm 6, we compute the minimal route of components that connects zone z_1 to z_2, i.e., $[C_1, C_2, \ldots, C_n] \in minimal\_route(z_1, z_2)$. Then, we decompose the set of components inside each path in downstream path ($path_d$) and upstream path ($path_u$). To do so, we use functions $head$ and $tail$. The first component $c_d \in path_d$, and the last component $c_u \in path_u$ are passed, respectively, as argument to the last two algorithms (i.e., Algorithm 7 and Algorithm 8) in order to conclude the set of necessary checks that guarantee the audit process.

The operator "$\backsim$" within algorithms 7 and 8 denotes that two rules r_i and r_j are correlated if every attribute in R_i has a non empty intersection with the corresponding attribute in R_j.

Let us conclude by giving an outlook to the set of warnings send to the security officer after the execution of Algorithm 5 over the scenario of Figure 4.4:

$C_1\{R_3\} - C_6\{R_3, R_4\}$: Full Shadowing
$C_1\{R_4\} - C_6\{R_4\}$: Partial Shadowing
$C_1\{R_5\} - C_2\{pol.\}$: Full Shadowing
$C_1\{R_6\} - C_2\{R_1, pol.\}$: Partial Shadowing

$C_2\{R_3\} - C_1\{pol.\}$: Full Misconnection
$C_2\{R_4\} - C_1\{R_7, pol.\}$: Partial Misconnection

$C_3\{R_1\} - C_6\{R_7, R_8\}$: Full Shadowing
$C_3\{R_2\} - C_6\{R_8\}$: Partial Shadowing

$C_4\{R_1\} - C_2\{R_5\}$: Full Misconnection
$C_4\{R_2\} - C_2\{R_5\}$: Partial Misconnection
$C_4\{R_3\} - C_6\{R_5\}$: Full Redundancy
$C_4\{R_4\} - C_6\{R_6\}$: Partial Redundancy
$C_4\{R_5\} - C_6\{pol.\}$: Full Misconnection

$C_5\{R_1\} - C_2\{R_2\}$: Full Misconnection
$C_5\{R_2\} - C_2\{R_2\}$: Partial Misconnection
$C_5\{R_3\} - C_6\{R_1\}$: Full Redundancy
$C_5\{R_4\} - C_6\{R_2\}$: Partial Redundancy
$C_5\{R_5\} - C_6\{pol.\}$: Full Misconnection

Correctness of the Inter-Component Algorithms

To prove the correctness of our Inter-Component Algorithms, we first define what is a deployment without anomalies for a set of rules. For this purpose, let us consider a set R of configuration rules to be deployed over a set C of components that partitions a network into a set Z of zones. We also assume that C has been rewritten by our intra-component-audit algorithm (cf. Section 4.2).

Let us now consider a rule $r \in R$ and let us assume that r applies to a source zone z_1 and a destination zone z_2, i.e., $s = z_1 \cap source(r) \neq \emptyset$ and $d = z_2 \cap dest(r) \neq \emptyset$. Let r' be a rule identical to r except that $source(r') = s$ and $dest(r') = d$. Finally, let us assume that $[C_1, C_2, ..., C_k] \in minimal\_route(z_1, z_2)$.

Deployment algorithm It defines how any rule $r \in R$ will be deployed over the set C of components. There are two different cases: $r[decision] = false$ or $r[decision] = true$.

If $r[decision] = false$ then, on every component on the minimal route from source s to destination d, deploy a negative rule (i.e., an $accept$ filtering rule if the component is a firewall, or a $pass$ alerting rule if the component is a NIDS).

Conversely, if $r[decision] = true$ then, the following two possibilities hold: (1) if r is a filtering rule, then deploy a $deny$ filtering rule on the most-upstream firewall on the minimal route (if such a firewall does not exist, then generate a deployment error message); (2) if r is an alerting rule, then deploy an $alert$ rule on the most-upstream NIDS on the minimal route (if such a NIDS does not exist, then generate a deployment error message).

Based on this deployment algorithm, we can now prove the following theorem:

Theorem 5 *Let C be a set of components. The inter-component algorithms presented in Section 4.3 do not detect any anomaly in the configurations of C iff there is a set R of rules such that configurations of C are obtained by applying the deployment algorithm showed above.*

4.4 Implementation and Performance Evaluation

We implemented the intra and inter-component algorithms presented in this chapter in a software prototype called MIRAGE (which stands for MIsconfiguRAtion manaGEr). MIRAGE has been developed using PHP, a general-purpose scripting language that is especially suited for web services development and can be embedded into HTML for the construction of client-side GUI based applications [Castagnetto et al., 1999]. MIRAGE can be locally or remotely executed by using a HTTP server (e.g., Apache server over UNIX or Windows setups) and a web browser.

We evaluated our algorithms through a set of experiments over two different IPv4 real networks. The topology for the first network consisted of a single firewall based on netfilter [Welte et al., 2006], and a single NIDS based on snort [Roesch, 1999] – connected to three different zones with more than 50 hosts. The topology for the second network consisted of six different components – based on netfilter, ipfilter [Reed, 2005], and snort – protecting six different zones with more than 200 hosts. The whole of these experiments were carried out on an Intel-Pentium M 1.4 GHz processor with 512 MB RAM, running Debian GNU/Linux 2.6.8, and using Apache/1.3 with PHP/4.3 configured.

During a first phase, we measured the memory space and the processing time needed to perform Algorithm 4 over several sets of IPv4 policies for the first IPv4 network, according to the three following security officer profiles: beginner, intermediate, and expert – where the probability to have overlaps between rules increases from 5% to 90%. The results of these measurements are plotted in Figure 4.5 and Figure 4.6. Though those plots reflect strong memory and process time requirements, we consider they are reasonable for off-line analysis, since it is not part of the critical performance of a single component.

We conducted, in a second phase, similar experiments to measure the performance and scalability of Algorithm 5 through a progressive increment of auto-generated rules, firewalls and zones for the second network. The results of these measurements are plotted in Figure 4.7 and Figure 4.8. Similarly to the intra-component evaluation, we consider these requirements very reasonable for off-line inter-component analysis.

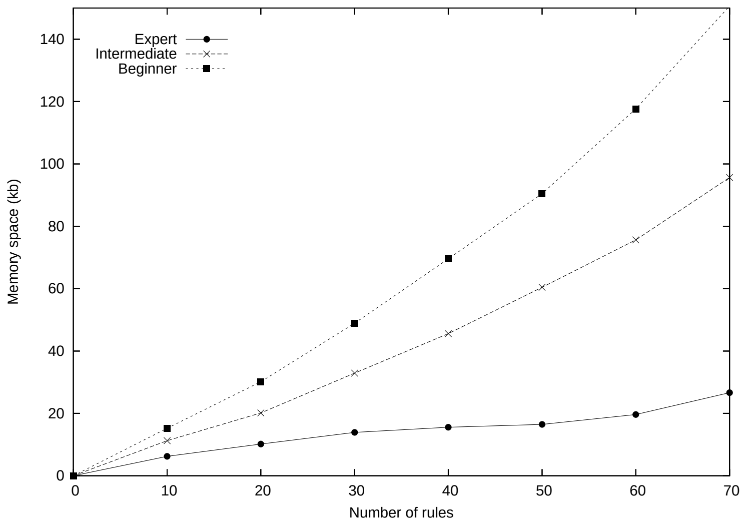


Figure 4.5: Memory space evaluation for the intra-component algorithms.

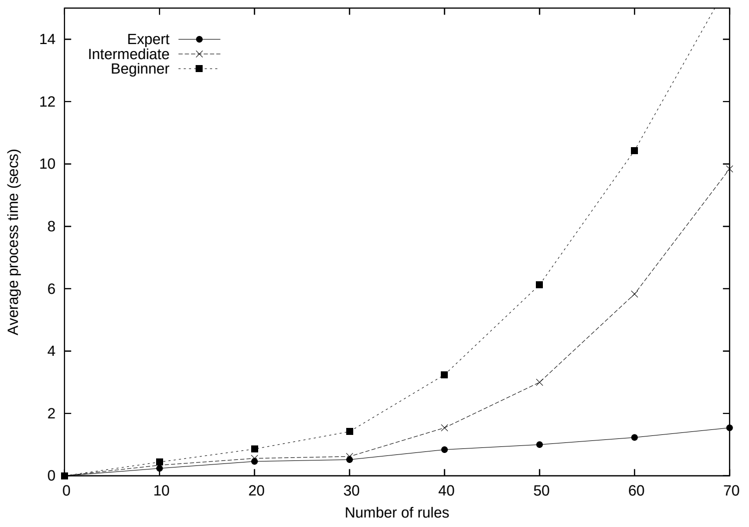


Figure 4.6: Processing time evaluation for the intra-component algorithms.

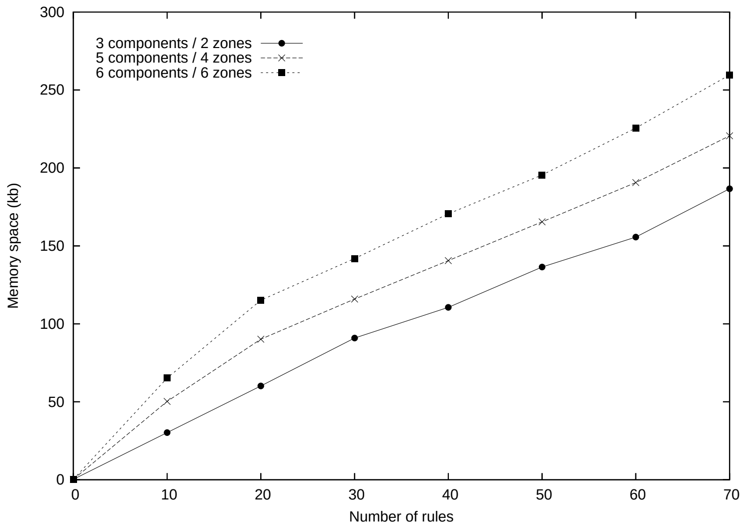


Figure 4.7: Memory space evaluation for the inter-component algorithms.

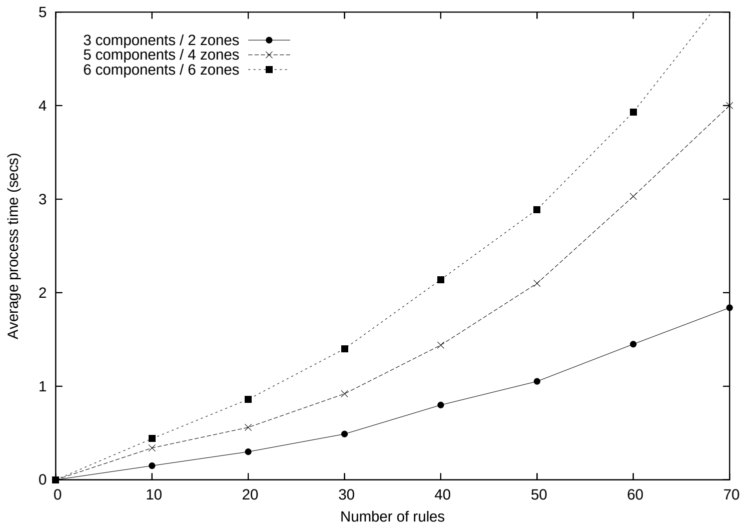


Figure 4.8: Processing time evaluation for the inter-component algorithms.

Summary

We have presented in this chapter an audit process to set a distributed access control policy free of anomalies. Our audit process has been presented in three main blocks. We first presented in Section 4.1 a network model to determine which components are crossed by a given packet knowing its source and destination. We then presented in Section 4.2 a set of algorithms for the discovering and removal of policy anomalies over single-component environments. We finally presented in Section 4.3 a set of algorithms for the detection and reporting of anomalies over a multi-component environments.

The advantages of our set of algorithms, regarding the related work presented in Section 3.1 of Chapter 3, are threefold. First of all, our approach not only considers the analysis of rules two by two but also a complete analysis of the whole set of rules. This way, those conflicts due to the union of rules that are not detected in [Al-Shaer et al., 2005] are properly discovered by our intra- and inter-component algorithms. Second, after applying our intra-component algorithms the resulting rules of each component are totally disjoint, i.e., the ordering of rules is no longer relevant. Hence, one can perform a second transformation in a positive or negative manner, generating a configuration that only contains positive rules if the component default policy is negative, and negative rules if the default policy is positive. Third, our approach also presents a network model to determine which components are crossed by a given packet knowing its source and destination, as well as other network properties. Thanks to this model, we better define all the set of both intra- and inter-anomalies. Furthermore the lack of this model in [Al-Shaer et al., 2005] may lead to inappropriate decisions.

We also presented along Section 4.2 and Section 4.3 the correctness of our algorithms and it complexity. In Section 4.4, moreover, we discussed the implementation of our set of algorithms in a software prototype and a first evaluation of such an implementation. The results of performance of our implemented prototype demonstrates the practicability of our work. Although these results show that our algorithms have strong memory and time processing requirements, we believe that these requirements are reasonable for off-line analysis, since it is not part of the critical performance of the audited components.

Chapter 5

Infrastructure for the Exchange of Messages and Audit Information

"I've ignored stop signs, I've jaywalked, I've even opened fires on Jones Beach.
But this is the U.S. Mail! And since I was old enough to lick a stamp,
I was inculcated with the sanctity, the inviolability of the mail."
– JOEL FLEISCHMAN (NORTHERN EXPOSURE)

As pointed out in Chapter 3 (cf. Section 3.2), traditional client/server solutions for the exchange of audit information between security components can quickly become a bottleneck – due to saturation problems associated with the service offered by centralized or master domain analyzers. On the one hand, traditional systems like DIDS [Snapp et al., 1991] and NADIR [Hochberg et al., 1993] process their data in a central node although the collection of data is distributed. These schemes are straightforward as they simply push data to a central node and perform the computation there. On the other hand, hierarchical approaches, such as GrIDS [Staniford-Chen et al., 1996] and NetSTAT [Vigna and Kemmerer, 1999], have a layered structure where data is locally preprocessed and filtered. Although they mitigate some weaknesses present in centralized schemes, they still cannot avoid vulnerabilities at the root level. In contrast to these traditional designs, alternative approaches try to eliminate the need for dedicated elements. The idea of distributing the detection process has some advantages regarding centralized and hierarchical approaches. Mainly, decentralized architectures have no single point of failure and bottlenecks can be avoided. Some message passing designs, such as CSM [White et al., 1999] and Quicksand [Kruegel, 2002], try to eliminate the need for dedicated elements by introducing a peer-to-peer architecture. Instead of having a central monitoring station to which all data has to be forwarded, there are independent uniform working entities at each host performing similar basic operations, i.e., the different entities collaborate on the detection activities.

These designs seem to be a promising technology to implement decentralized architectures for the detection of attacks. However, the presented systems still exhibit very simplistic designs and suffer from several limitations. For instance, in some of them, every node has to have complete knowledge of the system; all nodes have to be connected to each other which can make the matrix of the connections, that are used for providing the alert exchanging service, grow explosively and become very costly to control and maintain. Another important disadvantage present in this design is that the different entities always need to know where a received notification has to be forwarded (similar to a queue manager). This way, when the number of possible destinations grows, the network view can become extremely complex, which leads to a system that is not scalable. Other designs are based on flooding which makes the system easier to maintain on the cost of scalability, as the message complexity grows fast with the number of nodes.

Most of these limitations can be solved efficiently by using a publish/subscribe based system. The advantage of this model for our problem domain over other communication paradigms is on the one hand that it keeps the producer of messages separated from the consumer and on the other hand that the communication is information-driven. This way, it can avoid problems regarding the scalability and the management inherent to other designs, by means of a network of publishers, brokers, and subscribers. A publisher in a publish/subscribe system does not need to have any knowledge about any of the entities that consume the published information. Likewise, the subscribers do not need to know anything about the publishers. New services can simply be added without any impact on or interruption of the service to other users.

The rest of this chapter is organized as follows. We start with an introduction to the publish/subscribe communication paradigm in Section 5.1. In Section 5.2 we briefly overview the Intrusion Detection Message Exchange Format (IDMEF), which is the format we use in order to exchange audit information in our proposal. We then discuss in Section 5.3 our communication mechanism and the current state of our implementation based on xmlBlaster, an open source publish/subscribe message oriented middleware [Ruff, 2006]. We finally conclude this chapter in Section 5.4 by giving an outlook of the performance obtained with a first deployment of such an implementation.

5.1 Publish/Subscribe Model

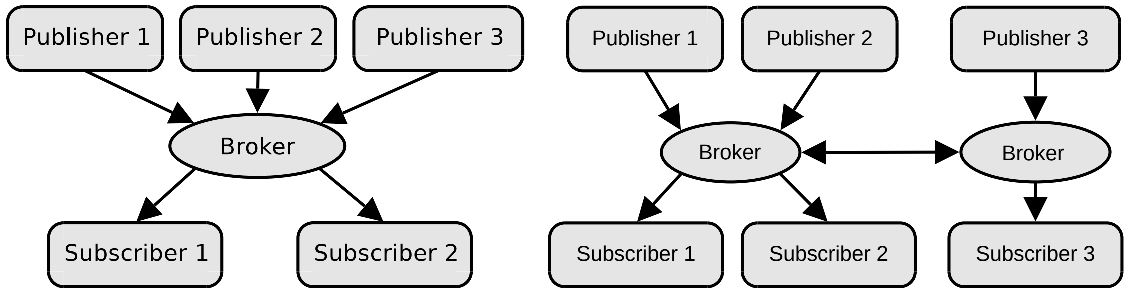
The publish/subscribe model is an asynchronous, many-to-many communication model which is intended for distributed systems [Eugster et al., 2003]. It is often used in those situations where a message (often referred in the literature as *notification*) sent by a single entity is required by, and should be distributed to, multiple entities. It is often used for efficient and comfortable information dissemination to group members which may have individual interests in arbitrary subsets of messages published. In contrast to multicast communication, clients have the possibility to describe the events they are interested in more precisely (e.g. based on the contents of the notification). Clients can choose to either subscribe or unsubscribe to messages as time goes by, and all the subscribers are independent of each other. More formally, and according to [Pietzuch, 2004], we can define the publish/subscribe communication paradigm as follows.

Definition – *The Publish/Subscribe model is based on the use of message publishers – which produce information (or message publications) – and message subscribers – which receive such messages. Message subscribers describe the kind of messages that they want to receive with a message subscription. Messages coming from message publishers will subsequently be delivered to all interested message subscribers with matching interests.*

Publish/Subscribe Systems

A publish/subscribe system implements the publish/subscribe model and consists of, at least, one broker forwarding notifications published by clients to other clients that are interested in them. For scalability reasons, it is common to implement a distributed broker network that forms a so-called *notification service* through an overlay network consisting of brokers.

This service provides a distributed infrastructure for notification routing which includes the management of subscriptions and the dissemination of notifications in a possibly asynchronous way. Clients can publish notifications and subscribe to filters that are matched against the notifications passing through the broker network. If a broker receives a new notification it checks if there is a local client that has subscribed to a filter that matches this notification. If so, the message is delivered to this client. Additionally, the broker forwards the message to neighbor brokers according to the applied routing algorithm. We refer to [Mühl, 2002] for a good survey on the field.



(a) Simple publish/subscribe system. (b) Extended pub/sub system.

Figure 5.1: Examples of simple publish/subscribe topologies.

An example of a simple centralized publish/subscribe system is shown in Figure 5.1(a). Here, five clients are connected to a single broker: three clients that are publishing notifications and two clients that are subscribed to a subset of the notifications published on the broker. Subscribers can choose to subscribe to the notifications available through the broker or cancel existing subscriptions as needed. The broker matches the notifications it received from the publishers to the subscriptions, ensuring this way that every publication is delivered to all interested subscribers.

This very basic publish/subscribe setup can be extended by connecting multiple brokers (cf. Figure 5.1(b)), enabling them to exchange messages. The extended design allows subscribers on one of the brokers to receive messages that have been published on another broker, further freeing the subscriber from the constraints of connecting to the same broker the publisher is connected to. Most available implementation make this transparent for the programmer by keeping the same interface operations as in the centralized design. This way, an application can easily be distributed. In Figure 5.2, for instance, we show a distributed publish/subscribe topology, where a client p publishes a notification n that is matched by filter F, client s subscribed to, while the notification service takes care of forwarding the notification properly.

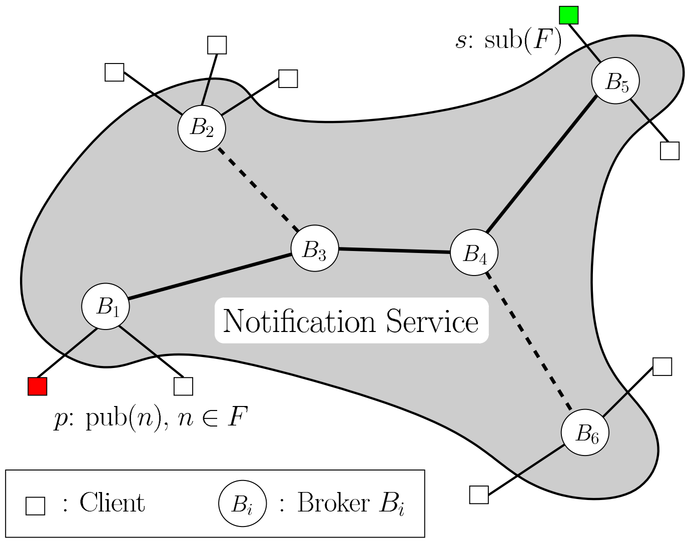


Figure 5.2: Example of a distributed publish/subscribe topology.

Regarding the subscription of information, clients are able to formulate their interests based, mainly, on the contents of the notifications and a special attribute they carry. This is known as content-based and topic-based subscription, respectively.

Topic-based publish/subscribe systems, on the one hand, are the earliest variant of the publish/subscribe communication model. Here, publishers publish messages with respect to a *topic* or *subject*, and subscribers specify their interest in a topic and receive all messages published on this topic. Topic-based subscriptions are, in turn, easier to handle than content-based subscriptions. Since topics can be seen as groups in group communication [Powell, 1996], topic-based subscription may efficiently be built on top of a group communication mechanism such as, for example, IP multicast [Deering, 1989].

Two different matching mechanisms are commonly used in topic-based publish/subscribe systems. One matches subscriptions successfully to notifications if the topic of the subscription exactly matches the topic under which the notification is published. Using this mechanism, topics become equivalent to *channels*. The other mechanism arranges topics in a subject tree such that subscriptions not only match notifications if the topics are the same, but also if the topic of the subscription is an ancestor of the notification topic in the subject tree (in this case, a topic becomes equivalent to a *theme*).

Content-based publish/subscribe systems, on the other hand, allow subscriptions to evaluate the whole content of notifications. This way, in content-based selection the structure of a subscription is not restricted to a topic or a theme – it can be any function over the content of a message. Here, a subscription can be formulated extremely fine-grained based on the content of notifications using a query language that can be arbitrarily complex. Moreover, there does not have to be a system wide agreement on the set of topics as it is generally a good idea for topic based routing. In particular, this is important for applications that run on mobile devices with limited processing power and network bandwidth.

Content-based subscriptions usually depend on the structure of the message. It can be binary data, name/value pairs, semi-structured data, or even programming language classes with executable code. A subscription is often expressed in a subscription language that specifies a filter expressions over messages. For our work, we propose the use of content-based subscription over messages with semi-structured data. Indeed, we propose the use of XML for the structure of a message, and the use of XPath as the subscription language to specify filter expressions (cf. Section 5.3). In the following, we give an outlook on the main properties of the format built on top of the XML structure of our messages.

5.2 Representation of Messages

In order to exchange audit information in a standard manner, two main specifications have been considered (cf. Chapter 3, Section 3.3). The Common Intrusion Specification Language (CISL), on the one hand, which was initially proposed to allow the components of the Common Intrusion Detection Framework (CIDF) to exchange data in semantically well-defined ways [Feiertag et al., 1999]. The Intrusion Detection Message Exchange Format (IDMEF), on the other hand, was proposed by the IETF's Intrusion Detection Exchange Format Working Group (IDWG) to accomplish similar purposes [Debar et al., 2006].

Our approach is based on the IDMEF format for three main reasons. First, this format is the basis for the similarity operator used on the aggregation and fusion phases of our alert correlation approach (cf. Chapter 6). Second, there is a significant number of current tools and implementations based on the IDMEF format, such as [Migus, 2006], which makes it easy to integrate it in our work. Third, the exchange of messages between the components of our framework is compliant with the intrusion detection framework proposed by the IDWG. Furthermore, IDMEF allows the specification of messages generated by different network security components, such as *firewalls* and *network intrusion detection systems* (NIDSs), and it can be extended to incorporate additional data information, such as diagnoses and counter-measures, inside their proposed format.

Up to now, IDMEF is an *internet draft* approved by the IESG (Internet Steering Group) as an IETF's RFC (Request For Comments). It is represented in an object-oriented fashion. The class hierarchy of IDMEF has been represented by using the Extensible Markup Language (XML). The rationale for choosing XML is explained in [Debar et al., 2006], as well some examples of using IDMEF to describe IDS's alerts and the IDMEF's associate Document Type Definition (DTD) – although one may still find the current version of IDMEF defined by using DTDs, the authors also offer a new definition that uses XML Schemas instead of DTDs.

We show in Figure 5.3 the two main kind of messages supported by IDMEF: *heartbeats* and *alerts*. Heartbeats, on the one hand, are periodic messages between components, in order

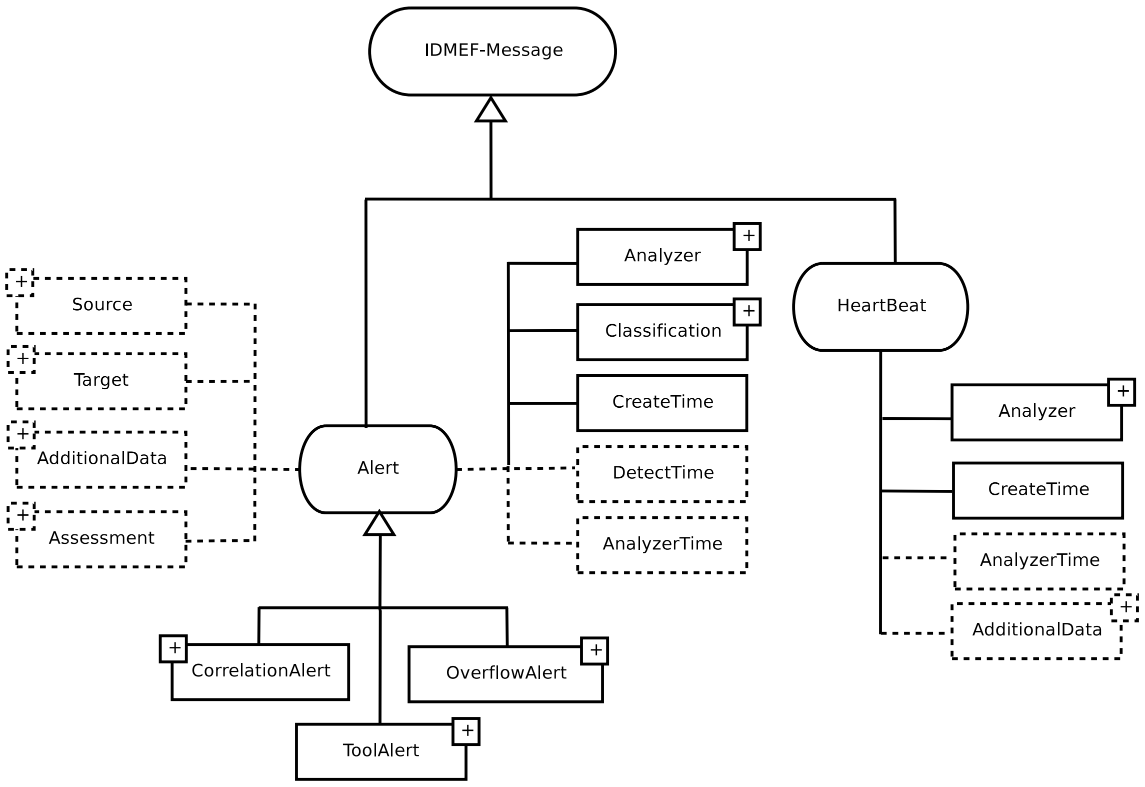


Figure 5.3: The IDMEF's message class.

to inform that they are operational. Alerts, on the other hand, carry audit information, such as the component that produced it, the classification of the detected activity, the source and target ports related to this activity, and other optional data. In the following, we summarize the main properties of IDMEF's alert class, concerning relevant aspects to our work such as determining the component which created the message, the time in which the message was created, the kind of activity the message is pointing out, and so on.

We start by overviewing the analyzer class which identifies the component from which the message originates. Only one component is encoded for each message – i.e., the one at which the message originated. The class is composed, in turn, of three aggregate classes: *node*, which includes information about the node on which the component resides; *process*, which holds information about the process in which the component is executing; and *analyzer*, which carries information about other component which, in turn, has forwarded the original information.

The idea behind the recursive aggregation of component's references within the IDMEF's analyzer class is that when a component receives an IDMEF's alert, and wants to forward it to another component, it needs to substitute the original component information with its own – since, as we pointed out above, just one component is encoded for each message. This way, and in order to preserve the original component information, it may be included in the new component definition as a reference to the previous component. This mechanism will allow component path tracking.

The class analyzer has eight attributes: *analyzerid*, *name*, *manufacturer*, *model*, *version*, *class*, *ostype*, and *osversion*. The *manufacturer*, *model*, *version*, and *class* attributes' contents are vendor-specific, but may be used together to identify different types of components. The *ostype* and *osversion* attributes' contents are, respectively, the operating system name and the operating system version in which the component's process is executed. Finally, the *analyzerid* and *name* attributes' contents provide, respectively, the unique identifier and the explicit name for the component in the system.

Regarding the timestamps of a message, the IDMEF standard defines the following three different classes to represent time: (1) *CreateTime*, which is the time when the message is created by a component; (2) *DetectTime*, which is the time when the event or events that caused the creation of a message were detected; (3) *AnalyzerTime*, which is the time at the original component whether the message has been forwarded. The final object for each instance contains information such as the number of seconds since the *epoch*, the local GMT offset, and the number of microseconds. Even though all the three timestamps can be provided by each component when generating a message, just the one defined by the *CreateTime* class is considered mandatory by the IDMEF standard.

The classes source and target contain, respectively, information about the possible origin and destination of the events that motivated the generation of the message. An event may have more than one source (e.g., a distributed denial of service attack), more than one target (e.g., a port sweep). Both source and target classes are composed of information about the *node*, the *user*, the *process*, and the *network service* that motivated the message. The target class includes, moreover, a list of affected *files*. Referring to their attributes, both source and target classes have the following two common attributes: (1) *ident*, which is a unique identifier for either the source or target class; (2) *interface*, which may be used by a component multiple interfaces to indicate which interface this source or target was seen on. Furthermore, the class source includes the attribute *spoofed*, which indicates whether the source is, as far as the component can determine, a spoofed address. Similarly, the class target includes the attribute *decoy*, to indicate whether the target is, as far as the analyzer can determine, a decoy.

The classification class contains the *name* of the event that motivated the creation of a message, or other information which allows the components to determine what is the message pointing out. It is composed of one aggregate class, the class *reference*, which contains information about external documentation sites, that will provide background information about such an event. Similarly, the assessment class is used to provide the component's assessment of an event, and it is composed of information about the *impact*, *actions* that may be taken in response, and a measurement of the confidence the component has in its evaluation of the event.

Finally, the IDMEF's alert class can be augmented with additional information by means of the aggregate classes *AdditionalData*, *CorrelationAlert*, *ToolAlert*, and *OverflowAlert*. The information aggregated by those classes is often useful in order to associate different messages pointing out to similar activities – and reported by different components – as well as to extend the standard IDMEF model with additional features, such as complex data types and relationships. The *AdditionalData* class, on the first hand, includes information that does not fit into the IDMEF's data model. This may be an atomic piece of data, or a large amount of data. The *CorrelationAlert* class, on the second hand, may include additional information related to the correlation process in which this message is involved. The *OverflowAlert* and *ToolAlert* classes, on the third hand, include, respectively, information related to buffer overflow attacks, and information related to the use of attack tools or other malevolent programs (e.g., *trojan horses*, *rootkits*, and so on).

5.3 Communication Infrastructure

In this section we give an outlook to the operational details of the communication infrastructure presented in [García et al., 2004b, García et al., 2005e]. As our motivation is not targeted on developing a new publish/subscribe system, we try to reuse as much available code and tools as possible. For our experiments (cf. Section 5.4) we used *xmlBlaster*, an open source publish/subscribe message oriented middleware [Ruff, 2006]. It connects a set of nodes that build up the infrastructure for exchanging alerts using the interface operations offered by the underlying middleware.

Each xmlBlaster's message consists of a header filtering that can be applied to, a body, and a system control section. The body of a xmlBlaster's message is formulated using IDMEF format (cf. Section 5.2). On the other hand, filters are XPath expressions that are evaluated over the header to decide if a message has to be delivered to a subscriber. We discuss the essential interface operations offered by xmlBlaster in the following section.

Interface Operations

Conceptually, the alert communication infrastructure offered through xmlBlaster can be viewed as a black box with an *interface* (cf. Figure 5.4). It offers a number of *operations*, each of which may take a number of *parameters*. Clients can invoke *input operations* from the outside, and the system itself invokes *output operations* to deliver information to clients.

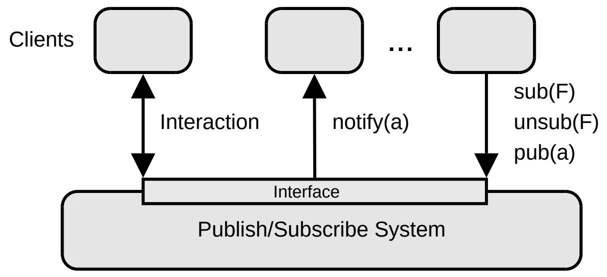


Figure 5.4: Black box view of a publish/subscribe system.

We list the main operations that are of interest for our work in Table 5.1. To publish alerts, clients invoke the *pub*(a) operation, giving the alert a as parameter. The published alert can potentially be delivered to all clients connected to the system via an output operation called *notify*(a). Clients register their interest in specific kinds of alerts by issuing subscriptions via the *sub*(F) operation, which takes a filter F as parameter. Each client can have multiple active subscriptions which must be revoked separately by using the *unsub*() operation.

$pub(C, a)$	Client C publishes alert a
$sub(C, F)$	Client C subscribes to filter F
$notify(C, a)$	Client C is notified about alert a
$unsub(C, F)$	Client C unsubscribes to filter F

Table 5.1: Main interface operations.

All these operations are instantaneous and take parameters from the set of all clients $\mathcal{C}$, set of all alerts $\mathcal{A}$, and the set of all filters $\mathcal{F}$. Formally, a filter $F \in \mathcal{F}$ is a mapping defined by

$$F: \quad a \longrightarrow \{\text{true}, \text{false}\} \qquad \forall a \in \mathcal{A}$$

We say that a *notification* n *matches filter* $F \in \mathcal{F}$ iff $F(a) = \text{true}$. We also assume that each alert can only be published once and that every filter is associated with a unique identifier in order to enable the alert communication infrastructure to identify a specific subscription.

Components and Interactions

As shown in Figure 5.5, and according to the general framework overviewed in Chapter 2 (cf. Section 2.2), each node of the architecture is made up of a set of local analyzers (with their respective detection units or sensors), a set of alert managers (to perform alert processing and manipulation functions), and a set of local reaction units (or effectors). These components, the interactions between them, and the alert communication infrastructure, are described below.

Analyzers, on the first hand, are local elements which are responsible for processing local audit data. They process the information gathered by associated sensors to infer possible

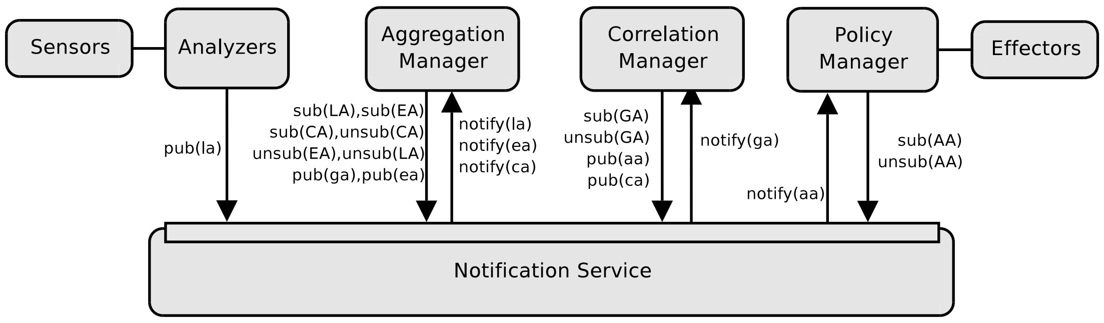


Figure 5.5: Overview of the main components and their interactions.

alerts. Their task is to identify occurrences which are relevant for the execution of the different steps of an attack and pass this information to the correlation manager via the publish/subscribe system. They are interested in local alerts. Each local alert is detected in a sensor's input stream and published through the publish/subscribe system by invoking the *pub*(la) operation, giving the *local alert* la as parameter. Local alerts are exchanged using IDMEF messages (cf. Section 5.2).

Each notification la has a unique classification and a list of attributes with their respective types to identify the analyzer that originated the alert (*AnalyzerID*), the time the alert was created (*CreateTime*), the time the event(s) leading up to the alert was detected in the sensor's input stream (*DetectTime*), the current time on the analyzer (*AnalyzerTime*), and the source(s) and target(s) of the event(s) (*Source* and *Target*). All possible classifications and their respective attributes must be known by all system components (i.e. sensors, analyzers and managers) and all analyzers are capable of publishing instances of local alerts of arbitrary types.

Managers, on the second hand, are the components in charge of performing aggregation and correlation of local alerts and external events. As pointed out in Chapter 3 (cf. Section 3.3), the use of multiple analyzers and sensors together with heterogeneous detection techniques increases the detection rate, but it also increases the number of information to process. In order to reduce the number of false negatives and distribute the load that is imposed by the alerts our architecture provides a set of aggregation and correlation *managers*, which perform aggregation and correlation of both, local alerts (i.e., messages provided by the node's analyzers) and external messages (i.e., the information received from other collaborating nodes). We describe in the following the basic interactions of the two main managers: *aggregation* and *correlation* managers.

Aggregation Manager – The basic functionality of each aggregation manager is to cluster alerts that correspond to the same occurrence of an action (cf. Chapter 3, Section 3.3). Each aggregation manager registers its interest in a subset $\mathcal{L}_A$ of local alerts published by analyzers on the same node by invoking the *sub*(LA) operation, which takes the filter LA as parameter, with

$$LA(a) = \begin{cases} \text{true} & , \ a \in \mathcal{L}_A \\ \text{false} & , \ \text{otherwise.} \end{cases}$$

Similarly, the aggregation manager also registers its interest in a set of related external alerts $\mathcal{E}_A$ by invoking the *sub*(EA) operation with filter EA as parameter, and

$$EA(a) = \begin{cases} \text{true} & , \ a \in \mathcal{E}_a \\ \text{false} & , \ \text{otherwise.} \end{cases}$$

Finally, it registers its interest in local correlated alerts $\mathcal{C}_A$ by invoking the *sub*(CA) operation with

$$CA(a) = \begin{cases} \text{true} & , \ a \in \mathcal{C}_A \\ \text{false} & , \ \text{otherwise.} \end{cases}$$

Once subscribed to these three filters, the communication infrastructure will notify the subscribed managers of all matching alerts via the output operations *notify*(la), *notify*(ea) and *notify*(ca) with $la \in \mathcal{L}_A$, $ea \in \mathcal{E}_A$ and $ca \in \mathcal{C}_A$. All notified alerts are processed and, depending on the clustering and synchronization mechanism, the aggregation manager can publish global and external alerts by invoking *pub*(ga) and *pub*(ea). Finally, it can revoke active subscriptions separately by using the operations *unsub*(CA), *unsub*(EA) and *unsub*(LA).

Correlation Manager – The main task of this manager is the correlation of alerts described in the following chapter (cf. Chapter 6). It operates on the set of global alerts $\mathcal{G}_A$ published by the aggregation manager. To register its interest in these alerts, it invokes *sub*(GA), which takes the filter GA as parameter with

$$GA(a) = \begin{cases} \text{true} & , \quad a \in \mathcal{G}_A \\ \text{false} & , \quad \text{otherwise.} \end{cases}$$

The notification service will then notify the correlation manager of all matched alerts with the output operation *notify*(ga), $ga \in \mathcal{G}_A$. Each time a new alert is received, the correlation mechanism finds a set of action models that can be correlated in order to form a scenario leading to an objective. It then includes this information into the *CorrelationAlert* field of a new IDMEF message and publishes the correlated alert by invoking *pub*(ca), giving the notification $ca \in \mathcal{C}_A$ as parameter. To revoke the subscription, it uses *unsub*(GA).

The correlation manager is also responsible for reacting on detected security violations. The algorithm used is based on the anti-correlation of actions to select appropriate counter-measures in order to reconfigure, for instance, the security policy (cf. Chapter 3, Section 3.3). As soon as a scenario is identified, the correlation mechanism may look for possible action models that can be anti-correlated with the individual actions of the supposed scenario, or even with the goal objective.

The set of anti-correlated actions represents the set of counter-measures available for the observed scenario. The definition of each anti-correlated action contains a description of the counter-measures which should be invoked (e.g. hardening the security policy). Such counter-measures are included into the *Assessment* field of a new IDMEF message and published by invoking *pub*(aa), using the *assessment alert* aa as parameter.

Finally, a *policy manager* will register and revoke its interest in these assessment alerts by invoking *sub*(AA) and *unsub*(AA). Once notified, the policy manager may perform the post-processing of the received alerts before sending them, for example, to a set of associated policy reconfiguration effectors.

5.4 Deployment and Evaluation

In order to evaluate the performance of our proposal, we deployed a set of analyzers and managers publishing and receiving IDMEF messages based on the *DARPA Intrusion Detection Evaluation Data Sets* [Lippmann et al., 2000]. This evaluation data set contains more than 300 instances of 38 different automated attacks that were launched against victim hosts in seven weeks of training data and two weeks of test data.

The complete set of messages were published as local and external alerts through the notification service of xmlBlaster, and then processed and republished in turn to the set of subscribed managers. The exchange of alerts proved to be satisfactory, obtaining a throughput performance higher than 150 messages per second on an Intel-Pentium M 1.4 GHz processor with 512 MB RAM, analyzers and managers on the same machine running Linux 2.6.8, using Java HotSpot Client VM 1.4.2 for the Java based broker. Message delivery did not become a bottleneck as all messages were processed in time and we never reached the saturation point.

The implementation of both publishers and subscribers was based on the *libidmef* C library [Migus, 2006] in order to build and parse compliant IDMEF messages. In turn, *libidmef* is built over the libxml library [Veillard, 2006]. The libxml library provides two interfaces to parser XML data: a DOM style tree interface, and a SAX style event based interface. Up to now, we are using for our implementation the DOM interface due to its easiness of use. Its main drawback is, however, that its memory usage is proportional to the size of the XML data. For this reason, we are actually moving our current implementation to the SAX based interface. This would help us to decrease the current amount of memory that is currently necessary to maintain the entire XML tree in memory.

The communication between analyzers and managers through xmlBlaster brokers was based on the xmlBlaster internal socket protocol and implemented using the xmlBlaster client C socket library [Ruff, 2006], which provides asynchronous callbacks to Java based brokers. The managers formulated their subscriptions using XPath expressions, filtering the messages they wished to receive from the broker.

In Figure 5.6 we show the processing time and memory space used by brokers during the exchange of alerts. The first curve represents the percentage of CPU load used by each broker. The second curve represents the quantity of memory used by each broker. As we can notice in the first curve, the percentage of processing time used by the brokers is quite stable and negligible for the normal performance of a normal system. The second curve reflects, however, that the cost in memory is quite intensive. We consider that this consumption is due to the managing of messages and we hope that the new version of our prototype based on a more efficient XML parsing and building scheme will decrease it.

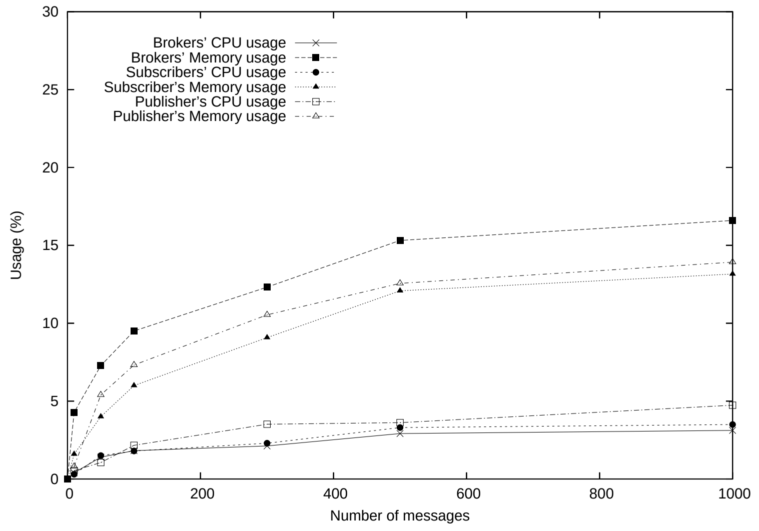


Figure 5.6: Processing and memory consumption.

Similarly, Figure 5.6 also shows the processing time and memory space used by publishers and subscribers. Here, we noticed again that the main drawback were the memory consumption due to the XML parsing and building. We consider this intensive use of memory will be decreased on the new version of our prototype based on the SAX based interface – avoiding the necessity of maintaining the entire XML tree of messages in both publishers and subscribers' memory.

Summary

We presented in this chapter a message passing design for the exchange of audit information between the components of our platform by means of a publish/subscribe model. Instead of having a central or master monitoring station to which all data has to be forwarded, there are independent uniform working entities at each host performing similar basic operations. The information gathered by each entity is exchanged to the rest through a publish/subscribe system which allows messages to be sent via a push or pull data exchange.

As we pointed out in Section 5.1, the advantage of this model for the exchange of audit information between components is, on the one hand, that it keeps the producer of messages separated from the consumer and, on the other hand, that the communication is information-driven. This way, it allows us to avoid problems regarding the scalability and the management inherent to other designs, by means of a network of publishers, brokers, and subscribers. A publisher in a publish/subscribe system does not need to have any knowledge about any of the entities that consume the published information. Likewise, the subscribers do not need to know anything about the publishers. Services can be added without any impact on or interruption of the service to other elements.

We then introduced in Section 5.2 the main properties of the Intrusion Detection Message Exchange Format (IDMEF) as the format that is built on top of the XML structure of the messages exchanged between the components of our platform; we presented in Section5.3 the operational details (interface operations and interaction) of our communication infrastructure; and we discussed and overviewed the initial results of a first prototype of our approach in Section 5.4. We consider these results give us a good hope that the use of a publish/subscribe system for the communication infrastructure indeed increases the scalability of the proposed architecture. Regarding the high memory usage, and as pointed out in Section 5.4, we are actually moving our current implementation to the SAX interface, since it does not maintain the entire XML tree in memory, which means that the load will considerably decrease.

Chapter 6

Anti-correlation and Selection of Counter-Measures

”What is the concept of defense? The parrying of a blow.
What is its characteristic feature? Awaiting the blow.”
– CARL VON CLAUSEWITZ (ON WAR)

The use of traditional security mechanisms, such as firewalls and cryptography, is not enough to guarantee the security of a given network system. Complementary mechanisms are necessary in order to cope with attacks when the first line of defense (i.e., cryptography and firewalls) has been evaded. Up to now, *Intrusion detection systems* (IDSs) have become the most important component of such a complementary kind of security mechanisms. They offer the necessary tools, methods, and resources to identify, assess, and report unauthorized activity against a target network.

A recent trend in intrusion detection is to try to understand and model intrusion strategies to provide a more global and precise diagnostic of the intrusion [Cuppens, 2001, Ning and Xu, 2003]. Although these approaches represent a step in the right direction, they are not sufficient. It is also necessary to develop automated defenses capable of appropriate responses to counter intrusions when they occur. Several response strategies are possible including launching counter measures against the intruder to prevent his or her malicious activity to proceed or acting on the target system to stop the intrusion and recover in a safe state [Gombault and Diop, 2002]. Direct responses against the intruder, however, is a complex problem that includes several technical difficulties.

In this chapter, we describe the main features of the reaction approach we presented in [Cuppens et al., 2006a]. This approach is intended to help security officers to choose proper counter-measures when an intrusion occurs. Such counter-measures generally depend on the type of intrusion being performed. For instance, the responses will not be the same in the case of a denial of service (DoS) attack or a user to root (U2R) attack. Thus, our approach is based on a library of responses that contains different types of possible counter-measures which the officer can launch to stop the detected intrusions.

Our approach is based on a logical formalization of both intrusions and counter-measures. This formalism is used to derive, from the intrusion description (specially the effects of an intrusion on the target system), one or several counter-measures that may circumvent the attack. For this purpose, we define in Section 6.2 the notion of *anti-correlation*. This notion is used to determine the counter-measures that will have a negative effect on the intrusion and therefore will enable the administrator to stop the attack.

The remainder of this chapter is organized as follows. In Section 6.1, we present our formalism to model intrusions and counter-measures. This formalism is based on LAMBDA, a language suggested in [Cuppens and Ortalo, 2000] to model intrusions. We then suggest in Section 6.1 how to use LAMBDA to model counter-measures. Section 6.2 recalls the definition of correlation presented in [Cuppens and Miège, 2002] and introduces the notion of anti-correlation. In Section 6.3, we show how to use anti-correlation to determine relevant counter-measures, either to act on the objective of an intrusion or to cut an ongoing attack scenario by acting on a given step of this scenario. We also present how our approach provides means to parameterize the selected counter-measures. Section 6.4 gives an example to illustrate the response mechanism suggested in this chapter.

6.1 Modeling Intrusions and Counter-Measures

In this section, we present our formalism, based on LAMBDA [Cuppens and Ortalo, 2000], to model both intrusions and counter-measures. LAMBDA is the acronym for LAnguage to Model a dataBase for Detection of Attacks. It is used to provide a logical description of an attack. The description of an attack specified by LAMBDA is generic, in the sense that it does not include elements specific to a particular intrusion detection process.

A LAMBDA description of an attack is composed of the following four attributes:

- **pre-condition –** defines the state of the system needed in to carry out the desired attack or action.
- **post-condition –** defines the state of the system after a successfully execution of the attack.
- **detection –** is a description of the expected alert corresponding to the detection of the attack.
- **verification –** specifies the conditions to verify the success of the attack.

The alerts launched by a detection system provide evidence of the occurrence of some malicious events but are not sufficient to conclude that these events will actually cause some damage to the target system.. This is why a LAMBDA description also includes a verification attribute that provides conditions to be checked to conclude that the execution of the action has been successful.

On the other hand, we define an *objective* as a specific system state. This state is characteristic of a violation of the security policy [Cuppens and Miège, 2002]. A LAMBDA description of an intrusion objective is composed by only one attribute:

- **state –** defines the state of the given system that corresponds to a security policy violation.

As explained above, LAMBDA is used to describe possible violations of security policy (intrusion objective) and possible actions (attack) an intruder can perform on a system to achieve an intrusion objective. This database of LAMBDA descriptions is used to recognize an intrusion process and predict the intention of the intruder.

Let us present here an example of intrusion modeled with LAMBDA. First an intruder scans port 139. If it is open, he concludes that the *operating system* is *windows* and uses the *NetBios* service. The intruder can then execute a *winnuke attack* on this target system that will cause a denial of service. We show in figures 6.1 and 6.2, respectively, the proper description in LAMBDA of the *port-scan* and *winnuke* attacks– performed by a malicious agent on a given host.

attack	$port\_scan(A, H, P)$	
pre:	$open(H, P)$	– port P is open on host H
detection:	$classification(Alert, 'TCP\_Scan')$	– the classification is $'TCP\_Scan'$
	$\wedge\ source(Alert, A)$	– the source in alert is A
	$\wedge\ target(Alert, H)$	– the target in alert is H
	$\wedge\ target\_service\_port(Alert, P)$	– the scanned port is P
post:	$knows(A, open(H, P))$	– agent A knows that port P is open
verification:	$true$	– always true

Figure 6.1: *port-can* attack performed by an agent A on a given host H.

attack	$winnuke(A, H, S)$	
pre:	$use\_os(H, windows)$	– OS on host H is Windows
	$\wedge\ use\_service(H, 'Netbios')$	– host H uses $'Netbios'$ service
	$\wedge\ open(H, 139)$	– port 139 is open on host H
detection:	$classification(Alert, 'Winnuke')$	– alert classification is $'Winnuke'$
	$\wedge\ source(Alert, A)$	– source in alert is A
	$\wedge\ target(Alert, H)$	– target in alert is H
post:	$deny\_of\_service(H)$	– deny of service on H
verification:	$unreachable(H)$	– host H does not reply

Figure 6.2: *winnuke* attack performed by an agent A on a given host H.

Let us also show how to model a violation of security policy. In Figure 6.3, for example, a policy violation after a web server goes down is presented.

objective	$webserver\_failure(H)$	
state:	$deny\_of\_service(H)$	– deny of service on host H
	$\wedge\ server(H, http)$	– host H is an http server

Figure 6.3: Intrusion objective: denial of service (DoS) on a web server.

As we can see in these three examples, each LAMBDA description uses several variables (corresponding to terms starting with an upper case letter). When an alert can be associated with a LAMBDA description through the *detection* attribute, we can then unify variables with values. We call *attack occurrence* a LAMBDA description where variables have been unified with values.

Using LAMBDA to Model Counter-Measures

In [Cuppens et al., 2006a] we suggest adopting the same formalism shown above to model counter-measures. Thus, a counter-measure has similar attributes to an attack. The main difference is that the *detection* attribute associated with an attack is replaced by the attribute *action*. This leads to the following model for representing counter-measures:

- **pre-condition –** defines the system state required for the success of the counter-measure.
- **post-condition –** defines the system state after applying the counter-measure.
- **action –** defines the actions necessary to perform the counter-measure.
- **verification –** specifies the conditions to verify the success of the counter-measure.

Figure 6.4 provides an example of counter-measure specified in this model. It consists in closing all connections between a given source S and a given target T.

counter-measure	$close\_remote\_access(S, T)$	
pre:	$remote\_access(S, T)$	– S has a remote access to T
action:	$TCP\_reset(S, T)$	– a TCP_reset closes the connection
post:	$not(remote\_access(S, T))$	– connections closed between both side
verification:	$not(TCP\_connection(S, T))$	– verify that all connections are closed

Figure 6.4: Counter-measure: closing a TCP connection.

As for the attacks and objectives, the approach is to use this formalism to specify a library of possible counter-measures that apply to the system to block an intrusion. We shall now define a response mechanism to select the adequate counter-measures for a detected scenario. This mechanism is based on a principle called *anti-correlation* which is close to the *correlation* principle suggested in [Cuppens and Miège, 2002]. These two principles are formally presented in the following section.

6.2 Correlation and Anti-correlation

Our response mechanism is based on recognizing the intruder's intentions. Using LAMBDA, [Cuppens and Miège, 2002] shows how to correlate detected attacks to identify a scenario. This approach was initially implemented in a PROLOG prototype called CRIM (CoopeRative Intrusion detection Module) and it has recently been improved in an enhanced version, implemented in C and C++, which also includes new features such as aggregating, merging, and weighted classification of correlated scenarios [Autrel, 2005].

In this section we first recall the definition of the correlation principle of both versions of CRIM. We also show how to extrapolate this definition to predict future attacks that the intruder will probably perform and the objective that he or she attempts to achieve. We then define a second notion, called *anti-correlation*, in order to formally design the response process, and we discuss how to use an anti-correlation process to select the counter-measures candidates as the response to the detected activity.

Correlation

Let so start by recalling the correlation approach as it was initially introduced for the first version of CRIM in [Cuppens and Miège, 2002]. Such a correlation process is based on the unification principle on predicates. Let a and b be two LAMBDA descriptions of attacks. $post_a$ is the set of literals of *post-condition* of the description a (i.e., $post_a = expr_{a1} \wedge expr_{a2} \wedge \ldots \wedge expr_{am}$) and pre_b is the set of literals of *pre-condition* of the description b (i.e., $pre_b = expr_{b1} \wedge expr_{b2} \wedge \ldots \wedge expr_{bn}$).

Direct correlation: *a and b are directly correlated if there exists $i \in [1, m]$ and $j \in [1, n]$ such that: $(expr_{ai} \wedge expr_{bj}) \vee ((not(expr_{ai})) \wedge (not(expr_{bj})))$ becomes $true$; and the literals $expr_{ai}$ and $expr_{bj}$ are unifiable through a most global unifier (mgu) θ.*

This definition of direct correlation represents the idea of positive influence between two attacks. We say that attack a has a positive influence over attack b if a is directly correlated to b. In such a case, the effects of a, namely the set of predicates in $post_a$, allows to satisfy a subset of the pre-requisites of pre_b. The notion of attack correlation allows us to find correlated attacks that are part of the same scenario.

In [Cuppens and Miège, 2002] it is also defined the notion of *Knowledge gathering correlation* as a variation of the above definition of correlation. This second notion is useful to integrate, in the detection process, preliminary steps the intruder performs to collect data on the target system.

Knowledge gathering correlation: *a and b are correlated by means of knowledge gathering if there exists $i \in [1, m]$ and $j \in [1, n]$ such that: $((knows(Agent, expr_{ai})) \wedge expr_{bj}) \vee ((knows(Agent, (not(expr_{ai}))) \wedge (not(expr_{bj})))$ becomes $true$; and the literals $expr_{ai}$ and $expr_{bj}$ are unifiable through a most global unifier (mgu) θ.*

This definition generally applies to the first steps of an intrusion. We say “generally” because an intruder may have no knowledge about the target machine. An intruder may directly try to exploit a vulnerability on a machine without trying to know if this security hole is present on the machine, but we argue that most of the time, the intruder will try to gather some information about the target. Hence the gathered knowledge may influence the attacker on the next attacks he will execute.

As an example, there is a knowledge gathering correlation between the Port-Scan attack (cf. Figure 6.1) and the *winnuke* attack (cf. Figure 6.2) through the predicate $open$ and the unifier that matches variable H in both attack definitions and variable P in the Port-Scan attack to constant 139. This means that an intruder who knows that port 139 is open on a given host, can then perform a *winnuke* attack on this host.

We now define the notion of correlation unifier that allows us to apply on-line correlation.

Correlation unifier: *denoted* Ξ_{ab}*, is the set of all possible unifiers – i.e. both direct or knowledge gathering correlation unifiers – to correlate* $post_a$ *and* pre_b*.*

Since two attacks a and b are correlated as soon as they have one predicate in common in $post_a$ and pre_b, we may have several unifiers for two attacks. The set of correlation unifiers allows us to know which attack can be correlated with a given attack under some unification condition between their variables. Applying on-line correlation consists in exploring the set of correlation unifiers each time a new alert is received, given that the alert corresponds to an instance of an attack model.

We can apply the notion of direct correlation between two attacks to an intrusion objective and an attack. This allows us to detect that some attack may allow to reach or help to reach an intrusion objective. In this case, we simply have to substitute the term *pre-condition* by *state* in the definition of direct correlation.

How to use correlation

Once attacks and intrusion objectives are specified in LAMBDA, we can generate all correlation unifiers between each pair of attacks (respectively between an attack and an intrusion objective). When two attack occurrences are detected, if some unifier in the unifier set is identified, we can then say that these attack occurrences are correlated in the same intrusion scenario. Using this approach, it is possible to build a correlation graph. Figure 6.5

presents such a correlation graph where nodes are LAMBDA descriptions and edges are correlation unifiers.

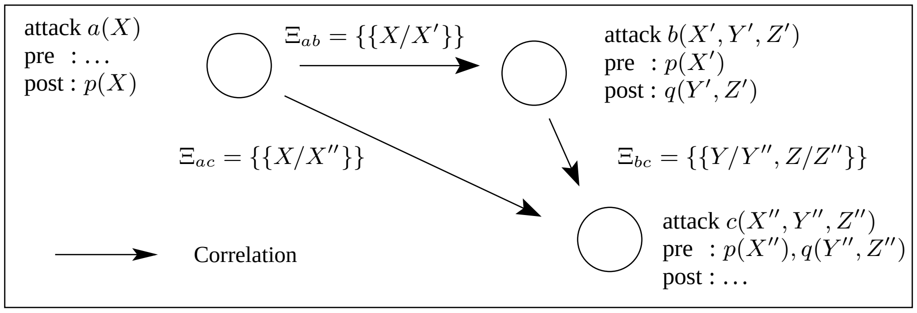

attack $a(X)$
pre : …
post : $p(X)$
$\Xi_{ab} = \{\{X/X'\}\}$
attack $b(X', Y', Z')$
pre : $p(X')$
post : $q(Y', Z')$
$\Xi_{ac} = \{\{X/X''\}\}$
$\Xi_{bc} = \{\{Y/Y'', Z/Z''\}\}$
attack $c(X'', Y'', Z'')$
pre : $p(X''), q(Y'', Z'')$
post : …
Correlation

Figure 6.5: Correlation graph example.

When first steps of a given intrusion scenario are identified, we can, with the same mechanisms, predict possible continuations of this scenario. We can generate hypothesis about future attacks and the intrusion objectives the intruder attempts to achieve. We shall call *virtual attack* an attack predicted by this process of intention recognition. A virtual attack becomes effective once its occurrence is detected.

Thus, it is sometimes possible to anticipate on the actions performed by the intruder and develop a specific counter-measure in response. This means that our approach may be used to launch a counter-measure not only after a given intrusion objective is achieved by the intruder but also when the beginning of a given scenario is detected. In this latter case, the counter-measure will be used to prevent continuations of this starting scenario.

We show now in the following section how to define and use the anti-correlation principle to elaborate the counter-measures. This new feature was recently included in a new version of the module CRIM, presented in [Autrel, 2005], which also includes the previous features for correlation and hypothesis generation.

Anti-correlation

Let a and b be respectively LAMBDA descriptions of a counter-measure and an attack. $post_a$ is the set of literals of *post-condition* of description a (i.e., $post_a = expr_{a1} \wedge expr_{a2} \wedge \ldots \wedge expr_{am}$) and pre_b is the set of literals of *pre-condition* of description b (i.e., $pre_b = expr_{b1} \wedge expr_{b2} \wedge \ldots \wedge expr_{bn}$).

Anti-correlation: *the descriptions a and b are anti-correlated if there exists $i \in [1, m]$ and $j \in [1, n]$ such that: $(expr_{ai} \wedge (not(expr_{bj}))) \vee ((not(expr_{ai})) \wedge expr_{bj})$ becomes $true$; and the literals $expr_{ai}$ and $expr_{bj}$ are unifiable through a most global unifier (mgu) θ.*

This definition formalizes the notion of negative impact of a counter-measure over an intrusion scenario. A counter-measure is an action which prevents the execution of an attack. Since our model of an attack includes the necessary conditions the system's state must meet in order to execute the attack, we can prevent the execution of an attack by making one of those conditions false.

Therefore, a counter-measure c for an attack a is a LAMBDA model of which the post-condition contains a predicate that contradicts one predicate of pre_a. We can then say that c is anti-correlated with a. Even though it may be sufficient for a counter-measure to anti-correlate an attack through only one predicate, it is also possible for a counter-measure to anti-correlate an attack through several predicates.

Anti-correlation unifier: denoted Ψ_{ab}, is the set of all unifiers θ possible to anti-correlate $post_a$ and pre_b.

As for a correlation unifier, an anti-correlation unifier defines which attacks can be anti-correlated by a counter-measure. It tells how the variables must be unified in the predicates which are involved in the anti-correlation link. Using the same approach, it is possible to define anti-correlation between a counter-measure and an intrusion objective. We have simply to replace *pre-condition* by *state* in the previous definition.

6.3 Using Anti-correlation for Response

When a scenario is identified, the correlation process provides a graph of attack occurrences, virtual attacks and intrusion objective. A counter-measure will apply to invalidate future attacks or invalidate an intrusion objective. Thus, we have two response mechanisms, one that applies against virtual attacks and the other on an intrusion objective.

Response to an Intrusion Objective

In this case, response aims at updating the system state to invalidate the intrusion objective in an intrusion scenario.

Let o be an intrusion objective. To invalidate this intrusion objective, we must find a LAMBDA definition r of a counter-measure such that $\Psi_{ro} \neq \emptyset$. Then, it is possible to parameterize this counter-measure candidate with the unifier of correlation Ψ_{ro}.

In the following figure, for example, we assume that two occurrences of attack are detected: an occurrence of a with argument $X = x$ and an occurrence of b with argument $Y = y$.

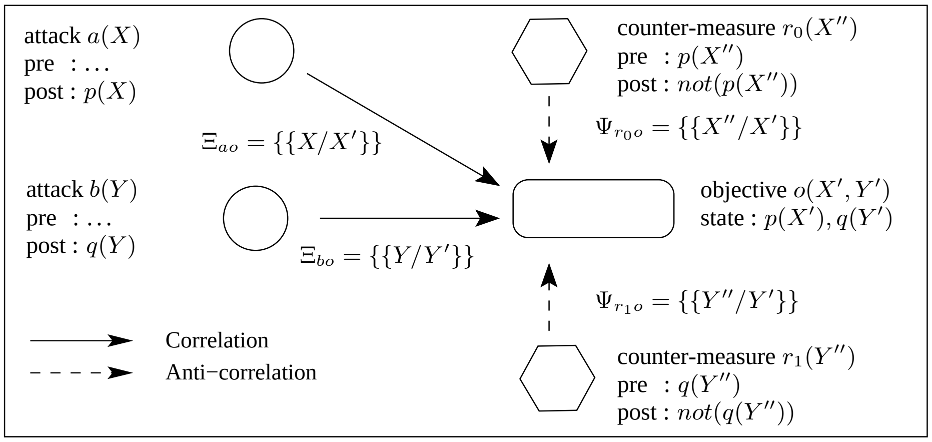


Figure 6.6: Correlation graph with direct response on the objective.

The correlation process shown in Figure 6.6 diagnoses that the two attacks a and b are correlated with a given intrusion objective o, and that this objective has been achieved. The response process then finds two counter-measure candidates.

The first counter-measure, r_0, with the parameter $X'' = X'$, and provided by $\Psi_{r_0,o}$, suggests the possibility to invalidate condition p (and thus objective o). The second countermeasure, r_1, with parameter $Y'' = Y'$, and provided by $\Psi_{r_1,o}$, suggests the possibility to invalidate the condition q (and thus also the objective o).

By combining $\Xi_{a,o}$ with $\Psi_{r_0,o}$, we can derive that counter-measure r_0 may apply with parameter $X'' = X' = X = x$ and similarly counter-measure r_1 may apply with parameter $Y'' = y$. Thus this provides means to derive which parameters must be selected when applying the counter-measure. In our approach, these two counter-measures are suggested to the administrator who can select one of them (or both).

Finally, the *verification* field of the selected counter-measure is then evaluated to check whether the counter-measure was executed successfully. If this is the case, we can reevaluate the state condition of the intrusion objective to false.

Response to an Ongoing Scenario

It is possible that a counter-measure may not apply directly to an intrusion objective if one of these conditions holds:

- There is not any counter-measure in the response library which may apply to invalidate the intrusion objective.
- The counter-measure does not apply to the system state because the pre-condition of this counter-measure is evaluated to false.
- All counter-measure candidates were launched without success.

In these cases, a possible solution is to modify the system state to invalidate one attack in a sequence of virtual attacks. When the correlation engine receives a new alert, it tries to find a path of correlated virtual attacks leading to one (or more) intrusion objective(s).

The path of virtual attacks and the intrusion objective found represents a possible evolution of the ongoing scenario. Let $a_1 ... a_n$ be a sequence of virtual attacks and o an intrusion objective such that for every $i \in [1, n-1]$, a_i is correlated with a_{i+1} and a_n is correlated with o. The affirmation stating that for every $i \in [1, n-1]$, a_i is correlated with a_{i+1} and a_n is correlated with o is not necessarily true for the entire set of virtual attack generated. But we can always find a subset of virtual attack satisfying this condition in the set of generated virtual attacks, given that the set of generated virtual attacks leads to an intrusion objective.

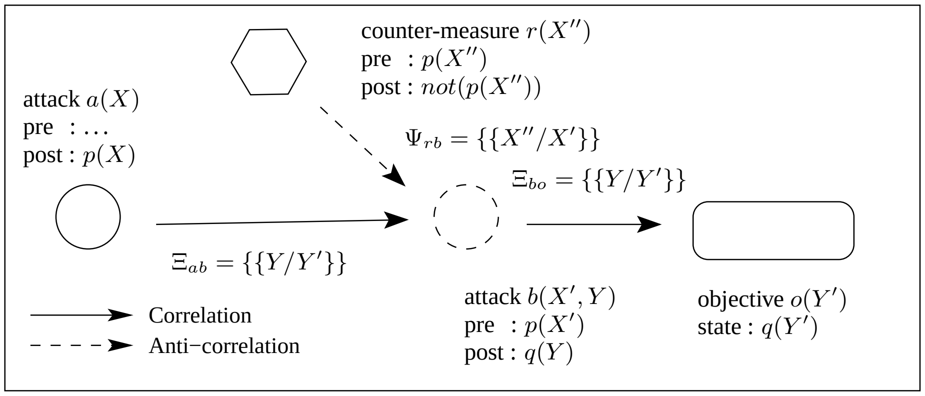


Figure 6.7: Correlation graph with response on a sequence of virtual attacks.

To block this sequence of attacks, we must find a valid LAMBDA counter-measure r such that r is anti-correlated with one of the attacks a_k ($k \in [1, n]$). For instance, let us assume, in Figure 6.7, that we detect an occurrence of a. The recognizing intention process identifies that the intruder may perform b after a to achieve the objective o. In this case, the response process can find a counter-measure r to invalidate the *pre-condition* of b. This will prevent performance of attack b and invalidate this scenario.

6.4 Reacting on a Sample Attack Scenario

This section presents the use of the response mechanism introduced in this chapter, together with the rest of components overviewed in previous chapters, and the enhanced version of CRIM [Cuppens and Miège, 2002] presented in [Autrel, 2005]. To illustrate the use of our approach, let us consider the Mitnick attack as defined in [Northcutt, 2002]. This attack tries to exploit the trust relationship between two computers to achieve an illegal remote access using the coordination of three techniques. First, a SYN flooding DoS attack to keep the trusted system from being able to transmit. Second, a TCP sequence prediction against the target system to obtain its following TCP sequence numbers. And third, an unauthorized remote shell by spoofing the IP address of the trusted system (while it is in a mute state) and using the sequence number that the target system is expecting. The correlation graph for this attack is presented in the following figure:

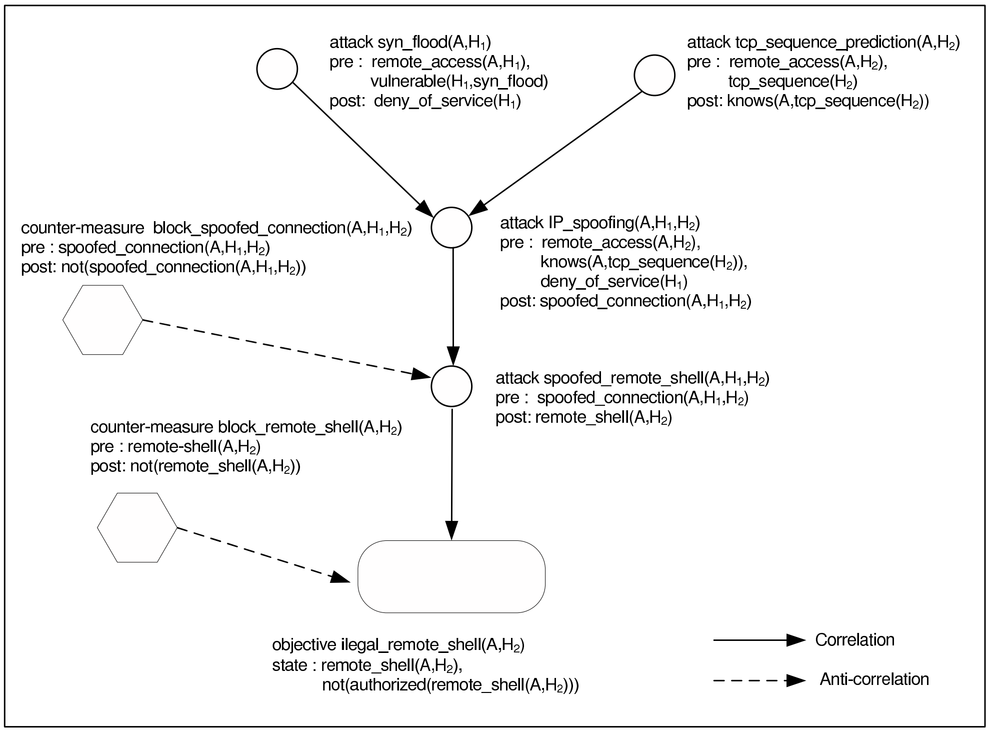

attack syn_flood(A,H1)
pre : remote_access(A,H1), vulnerable(H1,syn_flood)
post: deny_of_service(H1)
attack tcp_sequence_prediction(A,H2)
pre : remote_access(A,H2), tcp_sequence(H2)
post: knows(A,tcp_sequence(H2))
counter-measure block_spoofed_connection(A,H1,H2)
pre : spoofed_connection(A,H1,H2)
post: not(spoofed_connection(A,H1,H2))
attack IP_spoofing(A,H1,H2)
pre : remote_access(A,H2), knows(A,tcp_sequence(H2)), deny_of_service(H1)
post: spoofed_connection(A,H1,H2)
attack spoofed_remote_shell(A,H1,H2)
pre : spoofed_connection(A,H1,H2)
post: remote_shell(A,H2)
counter-measure block_remote_shell(A,H2)
pre : remote-shell(A,H2)
post: not(remote_shell(A,H2))
objective ilegal_remote_shell(A,H2)
state : remote_shell(A,H2), not(authorized(remote_shell(A,H2)))
Correlation
Anti-correlation

Figure 6.8: Correlation graph for the Mitnick attack scenario.

attack	$syn\_flood(A, H)$	
pre:	$remote\_access(A, H)$	– attacker has a remote access on the target
	$\wedge\ vulnerable(H,'syn\_flood')$	– target is vulnerable to $'syn\_flood'$ attack
detection:	$classification(Alert,'syn\_flood')$	– the alert classification is $'syn\_flood'$
	$\wedge\ source(Alert, A)$	– the source in alert is agent A
	$\wedge\ target(Alert, H)$	– the target in alert is host H
post:	$deny\_of\_service(H)$	– deny of service on host H
verification:	$unreachable(H)$	– host H does not reply
attack	$tcp\_sequence\_prediction(A, H)$	
pre:	$remote\_access(A, H)$	– attacker has a remote access on the target
	$\wedge\ tcp\_sequence(H)$	– the TCP sequence of H is predictable
detection:	$classification(Alert,'TCP\_seq\_prediction')$	– the alert classification is $'TCP\_seq\_prediction'$
	$\wedge\ source(Alert, A)$	– the source in alert is agent A
	$\wedge\ target(Alert, H)$	– the target in alert is host H
post:	$knows(A, tcp\_sequence(H))$	– attacker knows the TCP sequence of target
verification:	$true$	– always true
attack	$IP\_spoofing(A, H_1, H_2)$	
pre:	$remote\_access(A, H_2)$	– attacker has a remote access on H_2
	$\wedge\ knows(A, tcp\_sequence(H_2))$	– attacker knows the TCP sequence of H_2
	$\wedge\ deny\_of\_service(H_1)$	– deny of service on host H_1
detection:	$classification(Alert,'IP\_spoofing')$	– the alert classification is $'IP\_spoofing'$
	$\wedge\ source(Alert, A)$	– the source in alert is agent A
	$\wedge\ source(Alert, spoofed)$	– the source is spoofed
	$\wedge\ additional\_data(Alert, spoofed\_addr, H_1)$	– the spoofed address is H_1
	$\wedge\ target(Alert, H_2)$	– the target in alert is host H_2
post:	$spoofed\_connection(A, H_1, H_2)$	– attacker has a spoofed connection on H_2 as H_1
verification:	$unreachable(H_1)$	– host H_1 does not reply
counter-measure	$block\_spoofed\_connection(A, H_1, H_2)$	
pre:	$spoofed\_connection(A, H_1, H_2)$	– attacker has a spoofed connection on H_2 as H_1
action:	$block\_IP\_datagrams(A, H_1, H_2)$	– a packet filtering blocks the connection
post:	$not(spoofed\_connection(A, H_1, H_2))$	– spoofed connections blocked between both sides
verification:	$not(TCP\_spoofed\_connection(A, H_1, H_2))$	– verify that spoofed connections are blocked
attack	$spoofed\_remote\_shell(A, H_1, H_2)$	
pre:	$spoofed\_connection(A, H_1, H_2)$	– attacker has a spoofed connection on H_2 as H_1
detection:	$classification(Alert,'spoofed\_rshell')$	– the alert classification is $'spoofed\_rshell'$
	$\wedge\ source(Alert, A)$	– the source in alert is agent A
	$\wedge\ source(Alert, spoofed)$	– the source is spoofed
	$\wedge\ additional\_data(Alert, spoofed\_addr, H_1)$	– the spoofed address is H_1
	$\wedge\ target(Alert, H_2)$	– the target in alert is host H_2
post:	$remote\_shell(A, H_2)$	– attacker has a remote shell on host H_2
verification:	$unreachable(H_1)$	– host H_1 does not reply
counter-measure	$block\_remote\_shell(A, H)$	
pre:	$remote\_shell(A, H)$	– attacker has a remote shell on the target
action:	$block\_IP\_datagrams(A, H)$	– a packet filtering blocks the connection
post:	$not(remote\_shell(A, H))$	– remote shell between A and H_2 is blocked
verification:	$not(remote\_shell\_traffic(A, H))$	– verify that the remote shell connection is blocked
objective	$illegal\_remote\_shell(A, H)$	
state:	$remote\_shell(A, H)$	– attacker has a remote shell on host H
	$\wedge\ not(authorized(remote\_shell(A, H)))$	– attacker is not authorized to have this remote shell

Figure 6.9: LAMBDA models of the Mitnick attack scenario.

The LAMBDA models for each attack that composes the whole scenario, together with the intrusion objective, are shown in Figure 6.9. In the first step, *A* (the agent that performs the whole attack) floods a given host H_1. In the second step, *A* sends a TCP sequence prediction attack against host H_2 to obtain its following TCP sequence numbers. Then, by using these TCP sequence numbers, *A* starts a spoofed remote shell session to the host H_2 as it would come from host H_1. Since H_1 is in a mute state, H_2 will not receive the RST packet to close this connection. If the third and fourth steps are successful, *A* will establish an illegal remote shell session to system H_2. The model of Figure 6.9 also proposes two counter-measures to prevent the Mitnick attack. First, a counter-measure can apply before the intrusion objective is achieved. This counter-measure, called *block_spoofed_connection*, blocks the spoofed connection between the host and the intruder. Second, a counter-measure can apply directly on the intrusion objective. This counter-measure, called *block_remote_shell*, blocks the remote shell connection between the host and the intruder.

To show how the components of our architecture would handle the attack, let us consider the sequence of alerts described in Figure 6.10. There, we assume that an intruder targeting the network `victim.org` will use resources from another network to perform the attack. We also assume that this network is protected by a set of *firewalls*, *network intrusion detection systems* (NIDSs), the policy manager presented in Chapter 4, and an instance of CRIM on each node of the network. Moreover, those components are exchanging IDMEF messages through the notification service presented in Chapter 5. To simplify, we consider that the different parts of the attack are only detected by three different nodes, named *node1*, *node2*, and *node3*. For each node, we show in Figure 6.10 just such relevant messages published and notified within the system. We have also simplified quite a lot the information and format of each alert for clarity reasons. Each alert is so denoted with ordered identifiers t_i, which correspond to the *DetectionTime* field of each IDMEF message.

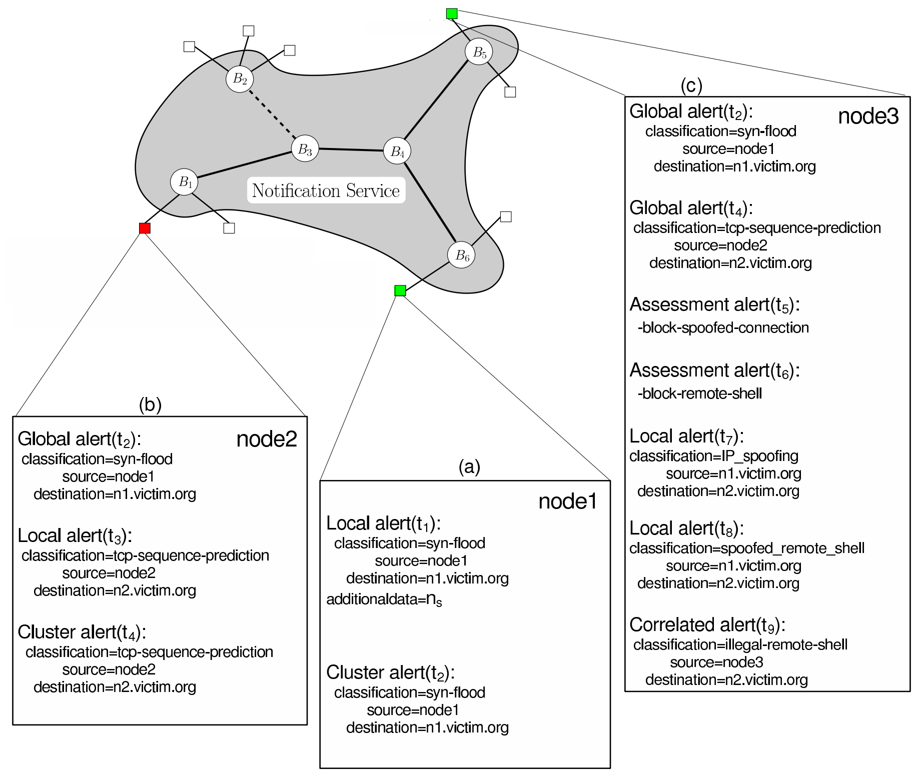
When alerts corresponding to different steps of the Mitnick attack scenario are raised within the complete system, the instances of CRIM installed on each node will apply the correlation mechanism to recognize the global scenario. For example, when the alerts corresponding to the attack $syn\_flood(A, H_1)$ and the attack $tcp\_sequence\_prediction(A, H_2)$

are raised, respectively, *node1* and *node2* (cf. cluster alerts t_2 and t_4 in Figure 6.10) the correlation mechanism of CRIM on *node3*, which receives those *cluster alerts* published by the other two nodes as *global alerts*, will generate the virtual alerts corresponding to the attacks $IP\_spoofing(A, H_1, H_2)$ and $spoofed\_remote\_shell(A, H_1, H_2)$, and the objective $illegal\_remote\_shell(A, H_2)$. The correlation engine of CRIM recognizes the whole scenario since the post-condition $deny\_of\_service(H_1)$ of the attack $syn\_flood(A, H_1)$ is correlated with the pre-condition $deny\_of\_service(H_1)$ of $IP\_spoofing(A, H_1, H_2)$, and the post-condition $knows(A, tcp\_sequence(H))$ of $tcp\_sequence\_prediction(A, H)$ is correlated with the pre-condition $knows(A, tcp\_sequence(H_2))$ of $IP\_spoofing(A, H_1, H_2)$.

Once this diagnosis is processed, the response module of CRIM installed on *node3* can select the two possible counter-measures discussed above. The counter-measure *block_spoofed_connection* is chosen since its post-condition $not(spoofed\_connection(A, H_1, H_2))$ is anti-correlated with the pre-condition $spoofed\_connection(A, H_1, H_2)$ from the attack $spoofed\_remote\_shell(A, H_1, H_2)$. Likewise, the counter-measure *block_remote_shell* is chosen since its predicate $not(remote\_shell(A, H_2))$ is then anti-correlated with the predicate $remote\_shell(A, H_2)$ of the objective $illegal\_remote\_shell(A, H_2)$.

At this point, the response module of CRIM on *node3* generates the *assessment alerts* t_5 and t_6 with the actions specified on counter-measures *block_spoofed_connection* and *block_remote_shell*. Those two counter-measures are then notified to the policy manager as IDMEF assessment alerts, which provides the security officer with the actions showed in Figure 6.9. Thus, the administrator can select one of those actions, in order to reconfigure, for example, the security policy, before the attacks $IP\_spoofing(A, H_1, H_2)$ and $spoofed\_remote\_shell(A, H_1, H_2)$ will be performed, i.e., before the intrusion objective $illegal\_remote\_shell(A, H_2)$ will be reached.

As we will further discuss in Chapter 8, we only use this response approach to provide a support to the administrator who takes the final decision to choose and launch a given response. We do not consider, expressly, a proactive response since it could, in certain cases, be leveraged by the attacker to damage the system itself or to cause a denial of service to authorized users.



(a) node1

Local alert(t_1):
classification=syn-flood
source=node1
destination=n1.victim.org
additionaldata=n_s

Cluster alert(t_2):
classification=syn-flood
source=node1
destination=n1.victim.org

(b) node2

Global alert(t_2):
classification=syn-flood
source=node1
destination=n1.victim.org

Local alert(t_3):
classification=tcp-sequence-prediction
source=node2
destination=n2.victim.org

Cluster alert(t_4):
classification=tcp-sequence-prediction
source=node2
destination=n2.victim.org

(c) node3

Global alert(t_2):
classification=syn-flood
source=node1
destination=n1.victim.org

Global alert(t_4):
classification=tcp-sequence-prediction
source=node2
destination=n2.victim.org

Assessment alert(t_5):
-block-spoofed-connection

Assessment alert(t_6):
-block-remote-shell

Local alert(t_7):
classification=IP_spoofing
source=n1.victim.org
destination=n2.victim.org

Local alert(t_8):
classification=spoofed_remote_shell
source=n1.victim.org
destination=n2.victim.org

Correlated alert(t_9):
classification=illegal-remote-shell
source=node3
destination=n2.victim.org

Figure 6.10: Sequence of alerts raised inside each node during the attack.

Let us finally suppose, to conclude our example, that the security officer does not launch neither the counter-measure *block_spoofed_connection* nor the counter-measure *block_remote_shell*. In this case, we can assume that the two local alerts t_7 and t_8 may be detected by the components installed on *node3*, leading the correlation engine of CRIM to determine that the intrusion objective $illegal\_remote\_shell(A, H_2)$ has been reached. It may manage such a conclusion since the pre-condition $spoofed\_connection(A, H_1, H_2)$ of the attack $spoofed\_remote\_shell(A, H_1, H_2)$ is correlated to the post-condition $spoofed\_connection(A, H_1, H_2)$ of $IP\_spoofing(A, H_1, H_2)$, the pre-condition $knows(A, tcp\_sequence(H_2))$ of $IP\_spoofing(A, H_1, H_2)$ is correlated with the post-condition $knows(A, tcp\_sequence(H_2))$ of $tcp\_sequence\_prediction(A, H_2)$, and the intrusion objective state $remote\_shell(A, H_2)$ is then correlated with the post-condition $remote\_shell(A, H_2)$ of the attack $spoofed\_remote\_shell(A, H_1, H_2)$.

Summary

In this chapter we have presented a response mechanism to select and apply counter-measures when an intrusion attack is detected. We first presented in Section 6.1 LAMBDA, an algebraic language that allows us to provide a logical description of system's actions; we then discussed how to use such a language to model both attacks and counter-measures; and we finally provided some examples of both intrusions actions and counter-measures expressed in this language.

In Section 6.2 we showed out how the correlation approach implemented in CRIM, a module for management of intrusion alerts. It uses LAMBDA to correlate detected attacks, and to identify a scenario that leads from an initial state to a final intrusion objective – where the security policy has been violated. We also showed in Section 6.2 how to use the semi-explicit correlation approach and discussed how we can use a generation of hypothesis to anticipate the occurrence of a possible complex intrusion scenario before this scenario will be successful. This last feature allows us to anticipate on the actions performed by the intruder and develop a specific response for a given scenario.

We then defined in Section 6.3 the notion of anti-correlation and discussed some examples to show how we can use this feature to determine relevant counter-measures, either to act on the objective of an intrusion or to stop an ongoing attack scenario by acting on a given step of this scenario. In Section 6.4 we offered an extended example to illustrate the use of the anti-correlation approach, together with the rest of features of CRIM. We discussed the detection of a complete attack, and the sequence of counter-measures offered by our approach in order to respond to different steps of the attack. These counter-measures are intended to help the security officer to decide which appropriate actions may be launched in order to terminate or to respond to the given attack. This is a prudent strategy that may only be used to provide a support to the administrator who takes the final decision to choose and launch a given response. We do not consider up to now a real time response. More investigation has to be done before to extend our approach in that way.

Chapter 7

Protection of Components based on a Kernel Security Module

”They don’t advertise for killers in the newspaper. That was my profession. Ex-cop. Ex-blade runner. Ex-killer.”
– RICK DECKARD (BLADE RUNNER)

Contrary to many other elements of a network, security components, such as firewalls and network intrusion detection systems (NIDSs), are almost always working with special privileges to properly execute their tasks. This situation is very likely to lead remote attackers to acquire these privileges in an unauthorized manner. Once acquired, the attacker can manage to compromise those elements, or even to obtain the control of the system itself. The existence of programming errors within the code of the component’s elements, the illicit manipulation of their related resources (such as processes, configuration files, log files, and so on), or even the increase of privileges though operating system’s errors, are just a few examples regarding means in which an attacker can bypass traditional security policy controls. The protection of these elements is a serious and important problem which must be solved. In this chapter, we overview a preventive mechanism which is intended for the protection of the network security components of our framework. Our proposal consists of a kernel access control scheme which handles the protection of each security component and its elements (i.e., processes, files, and so on), and which intercepts and cancels forbidden system calls launch by a remote intruder.

The remainder of this chapter is organized as follows. Section 7.1 gives an outlook to our protection strategy in order to protect the critical resources processes with special privileges. Section 7.2 takes a closer look at the development of the proposed mechanism. Section 7.3 presents a smart-card based authentication protocol intended to solve the administration constraints introduced by our protection mechanism. Finally, the configuration of our proposal and an evaluation concerning the efficiency and security of its implementation is presented in Section 7.4.

7.1 Kernel based Control of System Calls

As we introduced above, our main motivation is the protection of the network security components of our platform, such as *firewalls* and *network intrusion detection systems* (NIDSs), which, if successfully attacked, are very likely to lead an intruder to get the control of the whole system. This problem leads to the necessity for introducing a protection mechanism on the different elements of each component, keeping with their protection and mitigating – or even eliminating – any attempt to attack or compromise the component's elements and their operations. This way, even if an attacker compromises the security of the component, he would not be able to achieve his purpose.

According to [Onabuta et al., 2001], we consider the protection of the elements carried by the kernel of the operating system as a proper solution for such a protection.

On the first hand, the protection at kernel level avoids that potentially dangerous system calls (e.g., *killing* a process) could be produced from one element against another one. This protection is achieved by incorporating an access control mechanism into the kernel system calls. This way, one may allow or deny a system call based on several criteria – such as the identifier of the process making the call, some parameters of the given call, and so on. The kernel's access control allows to eliminate the notion of trust associated to privileged users, delegating the authorization for the execution of a given system call to the internal access control mechanisms. In addition, and contrary to other approaches, it provides a unified solution, avoiding the implementation of different specific mechanisms for each component.

On the other hand, this mechanism allows us to enforce the compartmentalization principle [Viega and McGraw, 2002]. This principle is based in the segmentation of a system, so several elements can be protected independently one from another. This ensures that even if one of the elements is compromised, the rest of them can operate in a trusted way.

In our case, several elements from each component are executed as processes. By specifying the proper permission based on the process ID, we can limit the interaction between these elements of the component. If an intruder takes control of a process associated to a given component (through a buffer overflow, for example), he will be limited to make the system call for this given process.

It is not always possible, however, to achieve a complete independence between the elements. There is a need to determine which system calls may be considered as a threat when launched against an element from the component. This requires a meticulous study of each one of the system calls provided by the kernel, and how they can be misused. On the other hand, we have to define the access control rules for each one of these system calls. For our approach, we propose the following three protection levels to classify the system calls: (1) critical process protection; (2) communication mechanisms protection; and (3) protection of files associated to the elements.

The first level of protection (critical processes) comprises actions that can cancel the proper execution of the processes associated to a component, either by interaction over them by signals, or the manipulation of the memory space. Some examples are: execution of a new application already in memory, cancellation or manipulation of the address space and process traces, and so on.

The second level (communication mechanisms protection) includes the protection of all those processes that allows an attacker to modify, generate or eliminate any kind of messages exchanged between component's elements. Finally, the third level of protection (protection of files associated to the elements) takes into account all those actions that can maliciously address the set of files used by the elements of the component, such as executable, configuration, or log files.

7.2 Implementation of a Linux Security Module

In this section we outline the current implementation of our approach in a research prototype called SMARTCOP (which stands for *Smart Card Enhanced Linux Security Module for Component Protection*). It consists of a kernel based access control mechanism, and its development has been done over the *Linux Security Modules* (LSM) framework for *GNU/Linux* systems [Wright et al., 2002].

The LSM framework does not consist of a single specific access control mechanism; instead it provides a generic framework, which can accommodate several approaches. It supplies several hooks (i.e., interception points) across the kernel that can be used to implement different access control strategies. Such hooks are: *Task hooks*, *Program Loading Hooks*, *File systems Hooks* and *Network hooks*.

These LSM hooks, can be used to provide protection at the three levels commented in the previous section. Furthermore, LSM adds a set of benefits to our implementation: first, it introduces a minimum load to the system when comparing it to kernels without LSM, and does not interfere with the detection and reaction processes (cf. Section 7.4); second, the access control mechanism can be composed in the system as a module, without having to recompile the kernel; and third, it provides a high degree of flexibility and portability to our implementation when compared to other proposals for the Linux kernel, such as [Onabuta et al., 2001] and [Ott, 2002], where the implementation requires the modification of some features of the original operating system's kernel.

The LSM interface provides an abstraction, which allows the modules to mediate between the users and the internal objects from the operating system's kernel. To this effect, before accessing the internal object, the hook calls the function provided by the module and which will be responsible to allow or deny the access. In Figure 7.1 we show how a module may register one function just to make a validation over the *i-nodes* of the filesystem.

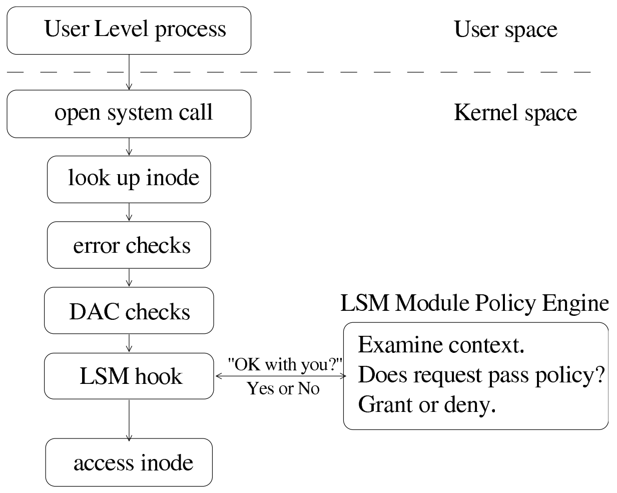


Figure 7.1: Linux Security Modules (LSM) Hooks.

At the same time, LSM allows us to keep the *discretionary access control* (DAC) provided by the operating system's kernel, by standing between the discretionary control and the object itself. This way, if a user does not have permissions in relation to a given file, the DAC of the operating system will not allow the access and no call to the function registered by the LSM will be made. This feature reduces the load of the system when compared to an access control check centralized in the operating system call interface, which always gets used for all the system calls.

The component's elements will be allowed to make operations only permitted to the system administrator – such as packet filtering and process or application cancellation. This implies that the system processes associated to each element will be executed by the administrator – i.e., root user in Unix systems. On the contrary, if we associate the processes to a non privileged user, the discretionary access control of the operating system's kernel will not allow the execution of some specific calls.

The internal access control mechanisms at the kernel is based in the process identifier (PID) that makes the system call, which will be associated to a specific element. Each function registered by a LSM module, determines which component is making the call from the PID of the associated process. It then, applies the access control constraints taking also into account the parameters of the system call. Thus, for example, a given element can access its own configuration files but not configuration files from other elements. In Figure 7.2, for example, we show how a function registered by our LSM protection module may allow or deny the modification of a configuration file.

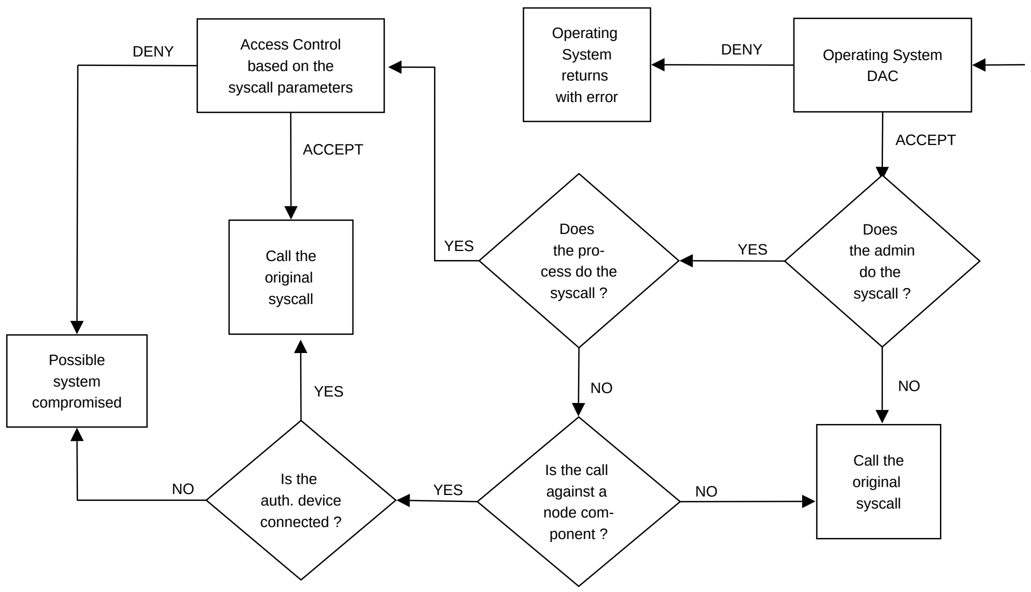

Access Control based on the syscall parameters
DENY
ACCEPT
Operating System returns with error
DENY
Operating System DAC
ACCEPT
Call the original syscall
YES
Does the process do the syscall ?
YES
Does the admin do the syscall ?
Possible system compromised
YES
NO
NO
Call the original syscall
NO
Is the auth. device connected ?
YES
Is the call against a node component ?
NO

Figure 7.2: Access control example through our protection module.

An important issue in the implementation is the administration of the access control mechanisms and the management of each one of the elements. As pointed out in previous sections, the administrators should not be able to throw a system call, which may suppose a threat to the component. This prevents an intruder doing any harm to the component even if he could scale his privileges to the administrator ones. This contrasts with the administration of the component, since, if an administrator can not interact with the elements of the component, he will not be able to carry out any management or configuration process and activities.

To solve this hazard, we propose a smart-card based authentication mechanism. Specifically, we use the functionality of a smart-card for ensuring the administrator's identity. Through the use of an authentication protocol, the LSM module verifies administrator's actions before holding him the indispensable privileges to manipulate the component. Otherwise, the access control enforcement will come to its normal operation. In the following section, a detailed description of such a mechanism is given.

7.3 Smart-card based Authentication Mechanism

In order to better assure the administrator's identity of SMARTCOP, we propose the use of a two-factor authentication mechanism based on the cryptographic functions of a smart-card. This mechanism is intended for authenticating the administrator to the LSM modules and holds with the following requirements:

- The actions must be authorized by the use of a smart-card;
- The smart-card only authorizes one action iff the PIN is correct;
- The LSM module only authorizes the action iff the smart-card response is valid.

Authentication Protocol Description

In this section we give a first pair of cryptographic protocols that lead our smart-card based authentication mechanism. Let us recall that the cryptographic engine of such a smart-card is capable of performing several cryptographic functions, such as symmetric key generation, symmetric cryptographic algorithms execution, and so on.

A first protocol is defined as follows.

Protocol 1

1. *The system administrator opens a new console and he requests an action X. It is assumed that X must be authorized by using the smart-card;*
2. *The LSM module receives the request from the console and it does the following steps:*
   (a) *Open a connection to the smart-card reader device, and protect the channel between the smart-card reader device and the LSM module itself to avoid being tampered or sniffed by any other process;*
   (b) *Print a message in the console, asking for the smart-card insertion to the smart-card reader device;*
   (c) *While the smart-card has not been inserted do;*
      i. *Detect the insertion of the smart-card;*
   (d) *Print a message in the console asking for the operation PIN;*
3. *The system administrator types the operation PIN in the keyboard.*
4. *The LSM module does the following steps:*
   (a) *Obtain the operation PIN;*
   (b) *Obtain a NONCE value at random;*
   (c) *Compute the Message Authentication Code (MAC) of NONCE with the shared key K, $\mu_1 = MAC(K, NONCE)$;*
   (d) *Execute the Procedure 1 inside the smart-card using the operation PIN and the NONCE, and obtain a response μ_2;*
   (e) *Print a message in the console to remove the smart-card from the smart-card reader device;*

(f) *While the smart-card has not been removed do;*
   i. *Detect the removing of the smart-card;*

(g) *if μ_2 is* ERROR *the LSM module does not authorize the action X;*

(h) *else do:*
   i. *if $\mu_1 \neq \mu_2$ the LSM module does not authorize the action X;*
   ii. *if $\mu_1 = \mu_2$ the LSM module authorizes the action X;*

As we can see in Protocol 1, an *operation PIN* and one *administration password* are used in our protocol. The operation PIN is at least six digits long. We use the operation PIN in order to authorize the actions. On the other hand, the administration password is used to change the operation PIN and other management tasks. The system administrator has three consecutive chances to enter the operation PIN. In the third chance if the smart-card receives an incorrect operation PIN it blocks itself. The smart-card only can be unblocked with the administration password. Again, there are three chances to enter the correct administration password. If the smart-card is blocked with the administration password the smart-card becomes useless.

The security parameters of the LSM module are properly initialized when it is installed. The system administrator inserts a smart-card in the reader device and the cardlet application is downloaded to the smart-card. Once the applet has been downloaded and registered, the system administrator introduces the administration password and the operation PIN. The LSM module then sends the shared key K – it stores the shared K in a secure file, so the file can be read exclusively by the LSM module.

Then, the smart-card and the LSM module share a secret key K. In Step 1 of such a protocol, the system administrator requests an action to the LSM module which, in turn, blocks the communication channel between the smart-card reader and the LSM module. The data sent between the LSM module and the smart-card can neither be sniffed nor tampered because the channel is blocked (cf. Step 2a). The protocol avoids the smart-card remains in the smart-card reader when is not necessary. In Step 2c, the LSM module waits until the smart-card insertion, and in Step 4f the LSM module does not proceed since the smart-card has been removed.

In Step 3 the operation PIN travels in a secure way from the keyboard since, as we recalled above, the LSM module has blocked the channel between the reader and the module itself (cf. Step 2a). Then, the LSM module sends a NONCE obtained at random and the PIN in Step 4d. The smart-card returns a Message Authentication Code (*MAC*) of the NONCE computed with the shared key K. In the last Step, i.e., Step 4h, the LSM module verifies whether the MAC has been properly computed.

Let us now define Procedure 1, which is executed within the smart-card to validate the operation PIN. If the operation PIN is correct, it computes the MAC of NONCE with the shared key K.

Procedure 1 *[PIN, $NONCE$]*

1. *Validate the operation PIN;*
2. *If the operation PIN is correct do:*
   (a) *Compute the Message Authentication Code (MAC) of NONCE with the shared key K, $\mu_2 = MAC(K, NONCE)$;*
   (b) *return μ_2;*
3. *If the operation PIN is no correct return* `ERROR`*;*

For performance improvement purposes, let us also show an alternative version of Protocol 1, where a smart-card reader device with a key-pad is used. This way, the operation PIN is not typed in the keyboard of the computer but in the keypad of the smart-card reader device. Such an alternative version (i.e., Protocol 2) is described as follows:

Protocol 2

1. *The system administrator opens a new console and he requests an action X. We assume X must be authorized using the smart-card;*

2. *The LSM module receives the request from the console and it does the following steps:*

   (a) *Open a connection to the smart-card reader device, and protect the channel between the smart-card reader device and the LSM module itself to avoid being tampered or sniffed by any other process;*

   (b) *Print a message in the console, where ask for to insert the smart-card to the smart-card reader;*

   (c) *While the smart-card has not been inserted do;*

      i. *Detect the insertion of the smart-card;*

   (d) *Print a message in the console asking for the operation PIN;*

3. *The system administrator types the operation PIN in the key-pad of the smart-card reader. The PIN can not be obtained by any process running in the system;*

4. *The LSM does the following steps:*

   (a) *Obtain a NONCE value at random;*

   (b) *Compute the Message Authentication Code (MAC) of NONCE with the shared key K, $\mu_1 = MAC(K, NONCE)$;*

   (c) *Execute the Procedure 1 inside the smart-card using the NONCE, and obtain a response μ_2;*

   (d) *Print a message in the console to remove the smart-card from the smart-card reader;*

   (e) *While the smart-card has not been removed do;*

      i. *Detect the removing of the smart-card;*

   (f) *if μ_2 is* ERROR *the LSM does not authorize the action X;*

   (g) *else do:*

      i. *if $\mu_1 \neq \mu_2$ the LSM does not authorize the action X;*

      ii. *if $\mu_1 = \mu_2$ the LSM authorizes the action X;*

Asymmetric version of our Cryptographic Protocol

Alternatively to protocols 1 and 2, we propose a second authentication mechanism using asymmetric cryptography. Let us start by introducing the necessary structure and elements for this second proposal. We first define the necessary architecture as a hierarchical structure with several organizational units, where the computer network is divided, in turn, in hierarchical domains, and where each domain of the network has several components that must be protected. We name such a component as SMARTCOP Node (SCN). Finally, each domain has a SMARTCOP Server (SCS), and each potential administrator holds a SMARTCOP Card (SCC). These components are briefly described below.

SMARTCOP Server (SCS) – Each SCS owns a cryptographic key pair *master key* and the corresponding certificate. This certificate has been issued by the upper SCS in the hierarchy and identifies the lower SCS as a valid SCS. This certificate is encoded as an X.509 Attribute Certificate [International Organisation for Standardization, 2000], where the issuer is the upper SCS master key and the subject is the lower SCS master key. The SCS of domain B can issue certificates authorizing a concrete SCC as an administration of the domain B (similar to the certificates between SCSs). Normally the SCS will be managed by the officer in charge of the network administration in the given domain – or organizational unit. That is, the person who has more knowledge about the network domain and its potential administrators, and, at the same time, the one that has the greatest interest in performing a good administration. This is a key point of the SMARTCOP framework, which enables the distribution of the administrative management between domains or organizational units.

SMARTCOP Node (SCN) – Each SCN is a component which has the SMARTCOP LSM module. The security parameters of the LSM module are properly initialized when it is installed. The main parameter is the *Source-of-Authority* (SoA), which is represented by a *master-key*. More precisely, the *master-key* of the top SCS. When an administrator requests a protected action on a given SCN, by using Protocol 3, the SCN verifies the certificate from the SCC. Then, if it comes from a certificate path rooted at the SoA's *master-key*, the operation is accepted.

SMARTCOP Card (SCC) – The SCC is owned by potential administrators. In order to be able to perform administrative tasks on a given domain, the SCC must be authorized (i.e., certified) by the SCS of the domain or an upper one in the hierarchy.

Each SCC has a key pair, which has to be certified by a *master-key* (i.e., a key from a SCS). Let us recall that the cryptographic engine of such a smart-card is capable of performing several cryptographic functions, such as asymmetric key generation, asymmetric cryptographic algorithms execution, and so on. The SCC has an *operation PIN* and an *administration password*.

The operation PIN is at least six digits long and is used to authorize the protected actions. On the other hand, the administration password is used to change the operation PIN and other management tasks. The system administrator has three consecutive chances to enter the operation PIN. In the third entry, if the smart-card receives an incorrect operation PIN, it blocks itself. The smart-card can only be unblocked with the administration password. Again, there are three chances to enter the correct administration password. Finally, if the smart-card blocks itself after the failing of three consecutive wrong administration passwords, it becomes useless.

Protocol Description

We give in this section a detailed description of the asymmetric cryptographic protocol version (cf. Protocol 3). In Step 1 of such a protocol, the system administrator requests an action to the LSM module which, in turn, blocks the communication channel between the smart-card reader and the LSM module. The data sent between the LSM module and the smart-card can neither be sniffed nor tampered because the channel is blocked (cf. Step 2a). The protocol does not allow the smart-card to remain in the reader when is not necessary. In Step 2c, the LSM module waits for the smart-card insertion, and in Step 4e the LSM module does not proceed until the smart-card has been removed. In Step 3 the operation PIN travels in a secure way from the keyboard because the LSM module has blocked the channel between the keyboard and the module itself. Then, LSM sends a NONCE obtained at random and the PIN in step 4c. The smart-card returns the digital signature of the NONCE computed with the smart-card's private key. Finally, in Step 4g LSM verifies whether the digital signature has been computed properly and the digital certificate is valid.

Protocol 3

1. *The system administrator opens a new console and he requests an action X. It is assumed that X must be authorized by using the smart-card;*

2. *LSM receives the request from the console and it does the following steps:*

   (a) *Open a connection to the smart-card reader device, and protect the channel between the smart-card reader device and the LSM module itself to avoid being tampered or sniffed by any other process;*

   (b) *Print a message in the console, asking to insert the smart-card into the smart-card reader;*

   (c) *While the smart-card has not been inserted do;*

      i. *Detect the insertion of the smart-card;*

   (d) *Print a message in the console asking for the operation PIN;*

3. *The system administrator types the operation PIN in the keyboard;*

4. *The LSM does the following steps:*

   (a) *Obtain the operation PIN;*

   (b) *Obtain a NONCE value at random;*

   (c) *Execute the Procedure 2 inside the smart-card by using the operation PIN and the NONCE, and obtain a response μ;*

   (d) *Print a message in the console to remove the smart-card from the smart-card reader;*

(e) *While the smart-card has not been removed do;*

  i. *Detect the removing of the smart-card;*

(f) *if μ is* `ERROR` *the LSM does not authorize the action X;*

(g) *else do:*

  i. *Check if the digital signature has been computed with a public key, which belongs to a certification path rooted at the* master key *(SoA).*

  ii. *Verify the smart-card certificate against a valid CRL.*

  iii. *Verify the digital signature μ with the public key P_K obtained from the smart-card certificate, $P_K(\mu) \overset{?}{=} H(NONCE)$;*

  iv. *if the verification is correct the LSM authorizes the action X;*

  v. *if the verification is not correct the LSM does not authorize the action X;*

The Procedure 2 is executed within the smart-card. The smart-card validates the operation PIN. If the operation PIN is valid it computes the digital signature of NONCE with the smart-card private key.

Procedure 2 *$[PIN, NONCE]$*

1. *Validate the operation PIN;*
2. *If the operation PIN is correct do:*
   (a) *Compute the digital signature of NONCE with the private key S_K, $\mu = S_K(NONCE)$;*
   (b) *return μ;*
3. *If the operation PIN is no correct return* `ERROR`*;*

Security Considerations

To ensure the proper execution of both protocols and procedures shown above, we must consider the protection of the entities and the channels involved in such a process, avoiding attacks like impersonation or channels data manipulation. The lack of ability to avoid these attacks and their impact makes our proposed protection mechanism usefulness.

Regarding the different entities that take part in the protocol, we suggest in this section the following considerations. First, the possible console attacks could be directed against the binary executable file and the console process in execution time. If this happens, an overwrite of the executable console's file using malicious code could lead an attacker to take the control of the authentication process, giving him the possibility to complete the protocol and get the control of the system – and even to steal the smart-card's PIN. To eliminate this attack, the LSM module guarantees that the binary file of the console can not be overwritten by anybody (even the administrator), remaining the permissions as read-only. Second, the binary executable of the administration console is compiled in a static manner. this allows us to reduce the complexity of the protection's console process, since we do not need to consider tasks introduced by the loading of dynamic or shared libraries and its associated files.

At the same time, it enables us to centralize and reduce the failure points that could be used by an intruder to tamper the console's process. Thus, and to protect the process associated to the console, the LSM module controls that each system call launched by some process can not be dangerous for the correct execution flow of the console process, such as keyboard key capture, cancellation, or debugging process system calls. Let us recall that the communication channels can not be manipulated by any opponent. To achieve this purpose, the LSM mediates between the system calls related with the communication channels and the entities that take part within the protocol (the LSM module, the smart-card and the console process).

To conclude, and as pointed out in [Biondi, 2003], the LSM module does not need to be directly protected since we can assume the kernel environment as a trusted area – since it is mandatory for the kernel security model of our prototype's operating system.

7.4 Configuration and Performance Evaluation

In order to define the objects and resources to protect, SMARTCOP can actually be configured through a set of security rules. Each security rule defines an *action* in $\{deny, accept\}$ that applies over a set of *condition* attributes, such as user_id (*UID*), process_id (*PID*), device, i-node, etc. These security rules are stored in a set of configuration files that are loaded at boot time through the *proc file system*. The *proc file system* (procfs) is a special virtual file system in the Linux kernel which allows user space programs to access to kernel data structures. We do not consider for the moment the reload of rules at runtime.

Up to now, we can define several sets of rules regarding the three basic levels of protection stated in Section 7.1. In Figure 7.3, for instance, we show an example of seven different configuration points through procfs for configuring the protection of, respectively, i-node permission verification, i-node renaming, i-node permission changing, i-node removing, process tracing, process creation, and process termination.

```
-rw------- 1 root root 0 jul 23 13:28 /proc/smartcop/iperms
-rw------- 1 root root 0 jul 23 13:28 /proc/smartcop/iren
-rw------- 1 root root 0 jul 23 13:28 /proc/smartcop/isetattr
-rw------- 1 root root 0 jul 23 13:28 /proc/smartcop/iunlink
-rw------- 1 root root 0 jul 23 13:28 /proc/smartcop/ptrace
-rw------- 1 root root 0 jul 23 13:28 /proc/smartcop/tcreate
-rw------- 1 root root 0 jul 23 13:28 /proc/smartcop/tkill
```

Figure 7.3: Sample configuration points of SMARTCOP.

The four first configuration points (i.e., iperms, iren, isetattr, and iunlink) refer to i-node related operations. Hence, they can be used not only for the protection of file resources, but also for the protection of communication operations through, for example, sockets and pipes. On the other hand, the three last configuration points (i.e., ptrace, tcreate, and tkill) are related to process operations. More specifically, the configuration point *ptrace* allows the protection of the system call *ptrace()* that is often used from some processes to control the execution of other processes; the configuration point *tcreate* allows the configuration of rules related with the creation of new processes through the system call *fork()*; and the last configuration point *tkill* allows the configuration of rules related with the managing of processes (such as termination, blocking, and resuming) through the system call *sys_kill()*.

Through the configuration points showed in Figure 7.3, we conducted several tests steered towards measuring the penalty introduced by the installation of SMARTCOP as a LSM module, over the normal operation of the system. The tests and benchmarks were based on the use of the Strace [Akkerman, 2003] and the LMbench [McVoy and Staelin, 1996] tools. Strace is a debugging tool, which allows us to trace the system calls made after the execution of a given process. This can be used to analyze and evaluate the time taken by these calls. On the other hand, LMbench is used to perform *microbenchmarks*, which are used to take more precise measures of the time taken for file access, memory access, and so on. The evaluation was carried out on a single machine with an Intel-Pentium M 1.4 GHz, with 512 MB of RAM memory and an IDE hard disc of 5400 rpm, running a Debian GNU/Linux operating system and ext3 file system.

The objective of these testbeds is to compare the performance of the system using a normal Linux 2.6.15 kernel without LSM support towards the performance of the same system and kernel but with LSM support and the SMARTCOP module loaded. The results of the testbeds are shown in Figure 7.4. They are organized in three tables depending on the three protection levels stated in Section 7.1. As it can be appreciated in the results, the penalty introduced by SMARTCOP has a minimum impact on the performance of a standard GNU/Linux 2.6.15 system.

The first table (*Process tests*) shows the latency in microseconds for a set of operations related to the execution of processes and system calls such as process creation through *fork()*, *fork()+exec()* and *sh()*, process cancellation through *kill()*, descriptor waiting through *select()*, opening and closing files through *open()*/*close()*, signal installation, and so on. This first category of tests shows that more than the 50% of the tests indicate a performance penalty below 2%. For example, the process creation with *fork()* is scarcely penalized with a 0.9%. The same can be noticed for process creation with *fork()+exec()* and *sh()*, which have an approximate penalty of 3.3%. On the other hand, the higher performance penalty is presented by the process cancellation through the system call *kill()* with a 4.6%. This higher

Test Type	**2.6.15**	**2.6.15 + smartcop**	**% Overhead with smartcop**
null call	0.255	0.255	0%
kill	231.10	241.65	4.6%
stat	1.99	2.03	2%
open/close	2.96	3.02	1.9%
select TCP	18.63	18.86	1.2%
sig inst	0.9	0.9	0%
sig handl	1.85	1.88	0.1%
fork proc	95.61	96.52	0.9%
exec proc	100.50	103.86	3.3%
sh proc	2227	2302	3.3%

Process tests, time in μseconds

Test Type	**2.6.15**	**2.6.15 + smartcop**	**% Overhead with smartcop**
pipe	1342	1338	0.2%
AF Unix	1334	1320	1%
TCP	1088	1078	0.9%
file read	1330	1308	1.6%
mmap read	1480	1425	3.8%
mem bcopy	5278	5277	0.01%
mem bzero	4548	4548	0%
mem read	25600	25590	0.03%
mem write	24888	24869	0.07%

Local communication bandwidth in MB/s

Test Type	**2.6.15**	**2.6.15 + smartcop**	**% Overhead with smartcop**
0K file create	193	193	0%
0K file delete	489	489	0%
10K file create	175	176	0.5%
10K file delete	658	668	1.5%
mmap latency	2348	2348	0%
par mem	1.26	1.26	0%
page fault	0.974	0.981	0.8%

File and VM sytem latencies, time in μseconds

Figure 7.4: Performance evaluation of the protection module.

penalty is produced by the access control verifications of SMARTCOP at kernel level, during the identification checks of the process, system call parameters, etc. The second set of tests shown in the second table of Figure 7.4, presents the bandwidth of operations related to communication issues such as reading, writing and copy of memory sections through *read()* and *mmap()*, Inter Process Communications (IPC) using TCP, pipes and sockets of the Unix address family (*AF Unix sockets*), etc. Again, the results show a minimum penalty in the performance. In this case the greater penalty (3.8% approx.) is found in the reading and summing of a file via the memory mapping *mmap()* interface.

Finally, the set of tests from the third table (Figure 7.4) shows the latency found in operations related to file and memory manipulation. The performance penalty of the system is also minimum. The greater penalty being introduced by the file elimination due to the verifications performed by SMARTCOP during the associated system calls.

Summary

In this chapter we have presented an access control mechanism intended for the protection of network security components, such as firewalls and intrusion detection systems. Whenever one of these components, or one of its elements, is compromised by an attacker, it may lead her to obtain the full control of the network. Our proposed solution consists of a kernel based access control method which intercepts and cancels forbidden system calls. Hence, even if an attacker gains administration permissions, he will not achieve its purposes.

We first introduced in Section 7.1 the motivation of using a kernel based approach and we gave an outlook to our protection strategy. We then discussed in Section 7.2 the choice of the *Linux Security Modules* (LSM) framework to implement our approach. The use of LSM allows us to use our kernel based access control in new components by just considering its environment and its interactions. It reinforces moreover the modularity of the system and provides an easy and generic way to introduce new elements without having to consider each component separately.

Our strategy introduces however some administration constraints, since administration officers should not be able to throw system calls that are defined as possible threats to the component. To solve these constraints, we presented in Section 7.3 a smart-card based authentication mechanism, which acts as a reinforcement of the kernel based access control. The objective of this complementary mechanism is twofold. First, it holds to the administrator the indispensable privileges to carry management and configuration activities just when he verifies his identity through a two-factor authentication mechanism. Second, it allows us to avoid some kind of logical attacks focused on getting the rights of the administrative entity, such as password forgery or buffer overflows.

We finally discussed in Section 7.4 some configuration issues of our proposal and presented the evaluation results of several tests steered towards measuring the overhead introduced by our strategy, over the normal operations of a given system. These results showed that our approach offers a good degree of transparency to the administrator in charge, and it does not interfere directly with user space's processes.

Chapter 8

Conclusions

”The reward of a thing well done is to have done it.”
– RALPH WALDO EMERSON

In this dissertation we have presented our theoretical and practical work for the design and development of a policy-based framework, whose main purpose is the managing of both detection and prevention of intrusion attacks. This framework is proposed as a complementary element to traditional network security mechanisms, such as firewalls and cryptography, in order to detect and prevent network security policy violations. Our proposal is intended to act as a central point within a given network security infrastructure, in order to specify security requirements free of anomalies, and deploy the necessary mechanisms to guarantee the interoperability, cooperation, and protection of the security components used within such a security infrastructure.

The first part of the dissertation (i.e., chapters 2 and 3) introduced the basic concepts of computer and network security, provided an overview of the classical security mechanisms (e.g., cryptography and firewalls) to guarantee the security of a network system, discussed the necessity of complementary mechanisms (i.e., intrusion detection systems), and surveyed some related work which has been previously done in the areas of research in which this dissertation falls into. The second part of the dissertation described the contributions we have done in the research domain of cooperation and protection of network security components, exchange of audit information, and analysis of network security policies.

More specifically, we started in Chapter 4 by giving and outlook to the audit process we presented in [García et al., 2006f, García et al., 2006d] to set a distributed security scenario composed of both *firewalls* and *network intrusion detection systems* (NIDSs) free of anomalies. Our audit process has been presented in two main blocks. We presented, in Section 4.2, a set of algorithms for intra-component analysis, according to the discovering and removal of policy anomalies over single-component environments; and, in Section 4.3, we presented a set of algorithms for inter-component analysis, in order to detect and warn the security officer about the complete existence of anomalies over a multi-component environment.

The main advantages of our approach are the following. First, our process verifies that the resulting rules are completely independent between them. Otherwise, each rule considered as useless during the process is reported to the security officer, in order to verify the correctness of the whole process. Second, the network model presented in Section 4.1 allows us to determine which components are crossed by a given packet knowing its source and destination, as well as other network properties. Thanks to this model, our approach better defines all the set of anomalies studied in the related work, and it reports, moreover, two new anomalies (*irrelevance* and *misconnection*) not reported, as defined in our work, in none of the other approaches. Furthermore, the lack of this model in other approaches (e.g., [Al-Shaer et al., 2005]) leads to inappropriate decisions.

The implementation of our approach in a software prototype demonstrates the practicability of our work. We shortly discussed this implementation, based on a scripting language [Castagnetto et al., 1999], and presented an evaluation of its performance. Although these experimental results show that our algorithms have strong requirements, we believe that these requirements are reasonable for off-line analysis, since it is not part of the critical performance of the audited component.

In order to communicate effectively and efficiently the different components of our platform, we presented in Chapter 5 an infrastructure to share messages and audit information between those components [García et al., 2005a, García et al., 2005e]. The framework itself is based on IDMEF [Debar et al., 2006] (Intrusion Detection Message Exchange Format) and the publish/subscribe communication paradigm (cf. sections 5.1 and 5.2).

In contrast to traditional client/server solutions, where centralized or hierarchical approaches quickly become a bottleneck due to saturation problems associated with the service offered by centralized or master domain analyzers, the information exchange between peers in our design achieves a more complete view of the system in whole.

We then presented in Section 5.3 an audit information exchange between components based on XML messages and XPATH filters, implemented in our proposal via a push or pull data exchange, and based on an open source publish/subscribe message oriented middleware [Ruff, 2006]. We also conducted experiments showing that the proposal is performant enough for the application in real-world scenarios (cf. Section 5.4).

In Chapter 6, we discussed the reaction mechanism presented in [Cuppens et al., 2006a]. This proposal extends the detection process introduced in [Cuppens and Miège, 2002], in order to select and apply a response mechanism when an intrusion occurs. It is based on an attack description language based on logic, and whose scenarios steps represent the attacker's actions [Cuppens and Ortalo, 2000]. In Section 6.1, we suggest how to use such a language to build libraries of intrusions and counter-measures.

The notion of anti-correlation is then used to select relevant responses to a given intrusion in order to help the administrator to decide which appropriate counter-measures may be launched. This mechanism is integrated together with the communication infrastructure presented in Chapter 5, and some examples have been shown to illustrate the applicability of our approach in real-world scenarios.

Up to now, we only use this approach to provide a support to the administrator who takes the final decision to choose and launch a given response. This is a prudent strategy but it introduces an overhead that is sometimes incompatible with real time response.

We finally described in Chapter 7 an access control mechanism specially suited for the protection of network security components, such as *firewalls* and *network intrusion detection systems* (NIDSs) [García et al., 2005b, García et al., 2005c]. As pointed out in [Geer, 2004], when one of these components, or one of its elements, is compromised by a remote adversary, it may lead such an adversary to obtain the full control of the network and its components.

The solution we provided proposes the protection of the components by making use of the *Linux Security Modules* (LSM) framework for the Linux kernel over GNU/Linux systems [Wright et al., 2002]. The developed mechanism works by providing and enforcing access control rules at system calls, and is based on a protection module integrated into the operating system's kernel, providing a high degree of modularity and independence between elements.

The use of LSM allows our protection system to be used in new components and elements, by just considering its environment and its interactions (regarding access control). It reinforces the modularity of the system and provides an easy and generic way to introduce new elements without having to consider each component separately. Thus, we consider that our proposal provides a high degree of scalability. The introduction of new components provides a minimum performance penalty, because the LSM framework and the access control schema do not introduce an excessive computational complexity. We also measured the penalty introduced by our approach against the usual performance of the system. The results show the minimum performance impact of our proposal.

To reinforce the protection mechanism itself, our implementation provides a complementary authentication method, based on smart-card technology. This additional enhancement is based both on a secret (smart-card PIN) and a physical token (the smart-card itself). This way, we can prevent some logical attacks (e.g., password forgery) against the protection mechanism itself. For all these reasons, we may conclude that the enhanced access control proposed, and integrated inside the operating system's kernel through the *Linux Security Modules* (LSM) framework, offers a good degree of transparency to the security officer, and it does not interfere directly with user space's processes.

Future Work

As an extension of the work presented in Chapter 4, we may consider the study of anomaly problems of security rules in the case where the security architecture not only includes firewalls and IDS, but also IPSec devices. More specifically, when the configuration of a network includes security rules of these three enforcement devices, and although there is a real similarity between the parameters of those devices' rules, a more complete set of anomalies may be addressed. Up to now, our study just addresses to single-trigger policies, i.e., a list of predicates leading a one possible action. Nevertheless, when considering IPSec devices, a new kind of policies appear. These policies, referred in the literature as multi-trigger policies, often represent a collection of predicates leading to more than one action. Multi-trigger policies are used on IPSec devices to represent network traffic transformations, such as sign and crypt the traffic by using different algorithms.

In parallel to this work, we are also considering to extend the approach presented in Chapter 4 to the analysis of stateful policies. For the moment, just static non-stateful policies have been studied. Stateful policies allows firewalls for dynamically inspecting network traffic by keeping the state of each connection that is established between the surveyed zones. In order to do so, this state is kept in dynamic tables at the memory of the device, and it allows it to inspect the actions that will follow the initial one. More investigation has to be done in order to extend our proposal for using these new classes of policies (i.e., multi-trigger and stateful policies).

As an extension of the work presented in Chapter 5, we may first consider to secure the communication partners by utilizing the SSL protocol [Schneier, 1996]. This way, each node will receive a private and a public key. The public key of each node will be signed by a certification authority (CA), that is responsible for the protected network. Hence, the public key of the CA has to be distributed to every node as well. The secure SSL channel will allow the communicating peers to communicate privately and to authenticate each other, thus preventing malicious nodes from impersonating legal ones. The implications coming up with this new feature, such as compromised key management or certificate revocation, would be part of this future work.

We may also consider as further work to the approach presented in Chapter 5 a more in-depth study about privacy mechanisms by exchanging alerts in a pseudonymous manner. By doing this, one may provide the destination and origin information of alerts (*Source* and *Target* field of IDMEF messages) without violating the privacy of publishers and subscribers located on different domains. Our study could cover the design of a pseudonymous identification scheme, trying to find a balance between identification and privacy. This also represents further work that remains to be done.

A possible extension of the work presented in Chapter 6 would be the analysis of situations where it would be possible to *automatically* decide to launch the response. Notice that a possible response consists in reconfiguring the security policy to prevent a new occurrence of a given intrusion. However, as suggested in [Petkac and Badger, 1997], dynamic changes of the security policy may cause failure of some software components. This is why [Petkac and Badger, 1997] suggests the notion of security agility, a strategy to provide software components with adaptability to security policy changes.

Security agility might be nicely included into the intrusion detection and response framework suggested in Chapter 6. This represents a possible extension of our work. When using anti-correlation, moreover, several responses may be selected. In this case, it would be interesting to rank these different responses and a possible ranking criteria would be to properly evaluate the effectiveness of the responses to stop the attack. A possible extension of our response formalism would be the use of temporal logic to include the fact that a given response will stop an intrusion *until* another additional event occurs. More difficult is performing an action that will cause this additional event, more effective is the response.

Finally, as an extension of the work we presented in Chapter 7 we are considering improving the customizing of policies. Up to now, the specific policy that is enforced by our protection module is loaded at boot time through the *proc file system* (procfs). We are planning to extend this feature to add the possibility of using text-based configuration files and the reload of policies at runtime. We are also considering to continue our study to address the security of the system from an intrusion tolerance point of view [Deswarte et al., 1991], i.e., the inclusion of mechanisms that may allow the system to maintain its services in an acceptable manner, though possibly degraded, despite being attacked.

Bibliography

[Adiseshu et al., 2000] Adiseshu, H., Suri, S., and Parulkar, G. (2000). Detecting and resolving packet filter conflicts. In *19th Annual Joint Conference of the IEEE Computer and Communications Societies*, pages 1203–1212.

[Akkerman, 2003] Akkerman, W. (2003). STRACE: System call trace debugging tool. `http://www.liacs.nl/~wichert/strace/`.

[Al-Shaer and Hamed, 2004] Al-Shaer, E. and Hamed, H. (2004). Discovery of policy anomalies in distributed firewalls. In *23rd Conference of the IEEE Communications Society (INFOCOM'04)*, pages 2605–2616.

[Al-Shaer et al., 2005] Al-Shaer, E., Hamed, H., Boutaba, R., and Hasan, M. (2005). Conflict classification and analysis of distributed firewall policies. *IEEE Journal on Selected Areas in Communications*, 23(10):2069–2084.

[Almgren and Lindqvist, 2001] Almgren, M. and Lindqvist, U. (2001). Application-integrated data collection for security monitoring. In *Fourth International Symposium on Recent Advances in Intrusion Detection (RAID2001)*, pages 22–36, Davis, CA, USA.

[Anderson et al., 1995a] Anderson, D., Frivold, T., and Valdes, A. (1995a). *Next-generation Intrusion Detection Expert System (NIDES): a summary*. SRI International, Computer Science Laboratory.

[Anderson et al., 1995b] Anderson, D., Lunt, T. F., Javits, H., Tamaru, A., and Valdes, A. (1995b). Detecting unusual program behavior using the statistical components of NIDES. NIDES technical report, SRI International.

[Anderson, 1980] Anderson, J. P. (1980). Computer security threat monitoring and surveillance. James P. Anderson Co., Fort Washington, PA.

[Asaka et al., 1999] Asaka, M., Taguchi, A., and Goto, S. (1999). The implementation of IDA: An intrusion detection agent system. In *11th Annual FIRST Conference on Computer Security Incident Handling and Response (FIRST'99)*.

[Autrel, 2005] Autrel, F. (2005). *Fusion, Weighted Correlation and Reaction in a Cooperative Intrusion Detection Framework*. PhD thesis, SUPAERO, France.

[Bartal et al., 1999] Bartal, Y., Mayer, A. J., Nissim, K., and Wool, A. (1999). Firmato: a novel firewall management toolkit. In *IEEE Symposium on Security and Privacy*, pages 17–31, Oakland, CA, USA.

[Bellovin and Cheswick, 1994] Bellovin, S. M. and Cheswick, W. R. (1994). Network firewalls. *IEEE Communications Magazine*, 32(9):50–57.

[Biondi, 2003] Biondi, P. (2003). Linux kernel level security. Free and Open source Software Developers' European Meeting (FOSDEM), Brussels.

[Castagnetto et al., 1999] Castagnetto, J., Rawat, H., Schumann, S., Scollo, C., and Veliath, D. (1999). *Professional PHP Programming*. Wrox Press Inc.

[Castillo et al., 2005a] Castillo, S., García, J., and Borrell, J. (2005a). Design and development of the detection and reaction subsystem of a platform that prevents coordinated attacks. In *Third Spanish Symposium on Electronic Commerce (SCE 2005)*, pages 47–58, Illes Balears, Spain. *In Spanish*.

[Castillo et al., 2005b] Castillo, S., García, J., Navarro, G., and Borrell, J. (2005b). Protection of the components of a platform that prevents coordinated attacks. In *First Spanish Conference on Informatics (CEDI 2005), Information Security Symposium*, pages 265–272, Granada, Spain. *In Spanish*.

[Cuppens, 2001] Cuppens, F. (2001). Managing alerts in a multi-intrusion detection environment. In *17th Annual Computer Security Applications Conference (ACSAC'01)*, pages 22–32, New Orleans, Lousiana.

[Cuppens et al., 2006a] Cuppens, F., Autrel, F., Bouzida, Y., García, J., Gombault, S., and Sans, T. (2006a). Anti-correlation as a criterion to select appropriate counter-measures in an intrusion detection framework. *Annals of Telecommunications*, 61(1-2):192–217.

[Cuppens et al., 2005a] Cuppens, F., Cuppens, N., and García, J. (2005a). Detection and removal of firewall misconfiguration. In *2005 IASTED International Conference on Communication, Network and Information Security*, pages 154–162, Phoenix, AZ, USA.

[Cuppens et al., 2005b] Cuppens, F., Cuppens, N., and García, J. (2005b). Misconfiguration management of network security components. In *7th International Symposium on System and Information Security (SSI 2005)*, pages 1–10, Sao Paulo, Brazil.

[Cuppens et al., 2006b] Cuppens, F., Cuppens, N., and García, J. (2006b). Detection of network security component misconfiguration by rewriting and correlation. In *5th Conference on Security and Network Architectures*, pages 225–240, Seignose, France.

[Cuppens et al., 2004] Cuppens, F., Cuppens, N., Sans, T., and Miège, A. (2004). A formal approach to specify and deploy a network security policy. In *Second Workshop on Formal Aspects in Security and Trust*, pages 203–218, Toulouse, France.

[Cuppens and Miège, 2002] Cuppens, F. and Miège, A. (2002). Alert correlation in a cooperative intrusion detection framework. In *IEEE Symposium on Security and Privacy*, pages 202–215, Oakland, CA, USA.

[Cuppens and Ortalo, 2000] Cuppens, F. and Ortalo, R. (2000). LAMBDA: A language to model a database for detection of attacks. In *Third International Workshop on the Recent Advances in Intrusion Detection (RAID'2000)*, volume 1907 of *Springer LNCS*, pages 197–216, Toulouse, France.

[Debar et al., 2006] Debar, H., Curry, D., and Feinstein, B. (2006). Intrusion detection message exchange format data model and extensible markup language. Technical report.

[Debar et al., 1999] Debar, H., Dacier, M., and Wespi, A. (1999). Towards a taxonomy of intrusion detection systems. *Computer Networks*, 31(8):805–822.

[Debar and Wespi, 2001] Debar, H. and Wespi, A. (2001). Aggregration and correlation of intrusion-detection alerts. In *4th International Symposium on Recent Advances in Intrusion detection*, pages 85–103.

[Deering, 1989] Deering, S. (1989). Host Extensions for IP Multicasting. STD 5, RFC 1112, Stanford University, May 1988.

[Denning, 1987] Denning, D. (1987). An Intrusion-Detection Model. *IEEE Transactions on Software Engineering*, 13(2):222–232.

[Deswarte et al., 1991] Deswarte, Y., Blain, L., and Fabre, J. C. (1991). Intrusion tolerance in distributed computing systems. In *IEEE Symposium on Security and Privacy*, pages 110–121, Oakland, CA, USA.

[Diffie, 1988] Diffie, W. (1988). The first ten years of public-key cryptography. *Proceedings of the IEEE*, 76(5):560–577.

[Eppstein and Muthukrishnan, 2001] Eppstein, D. and Muthukrishnan, S. (2001). Internet packet filter management and rectangle geometry. In *12th annual ACM-SIAM symposium on Discrete Algorithms*, pages 827–835, Washington, DC, USA.

[Eugster et al., 2003] Eugster, P., Felber, P., Guerraoui, R., and Kermarrec, A. (2003). The many faces of publish/subscribe. *ACM Computing Surveys*, 35(2):114–131.

[Feiertag et al., 1999] Feiertag, R., Kahn, C., Porras, P., Schnackenberg, D., Staniford-Chen, S., and Tung, B. (1999). A Common Intrusion Specification Language. CIDF working group document.

[Fox et al., 1990] Fox, K. L., Henning, R. R., Reed, J. H., and Simonian, R. (1990). A neural network approach towards intrusion detection. In *13th National Computer Security Conference*, pages 125–134, Washington, DC, USA.

[Frantzen et al., 2001] Frantzen, M., Kerschbaum, F., Schultz, E., and Fahmy, S. (2001). A framework for understanding vulnerabilities in firewalls using a dataflow model of firewall internals. *Journal of Computers and Security*, 20(3):263–270.

[García et al., 2004a] García, J., Autrel, F., Borrell, J., Bouzida, Y., Castillo, S., Cuppens, F., and Navarro, G. (2004a). Preventing coordinated attacks via alert correlation. In *9th Nordic Workshop on Secure IT Systems (NORDSEC 2004)*, pages 110–117, Helsinki, Finland.

[García et al., 2004b] García, J., Autrel, F., Borrell, J., Castillo, S., Cuppens, F., and Navarro, G. (2004b). Decentralized publish-subscribe system to prevent coordinated attacks via alert correlation. In *Sixth International Conference on Information and Communications Security*, volume 3269 of *Springer LNCS*, pages 223–235, Málaga, Spain.

[García and Barrera, 2006] García, J. and Barrera, I. (2006). Distributed exchange of alerts for the managing of coordinated attacks. In *Spanish Meeting on Cryptology and Information Security (IX RECSI)*, Barcelona, Spain. *In Spanish*.

[García et al., 2005a] García, J., Borrell, J., Jaeger, M. A., and Mühl, G. (2005a). An alert communication infrastructure for a decentralized attack prevention framework. In *IEEE International Carnahan Conference on Security Technology*, pages 234–237, Las Palmas de G.C., Spain.

[García et al., 2006a] García, J., Castillo, S., Castellà, J., and Navarro, G. (2006a). Protection of security devices based on a kernel access control. In *Spanish Meeting on Cryptology and Information Security (IX RECSI)*, Barcelona, Spain. *In Spanish*.

[García et al., 2006b] García, J., Castillo, S., Castellà, J., Navarro, G., and Borrell, J. (2006b). Protection of components based on a security module. In *1st International Workshop on Critical Information Infrastructures Security*, Springer LNCS, pages 129–140, Samos, Greece.

[García et al., 2006c] García, J., Castillo, S., Castellà, J., Navarro, G., and Borrell, J. (2006c). SMARTCOP - A Smart Card Based Access Control for the Protection of Network Security Components. In *International Workshop on Information Security (IS'06), 2006 International OTM Conference,* volume 4277 of *Springer LNCS*, pages 415–424, Montpellier, France.

[García et al., 2004c] García, J., Castillo, S., Navarro, G., and Borrell, J. (2004c). Design and development of a collaborative system to prevent coordinated attacks. In *Advances in Crytptology and Information Security (VIII RECSI)*, pages 475–493, Madrid, Spain. *In Spanish*.

[García et al., 2005b] García, J., Castillo, S., Navarro, G., and Borrell, J. (2005b). ACAPS: An Access Control Mechanism to Protect the Components of an Attack Prevention System. *Journal of Computer Science and Network Security*, 5(11):87–94.

[García et al., 2005c] García, J., Castillo, S., Navarro, G., and Borrell, J. (2005c). Mechanisms for attack protection on a prevention framework. In *IEEE International Carnahan Conference on Security Technology*, pages 137–140, Las Palmas de G.C., Spain.

[García et al., 2005d] García, J., Cuppens, F., Autrel, F., Castellà, J., Borrell, J., Navarro, G., and Ortega, J. A. (2005d). Protecting on-line casinos against fraudulent player drop-out. In *2005 IEEE International Conference on Information Technology (ITCC 2005)*, volume 1, pages 500–506, Nevada, USA.

[García et al., 2006d] García, J., Cuppens, F., and Cuppens, N. (2006d). Analysis of policy anomalies on distributed network security setups. In *11th European Symposium On Research In Computer Security (Esorics2006)*, volume 4189 of *Springer LNCS*, pages 496–511, Hamburg, Germany.

[García et al., 2006e] García, J., Cuppens, F., and Cuppens, N. (2006e). Anomaly analysis of network access control policies. In *Spanish Meeting on Cryptology and Information Security (IX RECSI)*, Barcelona, Spain. *In Spanish*.

[García et al., 2006f] García, J., Cuppens, F., and Cuppens, N. (2006f). Towards filtering and alerting rule rewriting on single-component policies. In *25th Conference on Computer Safety, Reliability, and Security (Safecomp 2006)*, volume 4166 of *Springer LNCS*, pages 182–194, Gdansk, Poland.

[García et al., 2005e] García, J., Jaeger, M. A., Mühl, G., and Borrell, J. (2005e). Decoupling components of an attack prevention system using publish/subscribe. In *IFIP* *conference on Intelligence in Communication Systems*, volume 190 of *IFIP series*, pages 87–98, Montreal, Canada.

[Geer, 2004] Geer, D. (2004). Just how secure are security products? *IEEE Computer*, 37(6):14–16.

[Gombault and Diop, 2002] Gombault, S. and Diop, M. (2002). Response function. In *First Symposium on Real Time Intrusion Detection (NATO)*, Lisbon, Portugal.

[Gouda and Liu, 2004] Gouda, M. G. and Liu, A. X. (2004). Firewall design: consistency, completeness and compactness. In *24th IEEE International Conference on Distributed Computing Systems (ICDCS'04)*, pages 320–327.

[Gupta, 2000] Gupta, P. (2000). *Algorithms for Routing Lookups and Packet Classification*. PhD thesis, Stanford University, Department of Computer Science.

[Guttman, 1997] Guttman, J. D. (1997). Filtering postures: local enforcement for global policies. In *IEEE Symposium on Security and Privacy*, pages 120–129, Oakland, CA, USA.

[Hamed and Al-Shaer, 2006] Hamed, H. and Al-Shaer, E. (2006). Taxonomy of conflicts in network security policies. *IEEE Communications Magazine*, 44(3):134–141.

[Helmer et al., 2002] Helmer, G., Wong, J., Slagell, M., Honavar, V., Miller, L., Lutz, R., and Wang, Y. (2002). Software fault tree and colored petri net based specification, design and implementation of agent-based intrusion detection systems. Submitted to IEEE Transaction of Software Engineering.

[Herrera et al., 2004a] Herrera, J., J.García, and Perramn, X. (2004a). *Aspectos avanzados de seguridad en redes*. Fundaci Universitat Oberta de Catalunya. 370 p.

[Herrera et al., 2004b] Herrera, J., J.García, and Perramn, X. (2004b). *Seguretat en xarxes de computadors*. Fundaci Universitat Oberta de Catalunya. 283 p.

[Herrera et al., 2004c] Herrera, J., J.García, and Perramn, X. (2004c). *Seguridad en redes de computadores*. Fundaci Universitat Oberta de Catalunya. 287 p.

[Herzog and Shahmehri, 2002] Herzog, A. and Shahmehri, N. (2002). Using the java sandbox for resource control. In *7th Nordic Workshop on Secure IT Systems (NORDSEC)*, Linköpings Universitet, Linköping, Sweden.

[Hochberg et al., 1993] Hochberg, J., Jackson, K., Stallins, C., McClary, J. F., DuBois, D., and Ford, J. (1993). NADIR: An automated system for detecting network intrusion and misuse. *Journal of Computers and Security*, 12(3):235–248.

[Hofmeyr et al., 1998] Hofmeyr, S., Forrest, S., and Somayaji, A. (1998). Intrusion detection using sequences of system calls. *Journal of Computer Security*, 6(3):151–180.

[Hope, 2002] Hope, P. (2002). Using jails in freebsd for fun and profit. *Login; The Magazine of Usenix & Sage*, 27(3):48–55.

[Householder et al., 2002] Householder, A., Houle, K., and Dougherty, C. (2002). Computer attack trends challenge internet security. *IEEE Computer*, 35(4):5–8.

[Ilgun, 1993] Ilgun, K. (1993). USTAT: A Real-Time Intrusion Detection System for UNIX. In *IEEE Symposium on Security and Privacy*, pages 16–28, Oakland, CA, USA.

[Ilgun et al., 1995] Ilgun, K., Kemmerer, R. A., and Porras, P. A. (1995). State transition analysis: A rule-based intrusion detection approach. *IEEE Transactions on Software Engineering*, 21(3):181–199.

[International Organisation for Standardization, 1989] International Organisation for Standardization (1989). ITUT Rec. X.800—ISO 7498-2, Information processing systems-open systems interconnection - basic reference model - part 2: security architecture. ISO/ITU, 7498-2 edition.

[International Organisation for Standardization, 2000] International Organisation for Standardization (2000). ITU-T Recommendation X.509, The Directory: Public-key and Attribute certificate frameworks. International Telecommunication Union, Geneva.

[Javits and Valdes, 1993] Javits, H. and Valdes, A. (1993). The NIDES statistical component: Description and Justification. SRI Anual Report A010, SRI International, Computer Science Laboratory.

[Julisch, 2002] Julisch, K. (2002). Mining alarm clusters to improve alarm handling efficiency. In *17th Annual Computer Security Applications Conference (ACSAC)*, volume 2, pages 111–136.

[Kamara et al., 2003] Kamara, S., Fahmy, S., Schultz, E., Kerschbaum, F., and Frantzen, M. (2003). Analysis of vulnerabilities in Internet firewalls. *Journal of Computers and Security*, 22(3):214–232.

[Kemmerer and Vigna, 2002] Kemmerer, R. and Vigna, G. (2002). Intruder detection: A brief history and overview. *IEEE Computer*, 35(4):27–30.

[Kemmerer, 1997] Kemmerer, R. A. (1997). NSTAT: A model-based real-time network intrusion detection system. Technical Report TRCS97-18, Reliable Software Group, Department of Computer Science, University of California Santa Barbara.

[Ko et al., 1997] Ko, C., Ruschitzka, M., and Levitt, K. (1997). Execution monitoring of security-critical programs in a distributed systems: A specification-based approach. In *IEEE Symposium on Security and Privacy*, pages 175–187, Oakland, CA, USA.

[Kruegel, 2002] Kruegel, C. (2002). *Network Alertness - Towards an adaptive, collaborating Intrusion Detection System.* PhD thesis, Technical University of Vienna.

[Kruegel and Toth, 2002] Kruegel, C. and Toth, T. (2002). Flexible, mobile agent based intrusion detection for dynamic networks. In *European Wireless*, Italy.

[Kruegel et al., 2005] Kruegel, C., Valeur, F., and Vigna, G. (2005). *Intrusion Detection and Correlation: Challenges and Solutions*. Springer, first edition.

[Kumar and Spafford, 1994] Kumar, S. and Spafford, E. H. (1994). A pattern matching model for misuse intrusion detection. In *17th National Computer Security Conference*, pages 11–21.

[Lee et al., 1999] Lee, W., Stolfo, S. J., and Mok, K. W. (1999). A data mining framework for building intrusion detection models. In *IEEE Symposium on Security and Privacy*, pages 120–132, Oakland, CA, USA.

[Lindqvist and Porras, 1999] Lindqvist, U. and Porras, P. A. (1999). Detecting computer and network misuse through the production-based expert system toolset (P-BEST). In *IEEE Symposium on Security and Privacy*, pages 146–165, Oakland, CA, USA.

[Lippmann et al., 2000] Lippmann, R., Haines, J., Fried, D., Korba, J., and Das, K. (2000). The 1999 DARPA off-line intrusion detection evaluation. *Computer Networks*, (34):579–595.

[Lonvick, 2001] Lonvick, C. (2001). The BSD Syslog Protocol. Rfc3164.

[Loscocco and Smalley, 2001] Loscocco, P. and Smalley, S. (2001). Integrating flexible support for security policies into the linux operating system. In *11th FREENIX Track: 2001 USENIX Annual Technical Conference*, USA.

[Lunt et al., 1990] Lunt, T., Tamaru, A., Gilham, F., Jagannathan, R., Neumann, P. G., and Jalali, C. (1990). IDES: A progress report. In *6th Annual Computer Security Applications Conference*, Tucson, AZ, USA.

[McVoy and Staelin, 1996] McVoy, L. and Staelin, C. (1996). LMbench, portable tools for performance analysis. In *USENIX 1996 Annual Technical Conference*, San Diego, CA, USA.

[Migus, 2006] Migus, A. C. (2006). IDMEF XML library version. `http://sourceforge.net/projects/libidmef/`.

[MITRE Corporation, 2005] MITRE Corporation (2005). Common vulnerabilities and exposures. `http://cve.mitre.org/`.

[Moore et al., 2001] Moore, B., Ellesson, E., Strassner, J., and Westerinen, A. (2001). Policy core information model – version 1 specification. Request for comments 3060.

[Morin et al., 2002] Morin, B., Mé, L., Debar, H., and Ducassé, M. (2002). M2D2: a formal data model for intrusion alarm correlation. In *Fifth International Symposium on Recent Advances in Intrusion Detection (RAID2002)*, Zurich, Switzerland.

[Mounji, 1997] Mounji, A. (1997). *Languages and Tools for Rule-Based Distributed Intrusion Detection*. PhD thesis, University of Namur, Belgium.

[Mounji et al., 1995] Mounji, A., Charlier, B. L., Zampunieris, D., and Habra, N. (1995). Distributed audit trail analysis. In *Symposium on Network and Distributed System Security*, pages 102–112.

[Mühl, 2002] Mühl, G. (2002). *Large-Scale Content-Based Publish/Subscribe Systems*. PhD thesis, Technical University of Darmstadt.

[Ning et al., 2002] Ning, P., Cui, Y., and Reeves, D. S. (2002). Analyzing intensive intrusion alerts via correlation. In *Fifth International Symposium on Recent Advances in Intrusion Detection (RAID2002)*, pages 74–94, Zurich, Switzerland.

[Ning and Xu, 2003] Ning, P. and Xu, D. (2003). Learning attack strategies from intrusion alerts. In *10th ACM Conference on Computer and Communication Security*, pages 200–209, Washington DC, USA.

[Northcutt, 2002] Northcutt, S. (2002). *Network Intrusion Detection: An analyst's Hand Book*. New Riders Publishing, third edition.

[Onabuta et al., 2001] Onabuta, T., Inoue, T., and Asaka., M. (2001). A protection mechanism for an intrusion detection system based on mandatory access control. In *13th Annual Computer Security Incident Handling Conference*, Toulouse, France.

[Open Security Foundation, 2005] Open Security Foundation (2005). Open source vulnerability database. `http://www.osvdb.org/`.

[Ott, 2002] Ott, A. (2002). The role compatibility security model. In *7th Nordic Workshop on Secure IT Systems (NORDSEC)*, Linköpings Universitet, Linköping, Sweden.

[Petkac and Badger, 1997] Petkac, M. and Badger, L. (1997). Security agility in response to intrusion detection. In *16th Annual Computer Security Applications Conference (ACSAC)*, New-Orleans, LA, USA.

[Pietzuch, 2004] Pietzuch, P. (2004). *Hermes: A Scalable Event-Based Middleware*. PhD thesis.

[Porras and Neumann, 1997] Porras, P. A. and Neumann, P. G. (1997). EMERALD: Event monitoring enabling responses to anomalous live disturbances. In *20th National Information Systems Security Conference*, pages 353–365.

[Powell, 1996] Powell, D. (1996). Group communication. *Communications of the ACM*, 39(4):50–53.

[Queiroz et al., 1999] Queiroz, J. D., da Costa Carmo, L. F. R., and Pirmez., L. (1999). Micael: An autonomous mobile agent system to protect new generation networked applications. In *2nd Annual Workshop on Recent Advances in Intrusion Detection*, Purdue, IN, USA.

[Reed, 2005] Reed, D. (2005). Ip filter. `http://www.ja.net/CERT/Software/ip-filter/ip-filter.html`.

[Roesch, 1999] Roesch, M. (1999). Snort: Lightweight intrusion detection for networks. In *13th Conference on Systems Administration*, pages 229–238. USENIX Association.

[Ruff, 2006] Ruff, M. (2006). XmlBlaster: open source message oriented middleware. `http://www.xmlblaster.org/`.

[Schneier, 1996] Schneier, B. (1996). *Applied Cryptography*. John Wiley and Sons, second edition.

[Sekar et al., 2002] Sekar, R., Gupta, A., Frullo, J., Shanbhag, T., Tiwari, A., Yang, H., and Zhou, S. (2002). Specification-based anomaly detection: A new approach for detecting network intrusions. In *9th ACM conference on computer and communications security*, pages 265–274, Washington, DC, USA.

[Snapp et al., 1991] Snapp, S. R., Brentano, J., Dias, G. V., Goan, T. L., Heberlein, L. T., Ho, C., K. N. Levitt, Mukherjee, B., Smaha, S. E., Grance, T., Teal, D. M., and Mansur, D. (1991). DIDS (distributed intrusion detection system) - motivation, architecture and an early prototype. In *14th National Security Conference*, pages 167–176.

[Spafford, 1991] Spafford, E. (1991). The Internet Worm Incident Technical Report CSD-TR-933. Department of Computer Sciences, Purdue University, West Lafavette, USA.

[Spafford and Zamboni, 2000] Spafford, E. H. and Zamboni, D. (2000). Intrusion detection using autonomous agents. *Computer Networks*, 34(4):547–570.

[Srinivasan et al., 1999] Srinivasan, V., Suri, S., and Varghese, G. (1999). Packet classification using tuple space search. *Computer ACM SIGCOMM Communication Review*, 29(4):135–146.

[Stallings, 1995] Stallings, W. (1995). *Network and internetwork security: principles and practice*. Prentice-Hall, Inc. Upper Saddle River, NJ, USA.

[Stallings, 2002] Stallings, W. (2002). *Network Security Essentials: Applications and Standards*. Prentice Hall Professional Technical Reference.

[Staniford-Chen et al., 1996] Staniford-Chen, S., Cheung, S., Crawford, R., Dilger, M., Frank, J., Levitt, J. H. K., Wee, C., Yip, R., and Zerkle, D. (1996). GrIDS – a graph-based intrusion detection system for large networks. In *19th National Information Systems Security Conference*.

[Teng et al., 1990] Teng, H. S., Chen, K., and Lu, S. C. (1990). Adaptive real-time anomaly detection using inductively generated sequential patterns. In *IEEE Symposium on Security and Privacy*, pages 278–284, Oakland, CA, USA.

[Valdes and Skinner, 2001] Valdes, A. and Skinner, K. (2001). Probabilistic alert correlation. In *Fourth International Symposium on Recent Advances in Intrusion Detection (RAID2001)*, pages 58–68, Davis, CA, USA.

[Veillard, 2006] Veillard, D. (2006). The XML C library for Gnome (libxml). `http://www.xmlsoft.org`.

[Viega and McGraw, 2002] Viega, J. and McGraw, G. (2002). *Building Secure Software - How to Avoid Security Problems the Right Way*. Addison-Wesley.

[Vigna and Kemmerer, 1998] Vigna, G. and Kemmerer, R. A. (1998). NetSTAT: A network-based intrusion detection approach. In *14th Annual Security Applications Conference*, pages 25–34, Scottsdale, AZ, USA. IEEE Press.

[Vigna and Kemmerer, 1999] Vigna, G. and Kemmerer, R. A. (1999). NetSTAT: A network-based intrusion detection system. *Journal of Computers and Security*, 7(1):37–71.

[Welte et al., 2006] Welte, H., Kadlecsik, J., Josefsson, M., McHardy, P., and et. al (2006). The Netfilter project: firewalling, NAT, and packet mangling for linux 2.4. `http://www.netfilter.org/`.

[White et al., 1999] White, G. B., Fisch, E. A., and Pooch, U. W. (1999). Cooperating security managers: A peer-based intrusion detection system. *IEEE Network*, 7:20–23.

[Wright et al., 2002] Wright, C., Cowan, C., Smalley, S., Morris, J., and Kroah-Hartman, G. (2002). Linux Security Modules: General security support for the linux kernel. In *11th USENIX Security Symposium*, San Francisco, CA, USA.

[Wu et al., 1999] Wu, S., Chang, H., Jou, F., Wang, F., Gong, F., Sargor, C., Qu, D., and Cleaveland, R. (1999). JiNao: Design and implementation of a scalable intrusion detection system for the OSPF routing protocol. *Journal of Computer Networks and ISDN Systems*.

Joaquín García Alfaro

Bellaterra, September 2006