

N° d'ordre : 2007telb0057

THÈSE

Présentée à

l'ÉCOLE NATIONALE SUPÉRIEURE DES TELECOMMUNICATIONS DE BRETAGNE

en habilitation conjointe avec l'Université de Rennes 1

pour obtenir le grade de

DOCTEUR de l'ENST Bretagne

Mention « Informatique »

par

Yohann THOMAS

« Policy-Based Response to Intrusions Through Context Activation »

Soutenue le 23 novembre 2007 devant la Commission d'Examen :

Composition du Jury

- *Rapporteurs* : Olivier FESTOR, Directeur de Recherche, LORIA-INRIA Lorraine
Christopher KRUEGEL, Assistant Professor, Technical University Vienna
- *Examineurs* : Mireille DUCASSE, Professeur, INSA/IRISA Rennes
Frédéric CUPPENS, Professeur, GET/ENST Bretagne
Hervé DEBAR, Expert Emérite, Orange Labs
Nora CUPPENS, Enseignant-Chercheur, GET/ENST Bretagne
Olivier PAUL, Maître de Conférence, INT Evry
- *Invité* : Patrick RADJA, Responsable de la R&D, EADS DS/DCS/IIS

Résumé

Nous présentons dans cette thèse une nouvelle approche de réponse face aux menaces auxquelles les systèmes informatiques sont soumis. Cette approche est basée sur l'intégration de la notion de contre-mesure au sein même de la politique de sécurité. En particulier, la notion de contexte permet d'évaluer l'état courant du système, et d'exprimer la politique en fonction de cet état. Pour ce faire, le modèle de contrôle d'accès basé sur l'organisation (Or-BAC) est utilisé, distinguant la définition générique de la politique de son implémentation effective en fonction du contexte.

Le contexte intègre aussi bien des paramètres spatiaux et temporels que des paramètres plus proprement liés au domaine de la sécurité opérationnelle, comme les alertes remontées par les systèmes de détection d'intrusions (IDS). Ces alertes permettent la caractérisation de la menace à laquelle est soumis le système d'information à un instant donné. Des contextes de menace sont instanciés par notre système de réponse, permettant de déclencher des mises à jour de la politique et son déploiement subséquent. Ainsi, le système est capable d'adapter dynamiquement ses paramètres de fonctionnement en considérant notamment la menace.

Nous proposons une approche innovante établissant le lien entre la politique de sécurité et l'un des principaux moyens qui permet d'en contrôler le respect, à savoir les systèmes de détection d'intrusions. Ce lien n'existait pas jusqu'alors, c'est-à-dire que les violations de la politique de sécurité détectées par les IDS n'avaient que peu de conséquences sur les exigences de la politique de sécurité effectivement implémentées par les points d'application. Pourtant, force est de constater que l'implémentation de la politique ne doit pas être statique. En particulier, nous montrons qu'il est possible de gérer dynamiquement l'accès aux services et aux ressources en fonction de la menace.

En outre, ce travail fournit un début de réponse à la problématique de la réactivité et de la pertinence de la réponse face aux menaces. La réponse aux attaques informatiques est le plus souvent gérée manuellement par l'opérateur de sécurité. Ce même opérateur de sécurité manque malheureusement bien souvent de réactivité et de discernement pour répondre de manière adéquate à la menace, notamment parce qu'il est bien souvent noyé sous le flot des alertes ; le travail d'analyse est fastidieux et difficile au vu du nombre de paramètres à considérer. D'un autre côté, les attaques se multiplient, les attaquants mettent de moins en moins de temps à pénétrer les systèmes et à produire des dégâts qui peuvent rapidement se chiffrer en millions d'euros pour les entreprises. Automatiser la réponse est donc une nécessité.

Abstract

We present in this thesis a new approach to respond to malicious events threatening information systems. This approach integrates the notion of response at the security policy level. We build upon the Organization-Based Access Control model (Or-BAC), which distinguishes generic policy definition from its actual instantiation depending on contextual conditions. The notion of context allows to assess the current state of the system, and to accordingly express the policy at policy enforcement points level.

Context includes spatial and temporal parameters, as well as parameters especially dealing with operational security, like alerts reported by intrusion detection systems (IDS). Alerts provide means to characterize current threat towards the information system. Threat contexts are instantiated by our system, triggering updates of the policy instantiation. Thus, the system is able to dynamically adapt to threat by adjusting its configuration.

We propose an new approach to bridge the link between the security policy and one of the components controlling its fulfillment, that is intrusion detection systems. This link had not been yet established, *i.e.* violations of the security policy detected by IDS do not lead to any variation of the actual implementation of the security policy at policy enforcement points level. In particular, we show that it is possible to manage dynamic paths to services and resources depending on threats.

In addition, this work provides a preliminary answer to the problematic of reactivity and relevancy of response to threat. The security operators dealing with response to attacks suffer from a lack of reactivity. They are overwhelmed through the flow of alerts, and the analysis process is particularly tedious given the number of parameters to consider. In addition, there are more and more attacks, and time required for compromising services or resources has greatly decreased, which may lead to disastrous effects, especially financial losses for corporations, which may rapidly represent million euros. Automating response is therefore a necessity.

The proposed system supports fail-safe dynamic security policies, making the best use of both monitoring and enforcement security components.

Remerciements

Mes premières pensées vont aux auteurs des prémices de cette histoire, mes parents. Maman, papa, vous m’avez aidé à écrire mes premières lignes. J’ai parfois râturé, vous avez corrigé, m’avez montré le chemin, je l’ai suivi, à mon rythme, en prenant parfois quelques sentiers détournés, mais je suis arrivé sereinement au premier embranchement, celui où les gens oublient parfois qu’ils ont été des enfants. Je n’oublie pas. Papa, tu m’as initié aux plaisirs du clavier et de l’écran, du disque dur et de la barrette, de la carte mère et du CPU ; maman, tu as su m’en relever la tête pour m’éviter de sombrer dans les abîmes d’un monde à tes yeux trop virtuel. Tu m’as aidé à garder les pieds sur Terre. Merci à vous deux, et merci à ma petite soeur, Hélène.

Ce monde virtuel, il m’a pourtant suivi, fait cependant de rencontres et d’expériences bien réelles. Et si *la vie, c’est d’abord des rencontres*, je tiens à témoigner ma gratitude, par ordre chronologique, envers ceux qui m’ont placé sur le chemin que j’emprunte aujourd’hui : Ludo, parce que ce sont tes interventions éclairées qui ont initié le petit ingénieur en herbe que j’étais au vaste domaine de la sécurité. Benjamin, car c’est toi qui m’a permis de découvrir le monde de la recherche. Je n’oublie pas ces quelques réunions mémorables au QG de NSS (le Café Latin). Bien évidemment, je remercie celui qui m’a véritablement mis le pied à l’étrier, Hervé. Merci de m’avoir accordé ta confiance et de m’avoir proposé cette longue et délicate aventure. J’ai vu passer ton voilier, je n’ai pas hésité, j’ai sauté dans ma barque, et j’ai ramé ! *Le rivage est plus sûr, mais [je crois que] j’aime me battre avec les flots.*

Merci ensuite à Fred, d’avoir accepté d’être mon guide, le phare vers lequel orienter mon embarcation de fortune, mon directeur, sans même me connaître au préalable. Or-BAC a été mon Dieu, j’en suis désormais un nouveau messager. De Caen à Rennes, j’ai suivi ta lumière, elle m’a mené également à Nora. Merci Nora pour ta bonne humeur, ta franchise, la pertinence et la justesse de tes remarques. Quelques mots également pour remercier tous les membres de l’équipe SERES de l’ENST Bretagne, et tout particulièrement Céline, pour l’accueil qui m’a été réservé à chacun de mes passages à Rennes.

Un grand merci à Christopher Kruegel et Olivier Festor d’avoir accepté de rapporter cette thèse. Merci à Mireille Ducassé d’avoir accepté de présider ce jury. Merci enfin à Olivier Paul et Patrick Radja, qui m’ont fait l’honneur de compléter ce jury.

Une pensée pour tous les collègues de Caen. Merci Fabrice pour ton accueil au sein de l’URD SPR du laboratoire NSS du CRD MAPS de France Télécom R&D / France Télécom - Division R&D / Orange Labs (ex-CNET / ex-SEPT). Et puis dans le désordre, et pas par discipline : Iwen, co-bureau, ta pancarte “désolé” me manquera,

ou pas...;-) ; Cécile, woow, Fffffff, trop de choses à dire, alors en un mot : merci :-) ; Pierre, "Salut Pierre !", dis-moi quand tu retournes à la love parade ; Emeline, Evelyne, pour le rosé et la barre de fer ; Jouni, mon Finlandais préféré, Kippis ! Merci pour ton calme à toute épreuve et tes paroles rassurantes quand j'ai douté ; Vince, moonchack, rimoute, le requin au grand coeur ; Diala, pour les pétages de bide, l'huile, et pour les conseils avisés pendant ma préparation de soutenance. Je pense également à ceux qui ont vogué vers d'autres horizons : Elvis, si tu veux toujours te battre, c'est pas trop tard ; Benoît, ton rire m'a manqué ; Fab, merci pour le bureau, trop classes tes baskets ; Tib, tout va bien au pays des cuillères tordues ? Un clin d'oeil également à Valérie. Et tous les autres, je ne les oublie pas.

Et parce qu'il n'y pas que le boulot dans la vie, un grand merci à tous ceux qui en coulisses m'ont accompagné dans cette aventure. Les Caennaises et Caennais : Elo, les bouffes entre amis, les weekends, la Tunisie, tu n'y es pas pour rien dans la logistique en coulisses, et ça a compté ; Claire, quand même, l'infirmière qui déchire (les sutures). Comme dirait Bob : "Me voilà, promotion !". Enfin docteur, mais pour ordinateurs, la même passion, mais pas le même maillot ! ;-) Ben, en deux mots : Apéro ? Apéro ! Benjamin, papa, vive l'intercontrat ! Frocky, une petite mousse ? Et les potes d'école : Arno, pour ton oreille et ta patience, pour ton "fauteuil" à Rennes, et ton oreille, encore merci ; Alex, pour le pied à terre à Paris au retour de missions, et pour les cocktails au Sister May ; Rem, un jour, on se fera une Bolino Party, et on finira de raser Boubou, gavés à l'Ichouchtar ; Flo, pour les vacances à Notre-Dame, j'aurais voulu être là cette année. On remet ça ? Et Fab, la force du binôme, je crois qu'elle m'a parfois manqué.

Durant ces trois années, j'ai parfois rechigné à demander mon chemin, m'efforçant tant bien que mal de le trouver par moi-même. Parfois, j'ai traîné en route, parfois même, je me suis perdu, mais au fond, la vie n'est pas une course, peu importe l'arrivée, c'est le chemin qui compte. Et puis, comme le dit si bien mon ami Winnie l'Ourson : "*Il n'y a pas d'urgence, nous y arriverons un jour.*" Avec la satisfaction d'y être finalement arrivé, je me retourne, et je me dis que le chemin était beau et qu'il en valait la peine. Il est maintenant temps pour moi de remonter dans ma barque fraîchement équipée d'une toute nouvelle paire de rames, et de voguer vers de nouvelles aventures.

Avant de reprendre le large, j'adresse une pensée toute particulière à mon grand-père paternel, qui s'est envolé pour d'autres cieux alors même que ma vie caennaise commençait. Papy, ce chemin, je ne l'ai pas fait sans toi. Tu étais là, et tu seras toujours là, car je suis fait d'un peu de toi. Je pense à toi, où que tu sois, là-bas...

Contents

Résumé	i
Abstract	iii
Remerciements	v
Contents	ix
List of Figures	xii
1 Introduction	1
1.1 Motivation	1
1.2 Objectives	2
1.3 Statement	3
1.4 Outline	3
2 Related Work	5
2.1 Introduction	5
2.2 Intrusion Detection	6
2.2.1 Overview of Intrusion Detection	6
2.2.2 Diagnosis enhancement	11
2.3 Intrusion and Threat Response	13
2.3.1 Problem statement	13
2.3.2 Intrusion response taxonomies	15
2.3.3 Implementing threat response	21
2.4 Security Policies and Access Control	23
2.4.1 Security Policies	23
2.4.2 Access Control Models	28
2.5 Conclusion	32
3 Proposal of a Threat Response System	33
3.1 Introduction	33
3.2 Architecture of the Threat Response System	35
3.2.1 Proposed architecture	35
3.2.2 Role of the components	37

3.2.3	Scalability of the architecture	41
3.3	Multiple Paths to Resources	42
3.3.1	Host level	43
3.3.2	Service / Application level	43
3.3.3	Information level	44
3.4	Conclusion	45
4	Use of Or-BAC for threat response	47
4.1	Introduction	47
4.2	The Or-BAC model	48
4.2.1	Or-BAC advantages for threat response	48
4.2.2	Operational contexts	49
4.2.3	Mapping Or-BAC onto our architecture	51
4.3	Or-BAC extensions for threat response	52
4.3.1	Threat contexts	53
4.3.2	Minimal contexts	58
4.3.3	Context composition	58
4.3.4	Conflict resolution and priorities assessment	60
4.4	Conclusion	66
5	Application: email use case	67
5.1	Introduction	67
5.2	Presentation of the email use case	68
5.3	Modeling roles	72
5.4	Modeling views	74
5.5	Modeling activities	76
5.6	Modeling contexts	79
5.6.1	Operational contexts	79
5.6.2	Threat contexts	80
5.6.3	Minimal contexts	81
5.7	Modeling policy	82
5.7.1	Operational policy	82
5.7.2	Reaction policy	83
5.7.3	Minimal policy	87
5.8	Modeling structured entities	88
5.9	Conclusion	92
6	Mapping Policy Instances onto Alerts: Response Strategy	93
6.1	Introduction	93
6.2	Response strategy taxonomy	96
6.2.1	Direction	96
6.2.2	Target layer	97
6.2.3	Scale of response	98
6.3	Information to consider	99
6.3.1	Information available into alerts	99

6.3.2	Information available from models	101
6.3.3	Information deduced from alerts	101
6.4	Conclusion	107
7	Implementation	109
7.1	Introduction	109
7.2	Implementation case study	110
7.2.1	Modified email use case	110
7.2.2	Definition of the security policy	111
7.3	Policy Instantiation Engine	113
7.3.1	Objectives	114
7.3.2	Prolog response engine	114
7.3.3	Perl sequencer	120
7.3.4	Experiment: injecting a set of alerts	122
7.4	Policy Decision Point	127
7.4.1	Objectives	127
7.4.2	Considered countermeasures	127
7.4.3	Policy enforcement	128
7.5	Conclusion	131
8	Discussion	133
8.1	Implementing and improving the current model	133
8.1.1	About strategy improvements	133
8.1.2	Adaptive response deactivation	136
8.1.3	About vertical dependencies in the three layer model	136
8.1.4	Dynamic management of minimal contexts	137
8.1.5	Extend the approach to other PEPs	137
8.2	Consistent cartography model	138
8.2.1	Information of interest	138
8.2.2	About the use of cartography for threat response	140
8.3	Considerations about the policy	145
8.3.1	About a cost-sensitive model	145
8.3.2	About deployed policy validation	146
8.3.3	About policy simulation	147
9	Conclusion and Perspectives	149
A	Glossary	159

List of Figures

2.1	IDWG functional schema of an IDS	6
2.2	IDMEF data model overview	10
2.3	Fisch Intrusion Response Taxonomy	15
2.4	Carver Intrusion Response Taxonomy	16
2.5	Stakhanova et al. Intrusion Response Systems Taxonomy	17
2.6	Context based security (Brézillon and Kouadri)	25
2.7	Generic AAA Server Interactions	27
2.8	Abstraction of access control entities in Or-BAC	31
3.1	John R. Boyd's OODA strategy model	34
3.2	Mapping our requirements onto AAA	35
3.3	Mapping our requirements onto OODA	36
3.4	Threat response system generic architecture	37
3.5	Scalability of the architecture	42
4.1	Threat response system architecture	52
4.2	Or-BAC model for threat context examples	56
5.1	Email use case: simplified layer model	68
5.2	Simplified network datagram structure	69
5.3	Email use case: layer model with infrastructure components	70
5.4	Email use case: logical architecture	71
5.5	Email use case: physical architecture	72
5.6	Email use case: roles modeling	73
5.7	Email use case: views modeling	75
5.8	Email use case: activities modeling	76
5.9	Email use case: logical architecture, actions and activities	78
5.10	Email use case: contexts modeling	79
5.11	Email use case: layer model for the Read activity	84
5.12	XML schema of the IDMEF Node class	88
5.13	Modeling structured subjects	89
5.14	Modeling structured objects	90
5.15	Modeling structured actions	91
6.1	Evaluation concepts and relationships in the Common Criteria	94
6.2	Detailed view of the PIE	95

6.3	Response strategy taxonomy	96
7.1	Implementation workflow	110
7.2	Implementation: case study layout	111
7.3	Implementation: Or-BAC abstract and concrete entities	112
7.4	Implementation: Threat & Response Characterization Engine workflow	113
7.5	Experimentation: sequence of alerts and associated contexts	123
8.1	Vertical dependencies in the three layer model	136
8.2	Correlation with vulnerabilities	141
8.3	Alert severity mitigation	142
8.4	False positive recognition	143
8.5	Threat response system with simulation component	147

Chapter 1

Introduction

1.1 Motivation

Computer systems and networks are playing a more and more significant role in our everyday life, for business as well as medical applications, or even personal use, for instance e-commerce. As a matter of fact, information systems management is becoming more and more complex, especially regarding security. In particular, one has to cope with an unceasingly increasing number of interconnections of hosts and networks. Moreover, additional requirements, such as service personalization and privacy, as well as new usages, especially linked with nomadism, and needs for service convergence have put the light on new security issues in a context where costs must be more and more controlled, providing performance as well as convenience to users.

Ensuring security is generally realized through the preservation of confidentiality and integrity of the resources, and availability of the resources and services. This means one should not be able to access a resource without being authorized, nor modify it. This also means that available resources and services should not be rendered unavailable to users when they need to access them. For this purpose, a security policy is defined, consisting of a set of requirements describing what should be done to ensure security.

Security policies are often not formally defined in operational networks. A list of requirements is formulated, and security equipments and methods are deployed, but without any validation of the global coherence and conformance of the actual deployment compared to what is expected in term of security. At best, when formally defined, security policies are generally static, that is they are defined once at design time, and do not take into account any contextual parameter. This means that security rules remain unchanged whatever the time, the location of the observed events, or even the threat level.

Security is currently provided by many different but complementary solutions, including preventive and corrective means. Authentication, encryption and access control are used to prevent confidentiality and integrity breaches. Firewalls are deployed for a better control of the traffic between different networks. Corrective means, such as antiviruses and intrusion detection systems, are also deployed to complement the prevention approach.

Our proposal is based on the fact that formally defined security policies exist (*e.g.* Or-BAC), that security monitoring exists (*e.g.* intrusion detection), but that they are currently totally uncorrelated one from another, that is security requirements are not linked with the means used to control their fulfillment.

1.2 Objectives

In this thesis, we establish the aforementioned connection between security monitoring and security policies. We propose the use of a formally defined security policy not only to describe “traditional” security requirements, such as access control and encryption methods, but also to provide response to threat. Threat is characterized with the use of intrusion detection systems, which report alerts, representing attacks, which sometimes lead to actual intrusions. The objective is to provide a new approach enabling automated response to threat through the use of a dynamic security policy. Our proposal thus relies on three major principles: automated response, dynamic response, and policy-level response.

Automated response. The objective of this thesis is partly based on the fact that security administrators, as human beings, lack the reactivity to launch appropriate response to threat. This lack of reactivity is one of the major issues that our work aims at solving, by providing automation in the response process. However, automated response to threat implies that threat characterization is reliable, since the contrary may lead to disastrous and harmful side-effects, such as self-inflicted denial-of-service. For this purpose, we thus make the assumption that intrusion detection diagnoses are reliable, especially that they do not provide false positives to our response system. Although this may seem a strong assumption, we do believe that recent advances as well as work in progress in the area of intrusion detection, especially dealing with alert correlation, allows to envision the design of an automated response system. We show in the related work section that we have good reasons to make this assumption.

Dynamic response. Automated response to threat is not a new area of research, but the fact is that most existing approaches are not very convincing, since they often rely on static mappings, linking characterized threats (or detected intrusions) to specific countermeasures. This means that every each alert reported by an intrusion detection system is triggering a static response, without taking into account any contextual data, which would reflect the state of the system at a given moment. From our point of view, considering context parameters is a crucial issue, since a same alert may require two different responses in two different contexts. In particular, a context parameter is the time. For example, one may prefer to preserve availability during working hours, and to focus on confidentiality and integrity during non-working hours. Our second contribution thus relies on the dynamic behavior of our threat response system. For this purpose, context parameters include alerts reported by intrusion detection systems, so that the system is able to provide response accordingly to characterized threats.

Policy-level response. The last major point of our contribution is the integration of the threat response system at the policy level. The policy must obviously consider security, but also additional parameters, such as performance - *e.g.* being able to serve more and more users - and convenience - *e.g.* providing ease-of-use. However, elevating security often leads to downgrading performance, for instance through the use of stronger encryption algorithms, requiring more resources to be processed, as well as degrading convenience, using for example stronger authentication means (*e.g.* certificates instead of simple passwords). In fact, we face here a problem of correctly balancing these parameters depending on the current context. Consequently, we wonder whether the response functionality could be directly provided at policy design time, resulting in a continuous assessment of different parameters through activated contexts, enabling thus to configure the resources according to performance and convenience requirements, but also taking into account the threat level. This means that threat response is seen as a re-evaluation of the policy given threat contexts, which dynamically enables adequate policy rules to be deployed over the information system. These policy rules, designed *a priori* to mitigate the threat, should also guarantee an acceptable level of performance and convenience, to fit the compromise between the considered variables.

1.3 Statement

Given the objectives, we first aim at leveraging information available in the alerts reported by intrusion detection systems to build new policy elements, *e.g.* threat contexts. These new policy elements allow the specification of a dynamic security policy, automatically adjusting to contextual requirements. We then show that it is possible to build an architecture based on this approach, providing response to threat by enabling dynamic policy decisions and enforcement considering threats. Finally, we bring up the possibility to express different mapping strategies between alerts and effective reactions, so that effects may change to allow the best adequate response to threat.

1.4 Outline

This thesis is organized as follows: chapter 2 presents related work, focusing on what is necessary to support our proposal, as well as the assumptions it relies on. We remind some basic principles about intrusion detection, give elements about existing threat response approaches, and provide an overview of security policies and access control models. Chapter 3 deals with our proposal of a threat response system. We present the architecture we developed and explain how we manage availability requirements thanks to multiple paths to resources. We then describe in chapter 4 how we make use of the Organization-Based Access Control (Or-BAC) model to provide response to threat through our architecture. Chapter 6 explains how we manage mapping from alerts to countermeasures to provide adequate response to threat. We propose in chapter 7 an implementation of a threat response system based on the aforementioned requirements. Discussion is provided in chapter 8, especially regarding suitable improvements, and chapter 9 concludes this thesis.

Chapter 2

Related Work

Contents

2.1	Introduction	5
2.2	Intrusion Detection	6
2.2.1	Overview of Intrusion Detection	6
2.2.2	Diagnosis enhancement	11
2.3	Intrusion and Threat Response	13
2.3.1	Problem statement	13
2.3.2	Intrusion response taxonomies	15
2.3.3	Implementing threat response	21
2.4	Security Policies and Access Control	23
2.4.1	Security Policies	23
2.4.2	Access Control Models	28
2.5	Conclusion	32

2.1 Introduction

In this chapter, we present the related work, dealing with three main fields of research: intrusion detection, intrusion and threat response, and security policies.

We give in a first section a brief overview of intrusion detection, putting the light on alert reporting and diagnosis enhancement, since we aim at providing reliable alerts correctly formatted to our threat response system.

A second section presents a state-of-the-art of intrusion and threat response. Vocabulary in the field of response is discussed, current response taxonomies are explained, and limitations are brought up as new points we would like to consider. Finally, we have a look at threat response implementation, distinguishing the prevention approach from the detection approach. We also provide a simple classification of the means to respond.

In a third section, we focus on security policies, restricting to the notions which are interesting for our work, especially dynamic access control policies.

2.2 Intrusion Detection

2.2.1 Overview of Intrusion Detection

Intrusion detection aims at revealing malicious activity occurring on an information system. IDS sensors collect data at different levels over the information system to detect attack attempts, sometimes leading to actual intrusions (successful attacks). Intrusion detection systems belong to a category of tools aiming at monitoring information systems in order to ensure confidentiality and integrity of the resources and communications, and to preserve availability of services.

The origin of intrusion detection is generally associated with Anderson [6] and Denning [37] in the 1980s. However, lots of works have been realized since, and recent works in the field of intrusion detection have led to the creation of the *Intrusion Detection Working Group* (IDWG) of the *Internet Engineering Task Force* (IETF), which proposed a normalization of intrusion detection systems. The IDWG aims at defining data formats and exchange procedures [88] for sharing information between entities composing intrusion detection systems. The *Intrusion Detection Message Exchange Format* (IDMEF) [31] defines a standard format for alert reporting.

The IDWG proposed an architecture for intrusion detection systems, as shown in figure 2.1, consisting of three main entities: a *sensor*, an *analyzer* and a *manager*. Data is collected thanks to activity monitored on the information system by *sensors*. *Sensors* report relevant information as *events* to the *analyzer*, in charge of the processing allowing *alerts* characterization. *Alerts* are then sent to the *manager*, which deals with alert reporting and potential response functionalities.

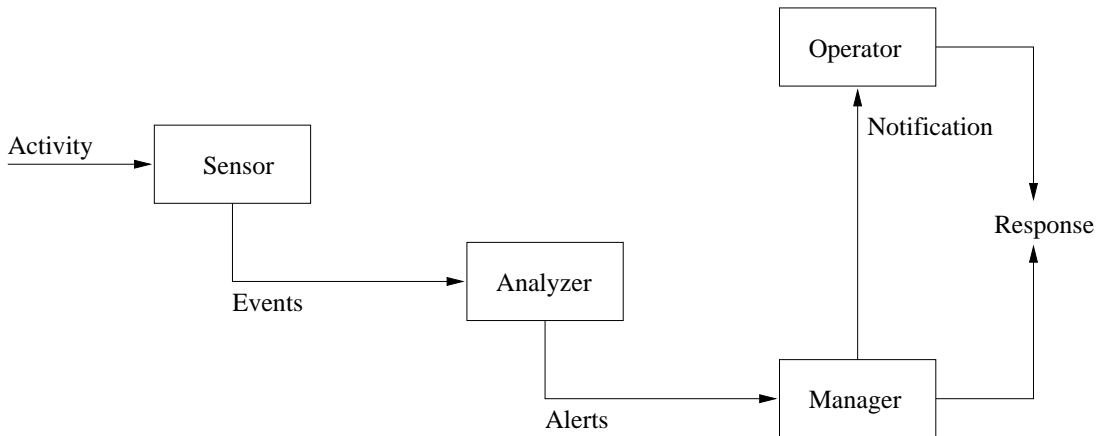


Figure 2.1: IDWG functional schema of an IDS

Figure 2.1 also mentions the security *operator*, receiving *notifications* from the *manager*. Note that threat response is envisioned both at the *operator* level and at the *manager* level. At the *operator* level, responses are applied manually, whereas automation of response selection may be envisioned at the *manager* level.

2.2.1.1 Data collection

Data collection is the first task of intrusion detection systems. Data sources can be classified in three different categories: network packets, host log files and application log files.

Network packets. Observing the network traffic (*a.k.a.* packet sniffing) is a popular and easy way to provide data for intrusion detection systems. Protocol decoding is used to extract relevant data from the observed traffic, in order to further process these data to find out malicious activity. This data source suffers from a few drawbacks. First, sensors location should allow to observe as many communications as possible (*e.g.* entry to server farms or next to proxy servers). Secondly, sensors have to cope with particularly heterogeneous environments, both in term of hardware and software. Therefore, they must be able to decode many different protocols. Finally, one should notice the inefficiency of packet sniffing in case of encrypted communications, due to the impossibility for a sensor to decode encrypted information.

Host log files. Host log files allow a fine-grained analysis at the system level of the hosts. For example, Syslog is an audit service provided by UNIX operating systems, which reports information about the application which have been run on the host. It offers possibilities to discover many detailed characteristics of the hosts, which could not be observed in the network packets, since it is able to monitor in detail the activity of each host. However, it consequently presents the drawback of requiring important processing and storage resources, to analyze the large amount of collected data.

Application log files. Application log files also provide relevant and detailed information, in particular concerning servers and services. For instance, web servers log files give information about HTTP requests. Apache can log the identity of the user making the request, the requested URL, the date, the associated return codes, etc. Application log files are known to report information quite easy to process and accurate, in particular because it does not face the same problem of session reconstruction that network packets and host log files methods encounter¹. However, attacks can only be detected when the log file has been written. This means that if an attacker is able to prevent the application from logging information, the attack will not be detected. Moreover, detection of lower-level attacks, such as network attacks, is impossible with application log files.

2.2.1.2 Detection method

Intrusion Detection Systems are generally classified according to two detection approaches: *misuse* detection and *anomaly* detection. The former aims at detecting known attacks, whereas the latter consists in comparing the activity with a predefined “normal” model of activity.

¹Application session reconstruction issues may however be encountered, for instance regarding transactions in database management systems (DBMS).

Misuse detection. It consists in the recognition of symptoms revealing attack attempts in the activity of the information system, using a knowledge base of known attacks or attack scenarios. Thus, misuse detection explicitly detects known unsafe events, the rest being considered as safe. As a consequence, undescribed scenarios can not be detected, and misuse detection suffers from false negatives apparition (absence of alert in presence of illegitimate activity). Defining accurate and sufficiently discriminant scenarios should prevent from false positives (alert in presence of legitimate activity).

Misuse detection is quite easy to implement and deploy, since it generally uses pattern matching with a knowledge base composed of signatures representing known patterns characterizing attack attempts. However, misuse detection suffers from a tedious administration process, requiring expertise and regular updates of the signatures database. On the other hand, thanks to the knowledge of attack scenarios, it offers accurate and detailed diagnoses, offering the possibility to envision response functionalities. The major drawback of misuse detection is that it is not able to detect new attacks, since signatures can be defined only once attacks are known. Thus, misuse detection unfortunately offers a time-window of successful exploitation of new vulnerabilities between the time of their discovery and their actual description as signatures.

Anomaly detection. It consists in the observation of activities which deviates from a normal behavior on the information system. It uses a model of normal behavior, which can be built by observing the system during a learning phase or by defining rules. Note that if the model is acquired through a learning phase, the observation must be realized during a period without any illegitimate actions. Thus, anomaly detection explicitly detects safe events, the rest being considered as unsafe. As a consequence, if the model is not exhaustive and does not exactly reflect what is safe, anomaly detection suffers from false positives. A consistent model which has been acquired without any intrusive activity should prevent from false negatives.

Anomaly detection is not so easy to implement and deploy than misuse detection, due to the need for building and maintaining a reliable and exhaustive model of normal behavior. Maintenance includes updates related to addition of new users and usages, new hosts and services, etc. One of the major drawbacks of anomaly detection is the lack of information concerning detected attacks, due to the absence of diagnosis (it only observes something abnormal). Anomaly detection nevertheless remains quite a generic application of intrusion detection, and offers the advantage of being able to detect new attacks, since it does not try to detect explicit illegitimate actions, but actions which differ from legitimate ones.

2.2.1.3 Alert reporting format

The availability of many commercial and open source intrusion detection systems has lead to the need for a common format for alert reporting. This is first a matter of alert readability and exploitation, to allow inter-operation of multiple products which may be used to monitor a given information system. Thus, a same output format would help further processing of the alerts, especially correlation of distributed intrusions detected

across multiple sites and administrative domains. Such a format would also provide easier communication between a variety of components related to intrusion detection and response. Although a few propositions have been made on this purpose in the past, we only present here the *Intrusion Detection Message Exchange Format* (IDMEF) [31], since it is being considered as a standard in the field of alert reporting.

IDMEF has been developed within the Intrusion Detection Working Group (IDWG) of the Internet Engineering Task Force (IETF). It is supported by many Intrusion Detection Sensors or Systems (natively or using a plugin), such as Snort², Prelude³, and Bro⁴.

IDMEF is based on an XML syntax, providing two main classes, *i.e.* two kinds of messages, *Alert* and *Heartbeat*, as shown in figure 2.2. Analyzers use the *Heartbeat* class to report their current status to managers. In particular, a manager receiving a heartbeat message from an analyzer is aware that the analyzer is up and running, whereas a lack of a significant sequence of heartbeats indicates a failed connection between the analyzer and the manager. Analyzers use the *Alert* class to report alerts resulting from the processing of events observed by sensors. Nine subclasses of *Alert* are provided by IDMEF to express information related to alerts: *Analyzer*, *Createtime*, *DetectTime*, *AnalyzerTime*, *Source*, *Target*, *Classification*, *Assessment* and *AdditionalData*.

- **Analyzer.** Information about the analyzer from which the alert is coming. In particular, it must provide a unique identifier for each analyzer in a given environment.
- **Createtime.** The time the alert was created,
- **DetectTime.** The time the event(s) producing the alert was detected by the analyzer.
- **AnalyzerTime.** The current date and time of the analyzer.
- **Source.** The source(s) of the event(s) leading up to the alert. The *Source* class may contain four subclasses allowing the definition of the source(s): *Node*, *User*, *Process* and *Service*, to describe the host (or device), the user, the process or the network service which appears to be causing the event(s).
- **Target.** The target(s) of the event(s) leading up to the alert. The *Target* class provides the same subclasses as the *Source* class, in order to describe the target to which the event(s) are directed. However, it adds the *File* subclass, to name the potential file(s) involved in the event(s).
- **Classification.** The name of the alert or any other way to identify what it refers to. In particular, a reference from a vulnerability database (such as CVE⁵, OSVDB⁶ or Bugtraq⁷) is frequently provided when possible,

²<http://www.snort.org>

³<http://prelude-ids.org>

⁴<http://bro-ids.org>

⁵Common Vulnerabilities and Exposures, <http://cve.mitre.org>

⁶Open Source Vulnerability Database, <http://www.osvdb.org>

⁷<http://www.securityfocus.org>

- **Assessment.** Information about the impact of the event, the confidence of the alert, and potentially actions to take in response. The *Impact* subclass gives information about the severity of the alert, the type of the alert (such as denial-of-service, reconnaissance probe, etc.), and the completion of the attempt (failed or succeeded).
- **AdditionalData.** Means to provide additional information which are not represented by the data model.

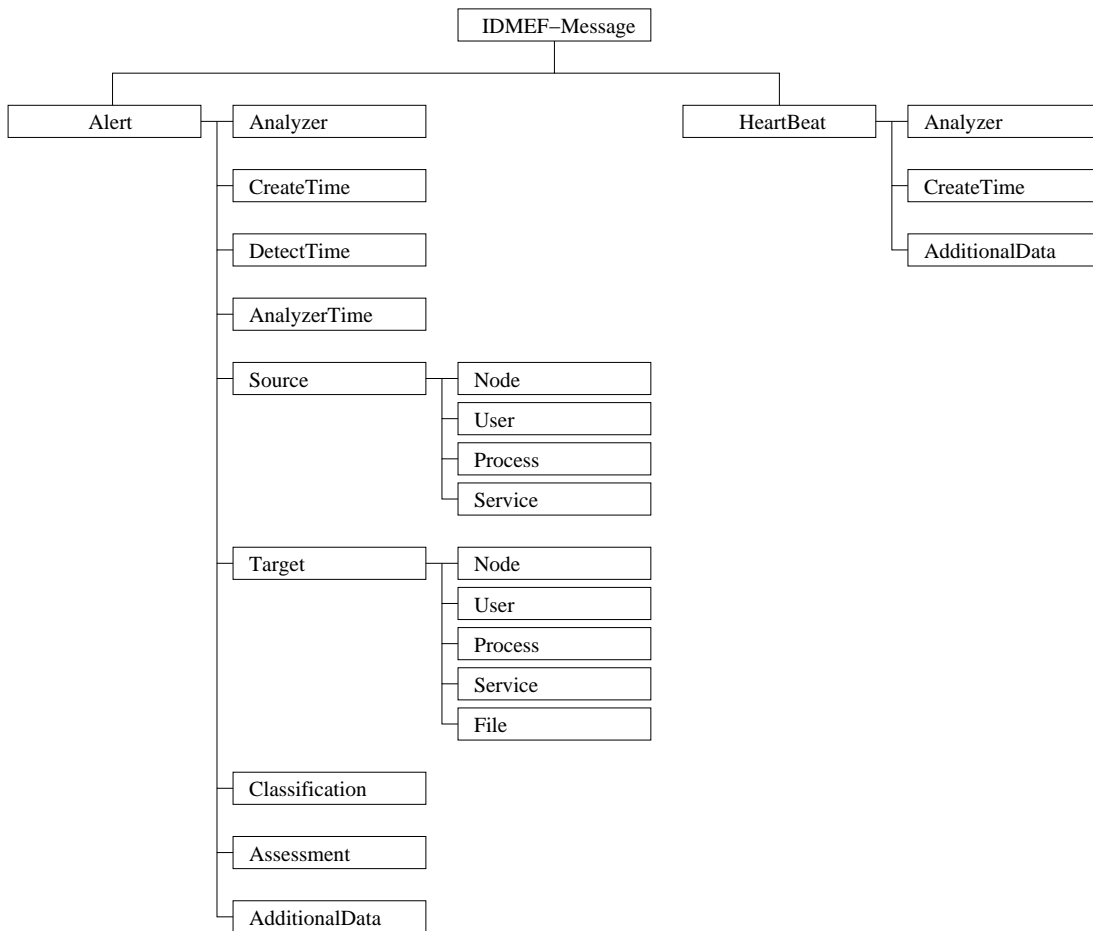


Figure 2.2: IDMEF data model overview

IDMEF *Alert* class also provides three additional possible subclasses, related to alerts:

- **ToolAlert.** Carries additional information about attack tools or malevolent programs, such as Trojan horses, when available. In particular, it includes the name of the program and possibly the complete command.

- **OverflowAlert.** Carries additional information available in case of buffer overflow attacks. It reports information about the program which was attempted to run (not the attacked program), the size of the overflow, and possibly some of the overflow data (buffer or part of it).
- **CorrelationAlert.** Carries additional information about alerts which have been grouped to produce the considered alert because they were related (correlated).

2.2.2 Diagnosis enhancement

As mentioned in 2.2.1.2, developing reliable intrusion detection systems is hard, and intrusion detection in practice is error-prone. As a consequence of the existence of false positives and false negatives, a lot of work has been done in the field of intrusion detection diagnosis enhancement, especially through alert correlation, vulnerability assessment and IDS combination.

2.2.2.1 Alert Correlation

Intrusion detection tools generally produce a too high number of alerts, containing information which are often not accurate and very low-level [32]. Alert correlation is an encouraging way of research to provide better alerts, reducing their volume and improving their semantics. Correlation approaches are classified among three categories: *explicit*, *semi-explicit* and *implicit* correlation.

Explicit correlation. Explicit correlation consists in the *a priori* definition of intrusion scenarios and the matching of low-level alerts with these scenarios. In [28], Cuppens and Ortalo propose a declarative language to specify attack scenarios named Lambda. In [61], Michel and Mé propose a procedural language named Adèle, used to describe signatures allowing the analysis of heterogeneous events. Morin and Debar propose in [65] to make use of the recognition of chronicles [38] to identify alert sequences.

Semi-explicit correlation. Semi-explicit correlation aims at generalizing the explicit approach. It is based on the recognition of the attacker's intention, considering attack *pre-conditions* and *post-conditions*. In [26, 25], Cuppens uses Lambda to link current attack post-conditions with potential attack pre-conditions. This correlates occurrences of the analysis ofd characterizes possible intrusion scenarios. Ning & al exhibit in [67] a similar approach considering *prerequisites* and *consequences* of attacks to construct hyper-alert correlation graphs.

Implicit correlation. Implicit correlation is the most widespread approach. Implicit correlation approaches deal with *data mining*, extracting implicit knowledge (like trends) from alert flow. In [50], Julisch analyzes *root causes* (the reason for which alerts are observed) to handle large groups of redundant alerts. For this purpose, Julisch states that *a small number of root causes generally accounts for over 90% of*

all alerts and that most root causes are linked with configuration issues. Debar and Wespi present in [36] the concept of *situations* to aggregate similar alerts, based on combinations of alerts attributes (source, target and alert class). Valdes *et al* bring up in [81] a probabilistic approach to correlate alerts considering similarity levels, relying on source and target IP addresses and attack identifiers, *i.e.* categories. In [29], Dain *et al.* present an algorithm for fusing the alerts produced by multiple heterogeneous intrusion detection systems, combining the alerts into scenarios. Their approach takes into account attack type, source IP address and alert date. In [8], Autrel *et al.* define functions to compare the set of attributes of alerts in order to compute a similarity operator between two considered alerts. Beyond traditional techniques which make use of alerts attributes, further analysis can be processed on the alert flow to enhance the intrusion detection diagnostic. In particular, Manganaris *et al.* [58] use *data mining* to construct a model of normal behavior of the alert flow, which provides means to detect anomalies in the alert flow considering the context in which they appear. In [83], Viinikka *et al* state that a large amount of the alerts can be seen as background noise. They explain how they aim at monitoring background noise in order to highlight the interesting phenomena in the alert flows by modeling and filtering regularities in alert flows. They address this through the use of classical time series methods.

2.2.2.2 Vulnerability assessment

Vulnerability assessment is of major interest to provide better alerts. First, as explained in [75], correlating information available from passive network observation is used for instance to know if a host is actually vulnerable to a certain attack, and thus to elevate the severity of the alert. An alert may report an attack which does not affect the offended host. For example, an attack specifically based on Microsoft IIS webserver will not affect a host running only an Apache webserver. In such a case, alert severity may be decreased (the alert may as well be deleted or reported as “informative” only), allowing thus a better characterization of what is actually happening. Finally, vulnerability assessment allows false positives recognition, considering hosts having activities which could be misinterpreted as attacks by intrusion detection systems. For instance, the normal behavior of a web proxy is to receive and emit a high number of web requests in a short amount of time, which may lead IDSes not only to consider that web proxies are victims of flooding attacks, but also that they are IP-sweep attackers. The knowledge of the characteristics of the hosts composing the monitored network, in particular the software being run at the time of the attack, is thus essential to enhance intrusion detection diagnostics. In [60], Massicotte *et al.* present a survey on context-based intrusion detection, aiming at connecting IDS (Intrusion Detection Systems) with VDS (Vulnerability Detection Systems). They make use of a Passive Network Monitoring Tool (PNMT) and vulnerability databases (*e.g.* Bugtraq) to define contextual intrusion rules, thus allowing false positive reduction.

2.2.2.3 IDS combination

Finally, intrusion detection diagnostic enhancement is also achieved by combining anomaly and misuse detection. This combination is used both in IDES (Intrusion De-

tection Expert System) [37] and NIDES (Next-generation Intrusion Detection Expert System) [5], implementing anomaly detection, as well as expert systems that encode known intrusion scenarios. In particular, in [5], Anderson *et al.* state that intrusions are well detected from individual users behavior with anomaly detection, whereas known intrusion scenarios are best detected through the use of an expert system rule-base. Barbará *et al.* propose as well ADAM (Audit Data Analysis and Mining) [11], an intrusion detection system based on anomaly detection, but implementing a module classifying the suspicious events into false alarms or reals attacks. In [76], Tombini *et al.* propose a serial combination of *anomaly* and *misuse* detection to better qualify the detection results and decrease the number of false alerts and unqualified events. They notice that anomaly detection only allows the detection of a safe behavior, the contrary only meaning an unknown behavior, but not necessarily an attack. They note as well that misuse detection is only able to detect known attacks (or attack scenarios), but that it does not mean that other events are safe. In their approach, anomaly detection allows first to qualify a set of events which can be considered as safe. Other events are then further analyzed using misuse detection to provide a set of intrusive events, the rest being a set of unknown events dealing with the capacity of the anomaly detection module (potential false positives) and the misuse detection module (potential false negatives).

2.3 Intrusion and Threat Response

2.3.1 Problem statement

Although lots of works have been focusing for years on intrusion detection, few works exist in the field of response. Brackney explains [13] that detection is no more than a first step in the response process. However, he argues that taking actions once the detection has occurred is not a trivial task since one needs to determine *the intent of the intruder, the extent of the damage and the impact to the system*. This means that even if the security operator is quickly informed of an attack, it is still difficult to react quickly and efficiently considering the analysis process he has to achieve on his own. Consequently, Brackney insists on the need for *automatic* and *autonomous* response systems. We shall come back later on response systems characteristics. However, before going further, we should have a look at response vocabulary. The fact is that some notions have to be clearly defined for the purpose of this work.

2.3.1.1 Threat

The first notion to clarify is the notion of *threat*. In the literature, it is difficult to find a unified definition of threat. We first refer to the vocabulary defined in the field of dependability [10], especially the notions of fault, error and failure. A *failure* is an occurrence appearing when the delivered service deviates from the behavior it has been designed for (correct service). A failure is caused by an *error* (*e.g.* an intrusion), which is that part of the system state resulting from a *fault*. A *fault* may either be malicious, *e.g.* attacks, or non-malicious, *e.g.* linked with undesired side-effects

of normal use. Dependability is a system property which deals with *means* (fault prevention, fault tolerance, fault removal and fault forecasting) to preserve *attributes* (mainly availability, reliability, safety, confidentiality, integrity and maintainability) from *threats*. In the MAFTIA project [1] (Malicious-and Accidental Fault Tolerance for Internet Applications), which aim was to investigate tolerance to both accidental faults and malicious attacks, a *threat* is assimilated to any fault, error or failure. In [49], Erland Jonsson defines the *threat* as the origin of *faults* (especially malicious faults), giving examples of threats going from human beings to natural phenomena, or even other computer systems, etc. Jonsson considers that *threat* are those entities which launch attacks towards the system.

In [82], Veríssimo *et al.* present the AVI (Attack-Vulnerability-Intrusion) composite fault model, considering vulnerabilities, attacks and intrusions as faults. We remind that a *vulnerability* is exploited through an *attack* within an *attack scenario*, which leads in case of success to an *intrusion*.

A more generic definition of *threat* is given by Anderson [6], as “the potential possibility of a deliberate unauthorized attempt to: a) access information, b) manipulate information, c) render a system unreliable or unusable”. Thus, a threat is a potential possibility of a deliberate attempt to violate confidentiality, integrity and availability, and thus, a threat is a potential possibility to violate the security policy. A similar definition is also recommended as an “Internet definition” by RFC 2828 [72], a threat being “a potential for violation of security, which exists when there is a circumstance, capability, action, or event that could breach security and cause harm.”

Regarding these multiple visions of the notion of threat, we choose to consider a generic definition, a *threat* being assimilated in this thesis as *any manifestation which represents a possibility of violation of the security policy*. We thus consider that threats include faults, errors and failures, as well as vulnerabilities, attacks and intrusions, as an extension of the notion of fault. As a consequence of security policy violation, threat is seen as the cause of harm to *assets*, *i.e.* resources to be protected.

2.3.1.2 Response

In the literature, one can read at least three denominations of the notion of response: *attack response*, *intrusion response* and *threat response*. In this dissertation, according to our definition of threat, we focus on threat response, and talk indifferently of *response* only. We mean by *response* the mechanism triggered to cope with threats. This means that response is envisioned to any manifestation which represents a possibility of violation of the security policy. Talking about threat as something potential does not mean that response necessarily intervenes before any effective security policy violation, since threat is often revealed by intrusion detection systems through observed attacks or intrusions. However, responding to an intrusion does not mean that other potential violations of the security policy are not considered. Given a particular intrusion, one may decide to not only respond to the intrusion given its victim, but also taking into account other potential victims. We then consider the notion of *reaction* as any action, or set of actions, taken in response to any manifestation of threat. Reaction includes all kinds of means allowing to cope with threat, that is notification as well as

actions changing the state of the system. *Countermeasures* are seen as a restriction of the notion of reaction, that is the part which changes the state of the system [25], *i.e.* simple notifications are not considered as countermeasures.

2.3.2 Intrusion response taxonomies

Although few works have been realized in the field of intrusion and threat response, a few studies have been made in order to provide a taxonomy of intrusion response, which are presented thereafter.

2.3.2.1 Fisch DC&A taxonomy

Fisch DC&A (Damage Control and Assessment) taxonomy [42] mainly focuses on the time of attack detection and the response goal, as shown by figure 2.3. First, it distinguishes whether the intrusion has been detected *during* or *after the attack*. Secondly, it provides a way to classify response goals, among *active damage control*, *passive damage control*, *damage assessment*, and *damage recovery*.

Although Fisch DC&A taxonomy is a first step towards the implementation of any intrusion response system, it suffers from a lack of necessary information to efficiently respond to threat, especially related to the attack and the attacker, but also dealing with the information being attacked.

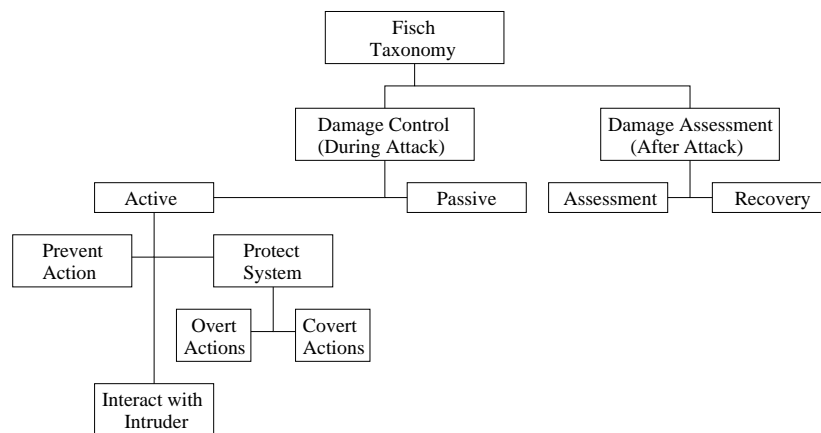


Figure 2.3: Fisch Intrusion Response Taxonomy

2.3.2.2 Carver taxonomy

Considering the limitations of Fisch DC&A taxonomy, Carver and Pooch [19] propose a taxonomy which takes into account 6 dimensions (see figure 2.4). The first dimension is *timing*, similarly to Fisch, but adding the possibility of preemptive responses (prior to an attack). The second dimension is *type of attack*, since a denial-of-service attack may need a different response than a buffer overflow attack, for instance. The third dimension is *type of attacker*, allowing to distinguish an attack issued by a script kiddie

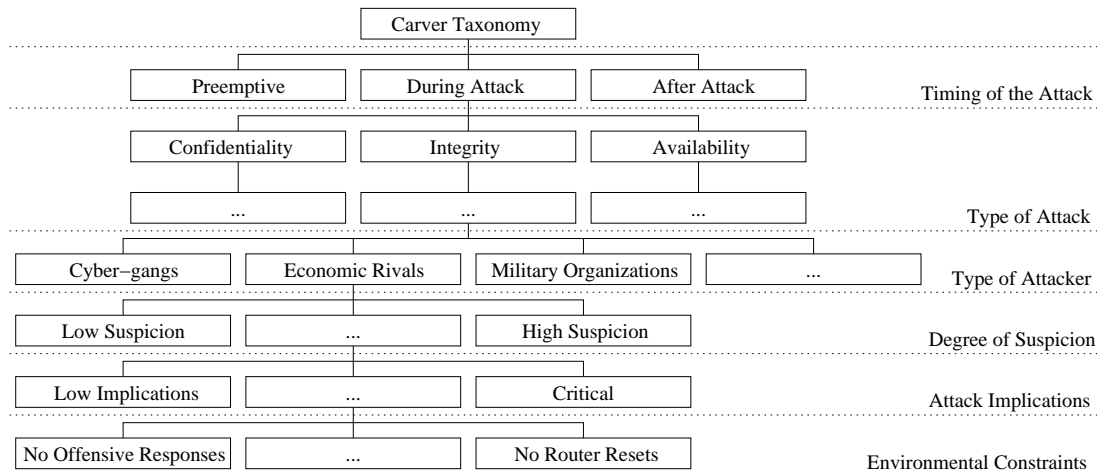


Figure 2.4: Carver Intrusion Response Taxonomy

from an attack issued by a military organization, for example. The fourth dimension is *attack implications*, permitting to consider the difference between an attack towards a sensitive host, such as a DNS (Domain Name Server), and an attack towards a simple workstation. The fifth dimension is the *strength of suspicion*, since intrusion detection is not an exact science. It would thus allow the response to be tempered thanks to the strength of suspicion that an actual intrusion is occurring. The sixth dimension is *environmental constraints*, dealing with legal, ethical, institutional and resource constraints, which may lead to the irrelevance of certain potential responses in the absence of such constraints.

Carver taxonomy does not actually classifies response, since it is attack-oriented. It mainly aims at classifying available information about attacks, which enters in the process of response selection.

2.3.2.3 Stakhanova taxonomy

In [73], Stakhanova *et al.* present a taxonomy which is system-oriented, distinguishing intrusion response systems *by degree of automation* and *by activity of triggered response*. Then, concerning automatic response systems, they are classified *by ability to adjust*, *by time of response*, *by cooperation ability*, and *by response selection method*. Since it is system-oriented, it neither considers any notion of type of attack or attack, nor any notion of impact of the attack. We present here this taxonomy, giving a few references of existing related work.

1. Activity of triggered response
 - (a) **Passive.** Passive response systems only aim at providing notification about the detected attack. Thus, they do not attempt to minimize damage, but rely on the security operator action, considering the reported alerts. For

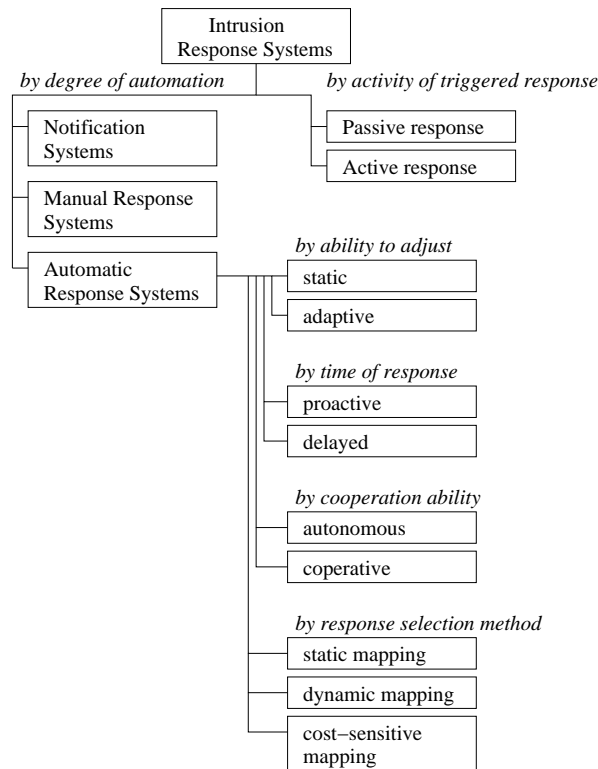


Figure 2.5: Stakhanova et al. Intrusion Response Systems Taxonomy

instance, intrusion detection systems may be considered as passive response systems, since their primary function is to report alerts to the operator when they detect attacks or intrusions.

- (b) **Active.** Active response systems go further than simple notification. Indeed, they provide means to effectively react considering the damage already done and/or the current threat. They may in addition try to locate the attacker, and even try to harm him. Active response systems provide actions such as port or IP address blocking, dynamic access control (for instance, using firewall rules), process terminations, etc.

2. Degree of automation

- (a) **Notification.** Most of the intrusion detection systems only provide notification functionalities. Thus, periodic reports can be considered as a first form of response to threat. However, it requires the intervention of the operator to effectively react. Thus, notification may be qualified of *passive response*. Moreover, one has to note that notification has to be as reactive as possible, since a delay of reporting provides a significant window of opportunity that an attacker could exploit.

- (b) **Manual Response.** It is a first step towards automation of threat response, since it allows the operator to trigger a countermeasure chosen from a set of predefined ones. A great improvement compared to notification is that such systems may be able to suggest potentially adequate responses thanks to functionalities allowing analysis concerning the attack. The operator finally has to choose the most appropriate one among the set of propositions. Such an approach helps gaining reactivity since the operator is assisted in the selection process, but the window of opportunity between the detection and the reaction still relies on the time taken by the human operator to react. Actions may be of various kinds, such as reconfiguring a service, applying a patch, deactivating a user account, redirecting traffic towards a cleaning center, etc.
- (c) **Automatic Response.** It allows responses to be selected and applied automatically. This means that not only is the system able to analyze what kind of response is needed, but also that it should be able to select the most efficient and adequate one among the set of potential ones. Automatic response implies reactivity and efficiency if alerts are reliable and if the process of selection is accurate enough to activate an adequate countermeasure. However, one has to note that if these two conditions are not fulfilled, automatic response could have dramatic side-effects on the monitored environment. Although a few response functionalities are sometimes included in existing intrusion detection systems, they mainly consist in transient and brutal responses, such as connection terminations, and actions are often selected considering too restrictive approaches, without taking into account contextual data ensuring the most adequate responses.

Automatic response systems can be classified through four parameters: ability to adjust, time of response, cooperation ability, and response selection method, as shown in the following:

i. **By ability to adjust**

- **Static.** Most intrusion response systems repeat the same static response for a particular attack. It is easy to implement and maintain, but it does not take into account requirements such as performance or convenience, whereas the environment may change during an attack, which may thus better balance the compromise between security, performance, convenience, etc.
- **Adaptive.** Adaptive systems are able to manage a dynamic compromise between the different requirements which should be fulfilled. Adaptive systems consider parameters characterizing the changing environment during the attack. For instance, adaptability may include activation of additional IDS and consideration of success or failure of previous triggered responses. Thus, it provides a better solution to respond to threat while considering additional issues, but it is more difficult to implement.

ii. **By time of response**

- **Proactive.** Proactive (or preemptive) response systems consist in characterizing threats before any actual intrusion has occurred. For this reason, the proactive approach is the most compliant with the “threat” definition from Anderson. This approach is the most difficult to implement, since it often relies on probability measures and analysis of user of system behavior. Consequently, this approach cannot guarantee 100% correctness. In particular, it may present negative side-effects, since it does not consist in full scenarios recognition, but only parts of them, which let think that there is a threat, and thus that there may soon be a violation of the security policy.
- **Delayed.** Delayed response systems do not respond before getting assurance that the attack is confirmed. It is the most widely used approach, in particular relying on intrusion detection systems diagnoses. However, although it provides better correctness of the response, it also requires more time, which may lead to the compromise of sensitive services or data.

To sum up, proactive systems are more likely to provide better reactivity regarding response, but the probability to make a mistake in the response choice is higher than with delayed systems. As a corollary, delayed systems are most likely to provide a coherent response, but lack reactivity compared to proactive systems. Cuppens & al. [25] propose to use the Lambda language [28] to implement either proactive or delayed response. For this purpose, they build upon the correlation technique presented in [26] to recognize attack scenarios, and introduce the *anti-correlation* technique, allowing them to determine the countermeasures acting on the objective of an intrusion or on a given step of the scenario. With this approach, if the considered step of the scenario has already happened, it provides delayed response, whereas if the step is predicted, it provides proactive response.

iii. By cooperation ability

- **Autonomous.** Autonomous response systems provide local response relying on local intrusion diagnostic, such as a host-based IDS providing response functionalities (*e.g. terminating a process*).
- **Cooperative.** Cooperative response systems may consider information about threat coming from different sources, and provide responses consisting in actions at different levels, on different entities. Cooperative response systems provide adequate and efficient responses, but they are also more complex to develop and deploy.

iv. By response selection method

- **Static mapping.** Defining static mappings is the most obvious way to implement intrusion response. In particular, *decision tables* consist in statically defining mappings between intrusions and corresponding actions. This approach is the most frequently used among existing intrusion response systems. Although simple to im-

plement, it is clearly the worst method to provide efficient response to threat, as it does not consider potential negative effects of the selected response. Moreover, it cannot be deployed over large networks, since analysis of all threat scenarios and definition of the table rapidly becomes impossible for the operator [79]. In addition, the need for considering services dependencies increases the difficulty to define decision tables in large networks. Static mappings also implies that the system is unable to manage incidents which have no corresponding action [70]. Moreover, the approach is particularly sensitive to false positives, since responses are triggered without any verification of the validity of the alerts or evaluation of the potential negative side-effects. Finally, systems implementing static mappings are easily predictable, and are thus very likely to suffer from denial-of-service attacks.

- **Dynamic mapping.** Contrary to static mappings, dynamic mappings do not associate a specific action in response to each possible attack, but a set of potential responses, among which the best one can be chosen thanks to the consideration of metrics relative to the considered attack, for example confidence and severity of the alerts. This approach is implemented through *expert systems*, like Cooperating Security Managers (CSM) [87] and Event Monitoring Enabling Responses to Anomalous Live Disturbances (EMERALD) [69]. Dynamic mappings may rely on adaptive systems, which allow fine-grained responses, thanks to their awareness of their inherent changes. For instance, Adaptive, Agent-based Intrusion Response System (AAIRS) considers false positive rate of the IDS in the decision process, in order to limit uncertainty in intrusion response [18]. Considering false positive rate provides means to adapt the severity of the triggered responses. For example, given an IDS with an expected low false positive rate, the system is likely to react with severe actions. Moreover, AAIRS is able to evaluate the success of a response and to adapt a response plan based on its success or failure.
- **Cost-sensitive mapping.** Last, but not least, cost-sensitive models are considered as the best approach to respond to threat. They aim at balancing intrusion damage and response cost in order to choose the best response. Several cost and risk factors are considered. Toth and Kruegel [79] propose an approach allowing to choose the response with the least impact. A model of the network is used by an *impact evaluation function* to consider topology and entity dependencies in the response selection process. Toth and Kruegel define the notion of *response configuration* as the set of response actions that have already been applied. They express the *capability* of an entity as a value describing how far a resource can perform its work given the current response configuration, compared to the sit-

uation where all needed resources are available. They also define the *penalty cost* as a value representing the cost when an entity becomes unavailable. Cost optimization is then possible thanks to the minimization of the penalty cost of new response actions. However, they point out the fact that finding the response with the least impact does not ensure that the overall response configuration is globally optimal. This means that one should also take into account minimization of the overall penalty cost. Lee & al also explain in [57] the need for cost-sensitive modeling in the field of intrusion detection and response. In particular, they discuss the need for consideration of the cost of damage of intrusions, the cost of manual and automated response to an intrusion, and the operational cost, measuring constraints on time and computing resources. For instance, we can compare the damage cost with the response cost in the decision process, so that an automated response could not lead to a situation which would be even worse than the current situation. Considering cost can be achieved using metrics such as *criticality* and *lethality*, as defined by Northcutt [68]. The former defines how critical the target is, and the latter how likely the attack is to damage. Northcutt uses a five-point scale for each metric, providing for instance a higher cost for a core router than a user UNIX desktop system, or a higher cost for a root access than a user access. Lee & al. use *criticality* and *lethality* as defined by Northcutt, but they add the notion of *progress* to compute the actual cost, assuming that a worst-case scenario may be envisioned with $progress = 1.0$.

2.3.2.4 Limitations of current taxonomies

Regarding the three presented taxonomies, there is a lack of actual response taxonomy. Fisch and Stakhanova taxonomies are system-oriented, whereas Carver taxonomy deals with response, but from the attack-side. To our knowledge, there does not exist any taxonomy of response itself, including response strategy, timing of response, and other properties related to actual relevance and efficiency of chosen response. It would also be interesting to distinguish short-term response, *i.e.* countermeasures which only deal temporarily with threat, like server port filtering, from long-term response, which actually fixes something durably, like application security patches.

2.3.3 Implementing threat response

There are two main ways to implement threat response: (1) Intrusion Prevention Systems (IPS) and (2) Intrusion Detection and Response Systems (IDRS). We present here these two approaches, focusing on the second one, since the subject of this thesis is dedicated to response triggered thanks to intrusion detection diagnosis. We then give a short description of the possible means which can be used to achieve threat response.

2.3.3.1 IPS vs. IDRS

IDRS: off-line approach. As explained previously, Intrusion Detection Systems work off-line, that is they analyze *a posteriori* events which occur in the information system. Although IDSs aim at detecting attacks as soon as possible, *a posteriori* detection means that the detected events have actually occurred. However, this does not necessarily mean that IDSs are only able to detect completed intrusions. Indeed, as explained in 2.2.2.1, semi-explicit correlation also allows the recognition of attacker's intention, based on the consideration of current attack post-conditions with potential attack pre-conditions. IDRSs build upon diagnosis of IDSs to provide response. IDRSs may be either based on *network-based* intrusion detection or *host-based* intrusion detection.

IPS: on-line approach. Intrusion Prevention Systems (IPSs) work on-line so that events which are considered to be illegitimate are prevented on the fly from completing. *Network-based* intrusion prevention means that sensors are crossed by network traffic, and *host-based* intrusion prevention means that a process is in charge of analyzing any event before allowing its completion. This approach does not deal with the two main limitations of intrusion detection. The false positives issue exists for IPSs as well as for IDSs. Response to false positives may lead to particularly negative side-effects, such as self-inflicted denial-of-service. Moreover, IPSs face as well as IDSs the issue of high volume of activity analysis. Since IPSs work on-line, they introduce a new issue dealing with potential bottlenecks, for network-based IPSs as well as host-based IPS. In fact, if an IDS is not able to process every packet on the network, it is dropping a part of the traffic without congesting the network. On the opposite, since IPSs are crossed by network traffic or by host-based activity, they have to be able to analyze it on the fly to avoid congestion.

2.3.3.2 Means to respond

Studies [47, 22] reveal that response may fall within four main categories: (1) notification, (2) dissuasion, (3) counterattack and (4) reconfiguration.

1. **Notification.** Notifying an attack or an intrusion (and possibly a threat) to the security operator is a first kind of response. Moreover, notifications can be extended to other entities outside the organization, for example the attacker's ISP (Internet Service Provider). In such a case, trust relationships between ISPs are required, since actions taken for their own clients are possibly triggered by their concurrents, which may be difficult to admit.
2. **Dissuasion.** It aims at intimidating the attacker in order to have him stop his intrusion attempt. It can be explicit, by notifying him that his malicious activity is being monitored. However, this method is likely to motivate certain attackers even more. It can also be implicit, for example by sending ICMP messages such as "Destination Unreachable". However, this is rendered difficult because most attack tools use their own TCP/IP stack, which is not necessarily implemented

conforming to standards, and most firewalls do not take ICMP messages into account and delete them.

3. **Counterattack.** A third way to respond to attacks is to deploy counterattacks, that is to say offensive responses. Session sniping is a kind of counterattack, since it aims at forging a RST packet in order to stop the attacker's connection. However, it sometimes lacks reactivity since it first needs the evaluation of the attacker's TCP window size, which is not an instantaneous task [43]. One can also imagine far more complex counterattack scenarios, but legal constraints and possibility of attacking innocent victims render these methods difficult to apply. In fact, once again, such responses can have harmful side-effects on the information system and degrade services availability.
4. **Reconfiguration.** Another way to react is to adapt the information system considering characteristics of the malicious activity. This is realized by reconfiguring equipments like firewalls or routers. A first technique is *shunning*, which consists in denying access to the attacker, typically by reconfiguring a firewall to block his IP address. This method is effective but can lead to self-inflicted denial-of-service, with the attacker forging packets in order to have services blocked for legitimate users. Another technique is redirection of assumed intrusive traffic to another path. It protects sensitive servers, by using mirror ones, or to study attackers activity, with the use of honeypots for example. Redirection is mainly realized by reconfiguring routing tables. Many services can be reconfigured in order to reflect an environment adapted to the security context. Being able to provide different path to access resources, different encryption algorithms or different authentication methods allows a dynamic and intelligent adaptation of the information system considering security contexts.

2.4 Security Policies and Access Control

We give in this section information and related work dealing with security policies and access control. We first explain the role of a security policy, before giving different qualities of security policies and presenting means for policy distribution. We then give three access control models, presenting interest for understanding the thesis proposal.

2.4.1 Security Policies

2.4.1.1 Definition

Security policies manage information systems security, ensuring *confidentiality* and *integrity* of the resources and *availability* of the services and resources. Mechanisms such as authentication, access control and communications encryption are deployed over the information system. A security policy is generally represented by a set of permissions, prohibitions, obligations and recommendations defining the legitimate actions which can be performed on the system. An intrusion is generally defined as a violation of the security policy. One has to note that violating the security policy may not be related

to an actual attack, but sometimes only linked with a misbehavior of a legitimate user. A security policy may vary from “a set of managerial platitudes” [7] to a set of rigorous and formal rules (RBAC [71], Or-BAC [51]), as explained in MAFTIA [1]. It may also present different levels of abstraction, and more or less fine-grained requirements, from security requirements dealing with normal behavior of components to minimal length of keys and passwords.

Security policies are sometimes also associated with *non-repudiation* and *accounting*. Non-repudiation could be considered as a fourth security variable (with confidentiality, integrity and availability), since it aims at ensuring that a transferred message has been sent and received by the parties claiming to have sent and received the message. Non-repudiation deals more with privacy and legal constraints than security itself and non-repudiation is generally ensured using signature. Accounting is more a mechanism to ensure security properties than a property itself, since it allows *a posteriori* analysis of the activity occurring on the information system.

A security policy may be divided into multiple sub-policies, including an *access control policy*, an *authentication policy*, an *encryption policy*, a *signature policy*, etc. Note that it is easy to understand that such policies express means to ensure confidentiality and integrity, but preserving availability is much more difficult. In particular, preserving availability sometimes requires to constrain access mechanisms, requiring for instance strong authentication or encryption. Availability is also strongly linked with network management, especially software updates, to rapidly bridge security breaches caused by software vulnerabilities.

Policy definition vs. instantiation. One should distinguish two aspects of a security policy: the policy *definition* and the policy *instantiation*. The former is related to the set of rules describing the policy specification. The latter refers to the actual implementation of the policy over the information system. Policy definition generally consists in a set of *prima facie* authorizations, which may differ from policy instantiation, namely the set of *actual* authorizations effective at a given time.

Defining and deploying security policies is generally associated with two main entities: *Policy Decision Points* (PDP) and *Policy Enforcement Points* (PEP). PDPs are specific entities deciding which rules of the policy definition are enforced. PEPs are devices of the information system (firewalls, authentication servers, applicative servers, etc.) in charge of current policy instantiation application.

Note that the terms *definition* and *instantiation* of the policy are both associated with the policy language. This means a policy language should be used to describe both definition and instantiation of the policy. However, enforcing the policy over the information system requires an additional phase, translating the current security policy into configurations of PEPs. For example, instantiation of an access control policy may be translated into firewall rules. When dealing with what is actually enforced at the PEP level, we will simply talk about *configurations*.

Static vs. dynamic policy. Security policies fall among two main categories: *static* policies and *dynamic* policies. The major difference between these two kinds of policies is that dynamic policies are able to take into account the context to better adapt to

security (and other) requirements. Brézillon and Kouadri [16] defines the notion of context as *the situation that surrounds both the requested service environment and the user's environment*. They illustrate in figure 2.6 the concept of context-based security, which relies on *context triggers* which activate *security contexts* to which the *security policy* adapts in order to better control a *pervasive system*.

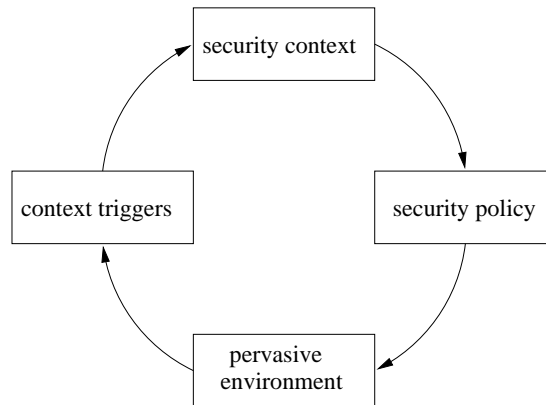


Figure 2.6: Context based security (Brézillon and Kouadri)

More generally, one can consider that the notion of context refers to the state of the system at a given time. In particular, the context is related to the threat level, but also to the cartography of the information system (topology and inventory, *i.e.* configuration of the hosts, allowing vulnerability assessment), or even with time, etc. This means that contrary to static policies, which do not change over time, dynamic policies express requirements which are dedicated to a given context. In fact, this is materialized by rules, consisting in permissions or prohibitions, which are activated considering the context, whereas they are always active in the case of static policies. Dynamic policies thus explicitly distinguish the policy definition from its instantiation. Note that systems built upon dynamic security policies able to take context into account are generally named “adaptive systems”, *i.e.* systems which adapt to the situation of use.

2.4.1.2 Formalism

As already explained, security policies are specified as sets of rules describing permissions and prohibitions (and possibly obligations). A security policy does not define every possibility, *i.e.* every permission and every prohibition. Only the necessary permissions and/or prohibitions are described, the rest being implicit, either “default permit” (*open policy*), or “default deny” (*closed policy*).

More generally, Policy Core Information Model (PCIM) [64, 63] has been proposed as a generic framework for defining policy rules, by the *Policy Framework Working Group* (PFWG) of the IETF. PCIM models a network managed by policies as a state machine. Implementing the policy is linked with defining in which state a given entity of the information system is supposed to be or to be authorized to be at a given time,

or more generally, in a given context. This is realized through the definition of rules linked with certain conditions and actions, allowing a dynamic policy, since these rules are activated by conditions. Note that PCIM is more a structural model than a ready-to-use model. No implementation of PCIM exists, and additional requirements are necessary to implement such a model, especially relatively to information persistency. [74] proposes to use a LDAP directory (Lightweight Directory Access Protocol), in order to address data storage and information manipulation issues.

2.4.1.3 Distribution

Distributing policies deals with configuring the devices responsible for the enforcement of the policy over the information system, namely the policy enforcement points (PEPs). Common Open Policy Service (COPS) [39] is a protocol which considers a client/server model to manage *bottom-up* policy deployments (*a.k.a. outsourcing*). In this approach, clients (PEPs) ask server (PDP) for access to resources or services requested by users. The *top-down* approach (*a.k.a. provisioning*) consists in updating the policy instantiation at the PEP level by pushing configuration updates independently from user requests (*e.g.* COPS-PR [20]). In this approach, the configurations of the PEPs are supposed to effectively represent the current instantiation of the policy, which means that any change in the policy instantiation is immediately updated in the configurations of the PEPs.

Distributing a security policy also requires the definition of an architecture, allowing interconnection of the entities in charge of the policy management, especially the PDPs and the PEPs. The generic Authentication, Authorization and Accounting (AAA) architecture [30, 85, 84, 41] has been proposed by the *Network Working Group* (NWG) of the IETF. AAA has been developed in particular to fulfill the need for authorization of many internet services, allowing multi-domain authorization between multiple service providers.

AAA architecture relies on *generic AAA Servers* (see fig. 2.7), which are entities capable of authenticating users, handling authorization requests and collecting accounting data. A *generic AAA Server* is connected to an *Application Specific Module* (ASM), ensuring both the specific part of authorization decision (what cannot be generic considering a given service) and managing resources and *service equipment* configuration to provide the authorized service. Each AAA Server is also aware of a *policy and event repository* giving the knowledge of available services and resources, and the rules to provide authorizations.

Interactions between multiple AAA Servers and in particular between AAA Servers of different administrative domains enable complex authorizations. A layered AAA protocol provides requirements for a AAA-compliant protocol, including (1) reliable and secure transport, (2) transaction and session management, (3) presentation (generic semantics for data structures) and (4) application (for instance, authorization decision functions, advertisement publication, etc.).

AAA is more a framework than an implementation of an authorization architecture. It mainly provides requirements and examples of applications. Generic use cases are given in RFC 2904, showing different scenarios of authorization, distinguishing autho-

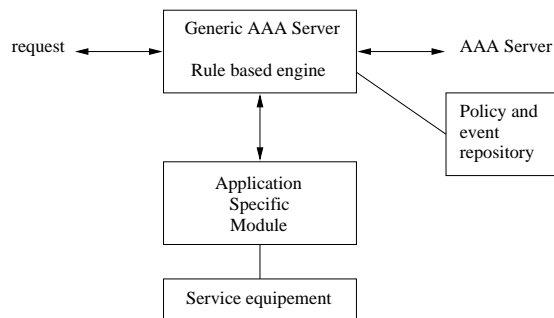


Figure 2.7: Generic AAA Server Interactions

rizations requested to AAA Servers (agent and push sequences) from those which are directly requested to service equipments (pull sequence), and explaining how to handle roaming and distributed services. However, implementations exist, such as Diameter (RFC3588), an authentication protocol developed considering the limitations of RADIUS (Remote Authentication Dial-In User Service), in particular concerning roaming support and number of possible simultaneous access requests.

2.4.1.4 Enforcement

Dedicated mechanisms. Policy enforcement may be realized through the use of many isolated mechanisms. Many vendors of network appliances, like Cisco or Juniper, propose devices able to receive and enforce new configuration scripts. This allows for instance the dynamic redefinition of routing tables or access control lists. SnortSAM⁸ is a plugin interfacing the Snort intrusion detection sensor with many current firewall implementations. SnortSAM is compatible with a few commercial frequently used appliances, like Cisco PIX and Checkpoint Firewall-1, but also with free software filtering solutions, especially Linux IPTables. However, SnortSAM is strongly linked with the use of Snort, and response is triggered directly based on reported alerts at the Snort engine level, so it is not easily natively usable to enforce any firewall policy rule.

Such enforcement capabilities are however not provided by many potential PEPs. In particular, various servers, like web servers and mail servers, do not propose such functionalities natively, and similarly for client stations. Thus, there is often a need for additional agents deployed on such hosts, which receive commands to enforce the policy locally.

Generic mechanisms. Some work exist which aim at providing means for generic policy enforcement. Netconf [40] is a configuration protocol defined to provide mechanisms to install, manipulate and delete the configuration of network devices. It uses an XML-based data encoding, to represent configuration data as well as protocol messages. Netconf provides a framework for policy enforcement, relying on RPC (Remote Procedure Call), ensuring independence towards transport layer (*e.g.* BEEP, SSH, etc.). The

⁸<http://www.snortsam.net>

major contribution of Netconf is to provide an operation layer, above the transport and RPC layers, allowing the use of XML-based generic operations for policy enforcement. Note that actual configuration data are to be defined at a higher level (content layer).

Cfengine⁹ [17] is a freely available distributed agent framework allowing policy-based network management. It thus brings up policy enforcement capabilities, based on a centralized policy-based specification and distributed agent-based action, which means that each host is responsible for its own maintenance. Cfengine permits the configuration of files and processes running on interconnected computers, *e.g.* UNIX or Windows workstations.

Microsoft also provides a solution for policy enforcement, namely Microsoft NAP (Network Access Protection) [21]. It aims at ensuring that computers which require for resources are compliant with respect to the current policy. It does not actually enable policy enforcement in the sense of automatically pushing configurations to hosts, but allows access control considering host capabilities regarding the current policy. With the use of Microsoft NAP, hosts are prevented from accessing resources until they are compliant with the policy requirements.

2.4.2 Access Control Models

We do not intend to give an exhaustive state-of-the-art of access control models. This section aims at providing the necessary elements to understand the approach we develop in this thesis. One can refer to [62] for more information concerning access control models.

2.4.2.1 DAC

Discretionary Access Control (DAC) is a model aiming at controlling access to information through the definition of *privileges* of *subjects* on *objects*. The HRU model defined by Harrison, Ruzzo and Ullman [46] is based on the definition of an *access control matrix* to describe authorizations. This mechanism is used in UNIX operating systems to define the user-group-other permission bits.

Let $S = s_1, s_2, \dots, s_n$ be a finite set of subjects and $O = o_1, o_2, \dots, o_p$ a finite set of objects. Let us consider a UNIX operating system, where subjects are users, and objects are files. Permissions to access these files are divided into three categories: read, write and execute. Table 2.1 give an example of an access control matrix.

	o_1	o_2	...	o_p
s_1	r	rwx	...	r
s_2	rw	r	...	rwx
\vdots	\vdots	\vdots	\ddots	\vdots
s_n	rwx	rw	...	r

Table 2.1: Sample access control matrix defining UNIX permissions

⁹<http://www.cfengine.org>

Although quite simple to implement and massively used as access control lists in many operating systems, the DAC model suffers from many drawbacks. DAC presents scalability issues and rapidly becomes unfeasible in environments where many entities (subjects and objects) and many privileges have to be described. Moreover, expressiveness in the DAC model is very poor, consisting in the exhaustive enumeration of numerous triples (*subject, object, privilege*), which results in the definition of a static policy.

2.4.2.2 RBAC

Role-Based Access Control (RBAC) [71], defined by Sandhu *et al.*, introduces the abstraction of subjects into *roles*, in particular to respond to scalability issues of models such as DAC. RBAC is used to define explicitly *permissions* of *actions* made by *subjects* (through *roles*) on *objects*. Thus, describing an access control policy is no more a matter of listing (*subject, object, privilege*) triples, but facts such as *permission(role, action, object)*. As a consequence, when a *role* is permitted to make an *action* on an *object*, all the subjects assigned to this role inherit this permission.

For example, *permission(physician, read, patient_bill_file)* defines that role *physician* is allowed to make action *read* on object *patient_bill_file*. This means that each subject assigned to the role *physician* is being able to *read patient_bill_file*.

Thanks to roles, RBAC offers better scalability. However, roles only provide a way to group subjects, not actions and objects. Moreover, RBAC only manages permissions, prohibitions being implicit. Finally, it still consists in a static policy, since permissions are statically defined *a priori*.

2.4.2.3 Or-BAC

Organization-Based Access Control (Or-BAC) [51, 62] provides many improvements considering RBAC limitations. First, access control is managed at the *organization* level, allowing not only to abstract *subjects* into *roles*, but also *actions* into *activities* and *objects* into *views*. Secondly, Or-BAC provides not only *permissions*, but also *prohibitions* and *obligations*. Thirdly, Or-BAC allows to trigger access control rules thanks to activated *contexts*, enabling thus to manage a *dynamic* security policy.

A generic policy definition thanks to entity abstraction. The concept of *organization* is central in the Or-BAC model. Intuitively, an organization is any entity that is responsible for managing a security policy. Thus, a company is an organization, but concrete security components such as a firewall may be also viewed as an organization.

The objective of Or-BAC is to specify the security policy at the *organizational* level, that is abstractly from the implementation of this policy. Thus, instead of modeling the policy by using the concrete and implementation-related concepts of subject, action and object, the Or-BAC model suggests reasoning with the roles that subjects, actions or objects play in the organization. The role of a subject is simply called a *role* as in the RBAC model. On the other hand, the role of an action is called an *activity* whereas the role of an object is called a *view*.

Each organization can then define security rules which specify that some roles are permitted or prohibited to carry out some activities on some views. These security rules do not apply statically as their activation may depend on contextual conditions. For this purpose, Or-BAC explicitly introduces the concept of *context*. Using a first order logic formalism, security rules are modeled using a 6-places predicate:

- $security_rule(type, org, role, activity, view, context)$,
where $type$ belongs to $\{permission, prohibition\}$.

For instance, the following security rule:

- $security_rule(prohibition, corp, pop_user, read_pop, mail_server, pop_threat)$.

means that, in organization *corp*, a pop user is forbidden to use the pop service to consult his mail in the context of pop threat.

All these concepts, organization, role, activity, view and context, may be structured hierarchically. Permissions and prohibitions are both inherited through these hierarchies (see [23] for more details).

Since a given security policy may include permissions and prohibitions, conflict management strategies have to be defined to solve the possible conflicts. In Or-BAC, such a strategy consists in assigning a priority to each security rule. Priorities define a partial order on the set of security rules so that when a conflict occurs between two rules, preference is given to the rule with the higher priority. Priority assigned to security rules must be compatible with hierarchies defined on entities such as organization, role, activity, view and context. Thus, in case of conflict, if a given security rule is inherited by a given entity, this rule will have lower priority than another security rule explicitly assigned to this entity.

Once the organizational security policy is defined, it is possible to check if the conflict management strategy is *effective*, that is it will solve every conflict at the concrete level (see [62] for further details). Since the Or-BAC model abides to the Datalog restrictions [80], we can prove that it is possible to decide in polynomial time that a conflict management strategy is effective.

The organizational policy is then used to automatically derive concrete configurations of PEPs. For this purpose, we need to assign to subjects, actions and objects, the roles they play in the organization. In the Or-BAC model, this is modeled using the three following 3-places predicates, as shown in figure 2.8:

- $empower(org, subject, role)$: means that in organization *org*, *subject* is empowered in *role*.
- $consider(org, action, activity)$: means that in organization *org*, *action* is considered an implementation of *activity*.
- $use(org, object, view)$: means that in organization *org*, *object* is used in *view*.

For instance, the fact $empower(corp, alice, pop_user)$ means that organization *corp* empowers Alice in role *pop_user*.

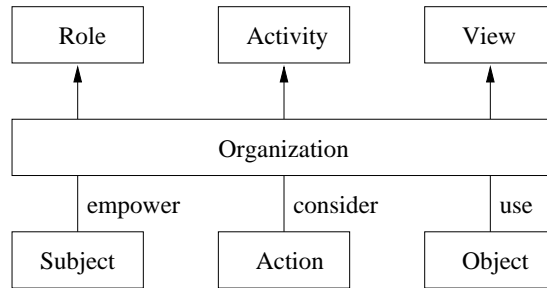


Figure 2.8: Abstraction of access control entities in Or-BAC

Instead of enumerating facts corresponding to instances of predicate *empower*, it is possible to specify role definitions which correspond to logical conditions that, when satisfied, are used to derive that some subjects are automatically empowered in the role associated with the role definition. Activity and view definitions are similarly used to automatically manage assignment of action to activity and object to view. For instance, in a network environment, we can use a role definition to specify that every host in the zone 111.222.1.0/24 is empowered in the role *DMZ*.

We use Prolog notation to specify Or-BAC security policies. The only important prolog constructs to remember are that constant values start with a lowercase character, that variables start with an uppercase character, and that `_` denotes any value.

Derivation of a concrete policy from active contexts. We also have to define logical conditions to characterize when contexts are active. In the Or-BAC model, this is represented by logical rules that derive the following predicate:

- *hold(org, subject, action, object, context)*: means that in organization *org*, *subject* performs *action* on *object* in context *context*.

Using the model, one can then derive concrete authorizations that apply to subject, action and object from organizational security rules. This is modeled by the derivation rule shown in Listing 2.1. In an organization *Org*, the security rule expresses a *permission* for a given *Role* to make a given *Activity* on a given *View* in a given *Context*. The predicates *empower*, *consider* and *use* indicate that *Role*, *Activity* and *View* are respectively abstractions of *Subject*, *Action* and *Object* in the considered organization. When the considered *Context* is being held for *Subject*, *Action* and *Object* through the *hold* predicate, we can thus derive the fact that it is permitted for *Subject* to make *Action* on *Object*.

Listing 2.1: Derivation of concrete authorizations

```

is_permitted(Subject, Action, Object) :-
    security_rule(permission, Org, Role, Activity, View, Context),
    empower(Org, Subject, Role),
    consider(Org, Action, Activity),
    use(Org, Object, View),
    hold(Org, Subject, Action, Object, Context).
  
```

2.5 Conclusion

In this chapter, we presented the related work, providing a basis for our work in three main fields of research, namely intrusion detection, response, and security policies. In the following, we shall now present how we use intrusion detection alerts to dynamically enforce a contextual security policy providing automated response to threat. Before going further on the use of a contextual policy, we propose a generic threat response architecture, according to our objectives, in the next chapter.

Chapter 3

Proposal of a Threat Response System

Contents

3.1	Introduction	33
3.2	Architecture of the Threat Response System	35
3.2.1	Proposed architecture	35
3.2.2	Role of the components	37
3.2.3	Scalability of the architecture	41
3.3	Multiple Paths to Resources	42
3.3.1	Host level	43
3.3.2	Service / Application level	43
3.3.3	Information level	44
3.4	Conclusion	45

3.1 Introduction

Our reflexion is based on the AAA architecture [30, 85], since AAA provide means to build upon a generic server able to deploy policies through the use of local decisional entities in charge of policy enforcement.

We also use Col. John Boyd OODA's strategy [12]. Figure 3.1 shows how Boyd defines an architecture that unifies planning and control for military command and control. This architecture, named Observe-Orient-Decide-Act (OODA), is a basis to develop autonomous systems. These concepts support the definition of an autonomic threat response system.

The four processes highlighted by OODA are defined as follows [45]:

- **Observe** is the process of acquiring information about the environment by interacting with it, sensing it, or receiving messages about it. Observation also

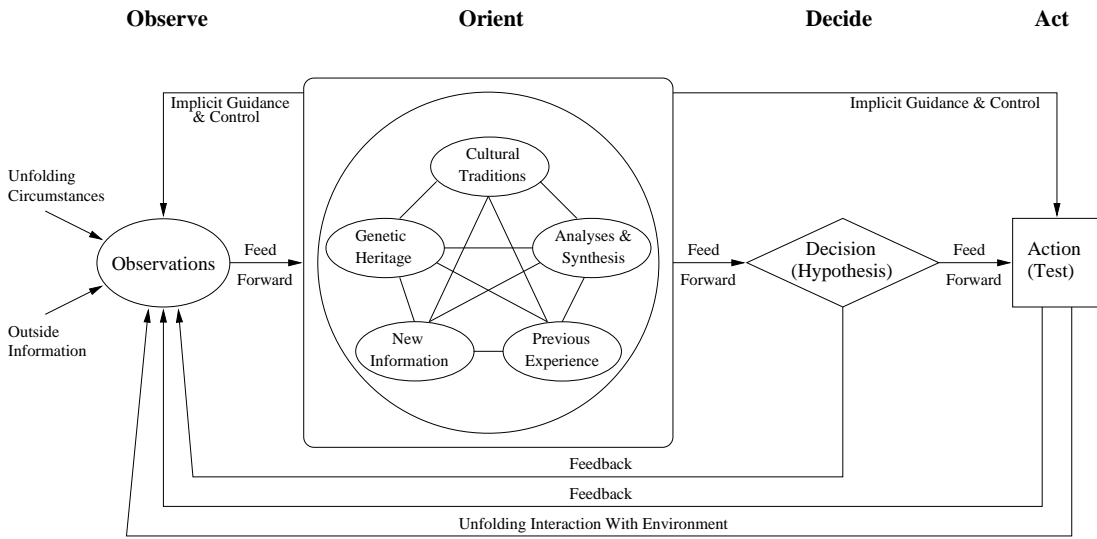


Figure 3.1: John R. Boyd's OODA strategy model

receives internal guidance from the Orient process, as well as feedback from the Decide and Act processes.

- **Orient** is known as situation analysis or situation assessment. Orient is the key process of Boyd's architecture. It reflects the vision of the world, the way to interact with the environment. According to Boyd, "Orient shapes the way we observe, the way we decide, and the way we act".
- **Decide** is the process of making a choice among hypotheses about the environmental situation and possible responses to it. Decide is guided by internal feed-forward from Orient, and provides internal feedback to Observe.
- **Act** is the process of implementing the chosen response by interacting with the environment. Act receives internal guidance from the Orient process, as well as feed-forward from Decide. It provides internal feedback to Observe.

We first describe our architecture, respectively mapping our requirements on the AAA components and on the OODA strategy. We explain the role of the different components and discuss the scalability of the architecture. In a second part, we show that it is possible to manage multiple paths to resources in order to ensure the three security properties, especially availability.

3.2 Architecture of the Threat Response System

3.2.1 Proposed architecture

3.2.1.1 Mapping onto AAA architecture

We support the AAA architecture for the following reasons:

- A generic component (generic AAA server) is responsible for managing generic policy rules, which reflects a policy defined abstractly from its implementation,
- Application Specific Modules (ASMs), acting as Policy Decision Points (PDPs), ensure local decisions and processing according to local contextual data,
- Service equipments act as Policy Enforcement Points (PEPs), allowing to actually implement the policy instantiation.

However, we do not intend to make full use of the Authentication, Authorization, Accounting framework. First, we restrict to an architecture with a single generic AAA-like server, which we call the Policy Instantiation Engine (PIE). Secondly, we consider that propagated policy rules are authorized *de facto* and automatically processed by PDPs, without any notion of authorization in the sense of AAA. Moreover, since we rely on intrusion detection to provide alerts as an input for our system, we need sensors, used to collect events allowing to report alerts. Analysis of events (or low-level alerts) should be realized by an Alert Correlation Engine (ACE), in order to bring up reliable and high-level meta-alerts, characterizing actual attacks or intrusions.

To sum up, we need *sensors* to feed an *Alert Correlation Engine* (ACE). The ACE sends alerts to a *Policy Instantiation Engine* (PIE), which provides global policy rules evaluation and instantiation. Policy instances are processed by a *Policy Decision Point*, deciding what should be done locally for each policy instance. The PDP is responsible for pushing configurations to *Policy Enforcement Points* (provisioning approach), which reflect the actual implementation of the policy over the information system. Consequently, we propose to develop an architecture based on figure 3.2.

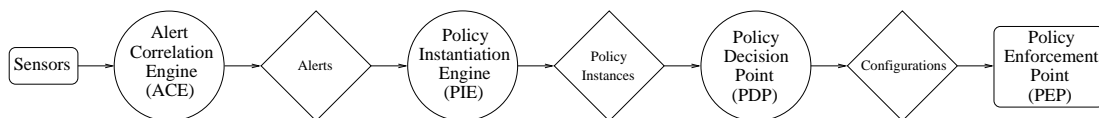


Figure 3.2: Mapping our requirements onto AAA

3.2.1.2 Mapping onto OODA architecture

We classify our requirements among the four OODA processes as follows:

- **Observe:** events are collected by *sensors* and received as messages about the environment by the **ACE**.

- **Orient:** situation analysis is partly achieved by the **ACE**, responsible for the alert-side vision of the world. Threat assessment and response strategy (countermeasures) is achieved by the **PIE**.
- **Decide:** choosing among hypotheses about the environmental situation to provide response is done by the **PIE** through policy compilation, that is choice of policy rules and instantiation with respect to current context.
- **Act:** the PDP implements the chosen response finding the policy enforcement points, translating policy instances and pushing new configurations. The **PEPs** are in charge of actually running new configurations characterizing the current policy implementation.

Figure 3.3 maps our architecture on the OODA strategy. Note that the **PDP** may present local decisional capabilities, we thus include it in the Decide process. However, decisions it is likely to make mainly deals with how policy instances are locally enforced, thus it should not fundamentally change vision of the world from the PIE's point of view, and the most important decisional capability actually remains to the responsibility of the PIE.

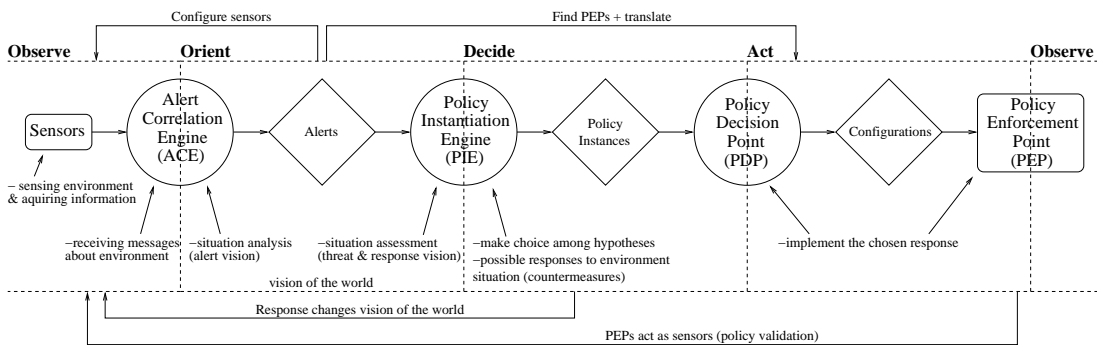


Figure 3.3: Mapping our requirements onto OODA

Feedback and guidance arrows defined in OODA appear as follows in our architecture:

- **Decide-Observe feedback:** It may be assimilated to the fact that Decide changes the vision of the world, that is a new policy implementation has implications on the observed environment. This is implicit in our scheme.
- **Act-Observe feedback:** PEPs act as sensors and provide information about the environment. This is useful to feed the ACE (*e.g.* firewall logs), and possibly to validate the correct implementation of the policy instantiation.
- **Orient-Observe guidance & control:** Vision of the world at the Orient process is used to dynamically configure sensors, according to situation assessment. For instance, this may correspond to dynamic signature activation of an intrusion detection sensor (*e.g.* Snort). We do not provide this functionality.

- **Orient-Act guidance & control:** Information about the environment stored at the Orient process level provides input to find policy enforcement points at the Act level (PDP). It also allows translation of policy instances into configurations to push on PEPs. This is also implicit in our scheme, but one should note that the PDP needs some contextual data available at the Orient level, mainly dealing with cartography (topology and inventory) of the information system, especially PEP capabilities.

3.2.1.3 Proposal for a threat response architecture

Confrontation of our requirements with AAA and OODA finally leads to the generic architecture presented in figure 3.4.

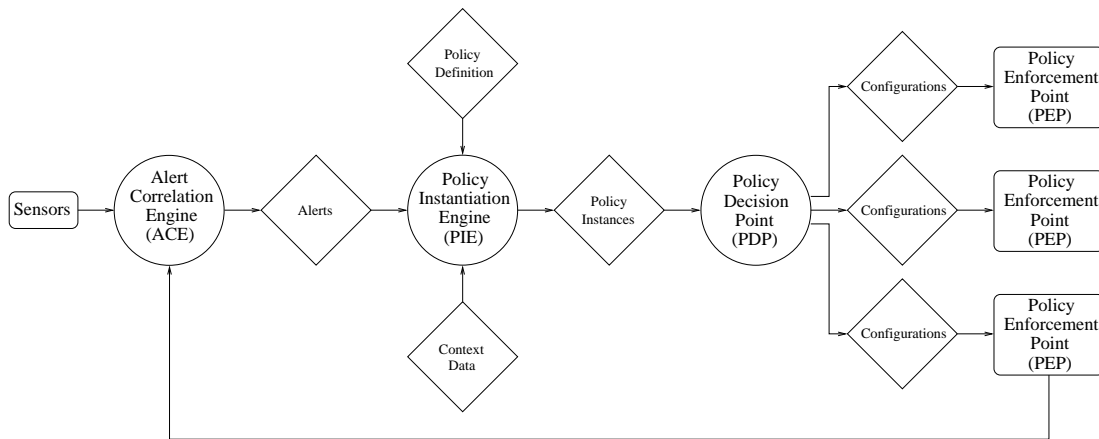


Figure 3.4: Threat response system generic architecture

Software or hardware modules are depicted by circles and messages and configuration information associated with our components by diamonds. We assume that any organization will deploy sensors and a security information management framework, from which we will collect alert information. This is depicted by the *sensor* block. The policy changes will be applied to *PEPs*, for example mail servers, firewalls or intrusion detection systems. It is therefore likely that some PEPs will also act as sensors.

3.2.2 Role of the components

3.2.2.1 Sensors

Sensors report events about the activity of the information system. As explained in Section 2.2, data collected by sensors may come from network packets, host log files and application log files. Sensors may be intrusion detection sensors (reporting events), intrusion detection systems (reporting low-level alerts, after a first analysis process), but also various hardware and software components of the infrastructure composing the information system, such as firewalls, routers, or hosts themselves, if we consider host log files analysis.

3.2.2.2 Alert Correlation Engine (ACE)

As explained in Section 2.2.2.1, information provided by sensors cannot be considered on its own. A fusion and aggregation process is necessary to characterize high-level alerts, describing as well as possible the situation. The ACE considers additional observed or predefined information about the information system. In particular, the knowledge of the characteristics (hardware and software) of the hosts and their function is necessary for false positives recognition, alert mitigation and severity assessment [75]. Additional techniques may be deployed, according to Section 2.2.2.1, so that we can consider that alerts reported by the ACE are reliable enough to envision automated response. Note that we mean by “reliable alerts” that (1) there is no false positive, and that (2) information contained into alerts is relevant. However, this does not mean that alerts are systematically fully qualified, which means that some valuable information regarding threat and/or response assessment may not be explicitly present in a given alert.

We formulate the requirement that the ACE must report the attacker (or at least what is known to be the source of the attack) as the source of the alert, and the victim as the target of the alert. At the packet level, this is true for a large majority of the alerts. A rare exception deals with *smurf attacks* which consist in sending ICMP *echo requests* to many hosts with a same spoofed source IP address which is in fact the actual victim of the attack, since the stimulated hosts are to send back *echo replies* leading the victim to denial-of-service. We assume that the ACE is able to analyze such exceptions and provide relevant higher-level alerts actually reflecting the victim as the target in the alert. It is sometimes difficult to trace the actual source of the attack, and we consider that information available in the alerts may sometimes be altered. At least, the ACE may possibly use the *spoofed* attribute of the *Source* class, or the *decoy* attribute of the *Target* class to indicate the relevancy of the information. We consider that response strategy should take this into account, so that information likely to be false are not to be considered in the response process.

3.2.2.3 Policy Instantiation Engine (PIE)

The Policy Instantiation Engine is responsible for the threat and response visions of the world. It is in charge of orienting response deploying a strategy with respect to attack/intrusion/threat. This means that the PIE should be able to evaluate potential threats considering attacks or intrusions given in the alerts.

The PIE is in charge of deploying a strategy of countermeasures activation. The strategy should provide the best adequate response to threat. For instance, the PIE should decide whether an attacker-centric response is better-suited or not compared to a victim-centric response. This notion of strategy is further discussed in the dissertation, so we do not give more details in this section.

The PIE also compiles the policy. It is the global decision point towards response. It considers a global policy definition and contextual data to dynamically activate policy instances. Contextual data reflect the vision of the world, in particular from the attack/intrusion/threat point of view, and is used to distinguish the implementation of the policy in a given context from another context.

3.2.2.4 Policy Decision Point (PDP)

The Policy Decision Point is a local decisional entity. It maps policy instances onto PEP capabilities, to decide what to actually enforce considering a given policy instance. For example, a policy instance may be enforced as a new firewall rule. The PDP should be able to determine which type of PEP is concerned by a given policy instance (*e.g.* a firewall), but also which implementation (*e.g.* a “Netfilter” firewall). A two-steps translation process is thus required with regard to both type and implementation of the PEPs.

Note that local strategy may be envisioned at the PDP level regarding policy instances. In fact, considering an elementary scenario which would consist in blocking access to a given service for a given user, one could imagine advanced response alternative scenarios, which would provide better convenience to users. For instance, informing users that the service is being closed soon and allowing them a grace period before closing would let them the possibility for correctly closing their session, which would probably avoid some undesirable side-effects.

3.2.2.5 Policy Enforcement Point (PEP)

Policy Enforcement Points are the entities actually running configurations reflecting current policy implementation. PEPs provide adjustment variables one can tune according to current policy requirements (to the current context). PEPs mainly deal with (1) authentication, (2) filtering, (3) encryption, (4) signature, (5) redirection/quarantine, but also with (6) infrastructure components, (7) patches, (8) application servers and (9) client software.

Authentication. Authentication servers and services provide multiple adjustment variables. First, most brutal response would consist in managing activation / deactivation of user authentication. For instance, in Microsoft environments, which build upon Kerberos using Active Directory, it is possible to realize dynamic reconfigurations of the LDAP Directory. A finer-grained response to threat is to constrain authentication means. When a service is threatened, authentication requirements can be elevated, so that it is not possible anymore to use standard password authentication, but certificates or biometry. A parallel should be made with provisional authorizations [53], in that users are still allowed to authenticate to the service, provided they have means to do so (certificate, fingerprint reader, etc.). For instance, PAM (Pluggable Authentication Module) on UNIX systems manages authentication modules providing different authentication means for various services (ftp, telnet, ssh, samba, etc.). PAM also proposes the possibility to ask users for a new password. This may be used when a brute-force password attack has been detected, to make users change their password, and maybe require a stronger password. Another example dealing with authentication is Single Sign-On (SSO). The purpose of SSO is to bring up convenience to users, which only authenticate once to be able to access multiple services, instead of being compelled to re-authenticate for each service. One should note that SSO provides the advantage of allowing continuity of (various) service(s) for users already logged on if a countermeasure aims at preventing new users from authenticating.

Filtering. It includes host and network firewall rules deployment at the network / transport level, but also possibly at the application level, dealing with user accounts, application commands, etc. Various access control mechanisms belong to this category, including the definition of Access Control Lists (ACLs) on routers and high-level switches, but also Network Access Servers (NAS).

Encryption. A simple way to enforce encryption requirements is to use secure protocols (HTTPS for encrypted web access, POPS or IMAPS for encrypted mail access, etc.). One may choose to provide both non-encrypted access and encrypted access when no particular threat is characterized, or non-encrypted access only, to benefit from optimum performance and convenience (no key exchange, no password required, etc.). On the contrary, if a threat to confidentiality and integrity is detected, one would probably choose to force the use of secured protocols. Note that availability threats may be treated by forcing the use of non-secured protocols, to provide better performance. Note also that it is possible to manage configuration of secured protocols, to ensure that encryption algorithms are adequate considering threat.

Signature. It provides means to verify the integrity and authenticity of a received message. Adjusting signature requirements to different confidence security levels manages the integrity confidence level depending on the current context. For instance, one may require no signature in a normal context, and force its use in a given threat context, adjusting both the desired algorithm and the length of the used key(s).

Redirection/Quarantine. In some circumstances, it may be too risky to deploy a countermeasure in the operational environment. In such cases, packets may be re-routed to mirror servers, allowing to test efficiency and stability of the countermeasures. Another application of redirection is to redirect attackers towards honeypots, which can let time to deploy countermeasures over operational equipments. In addition, honeypots may be used to study attacker behavior, which may be useful to characterize threats. Finally, redirection can be used for quarantine to isolate attackers. Redirection can be achieved by reconfigurations of routing tables. For instance, this may be realized through IOS (Internetwork Operating System) command line on Cisco routers, or with software routers using applications like Quagga. Quarantine may also be realized through the use of Virtual Local Area Networks (VLAN), as proposed in Ungoliant ¹.

Infrastructure components. Infrastructure components, like Dynamic Host Configuration Protocol (DHCP), Domain Name System (DNS), or Active Directory are essential for the information system to correctly operate. If DHCP is rendered unavailable, hosts may not be able to connect to the network. DHCP-based environment, hosts get their configuration, especially their network configuration, via DHCP at boot time. This means that hosts already active on the information system will keep service, at least for a limited time (DHCP lease). DNS exhibits similar properties, that is hosts

¹<http://ungoliant.sourceforge.net>

need to access DNS servers for hostnames resolution, in order to find DNS hostnames-IP addresses associations. Hosts are able to keep a certain number of already resolved addresses in their cache, but if a required association is not already known, a client has to request for it to a DNS server. This means that in case of unavailability of Domain Name System, hosts cannot connect to unknown DNS hostnames.

Patches. Applying software security patches is a form of policy enforcement, since it aims at preventing exploitation of software vulnerabilities, leading to violations of the security policy. Every host of the information system is eligible to software security patches application. A specific architecture is required to enforce such policy instances, requiring agents on all hosts. Microsoft Systems Management Server (SMS) may be used to deploy such updates in Windows environments.

Application servers. Application servers provide interesting ways to enforce policy instances in response to threat. Managing multiple paths to resources through different protocols related to the same activity is an interesting adjustment variable. Mail servers may for instance indifferently provide POP (Post Office Protocol) access and IMAP (Internet Message Access Protocol) access to mail, or forbid the use of a protocol in case of threat. Similarly, file servers may provide different access protocols, like CIFS (Common Internet File System) and NFS (Network File System). Web servers may prefer to permit HTTPS only in case of threat to confidentiality, and force the use of HTTP in case of availability issue, since it requires less network resources than HTTPS. Web application modules like PHP (PHP: Hypertext Preprocessor) and DAV (Distributed Authoring and Versioning protocol) can also be activated or de-activated according to threat and other requirements (*e.g.* performance and convenience).

Client software. Enforcing policy instances on client hosts requires the presence of agent software on clients, able to receive and apply new configurations. Response at the client-side are linked with notification of recommendations or obligations (*e.g.* “use a specific client application because another is currently threatened”), and configuration of client application (*e.g.* reconfigure a mail client to use POP instead of IMAP).

3.2.3 Scalability of the architecture

The suggested architecture presents the advantage of being scalable, both regarding PDPs and PIEs, as illustrated in figure 3.5.

Multiple PDPs. This may be required to deploy different local strategies. For instance, in figure 3.5, *Site1* has a single PDP, which means a single local strategy for local policy enforcement. However, in some circumstances, multiple local strategies may be required, to distinguish different physical or logical sub-parts of the site. One may also want to segment PDP capabilities according to PEP capabilities. *Site2* illustrates the deployment of multiple PDPs.

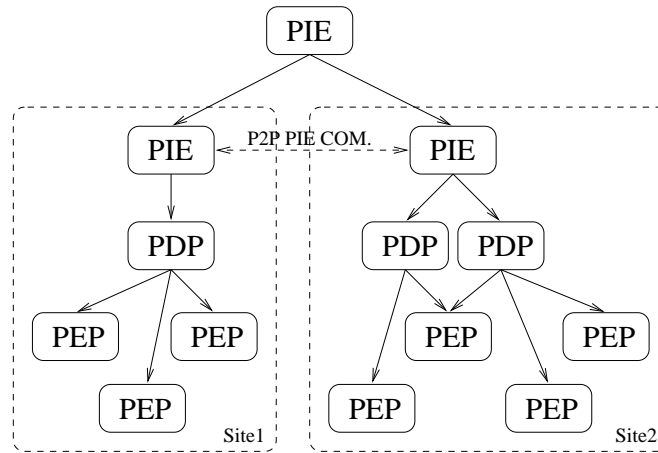


Figure 3.5: Scalability of the architecture

Multiple PIEs. In a large corporation, a common corporate generic policy may be defined for all sites. A super-PIE entity is responsible for distributing the generic policy to PIEs localized on different sites. The super-PIE may slightly differ from local PIEs. Information transmitted from the super-PIE to a local PIE semantically differs from information transmitted from local PIEs to PDPs. PIE-PIE communications mainly deal with generic policy distribution, whereas PIE-PDP communications is much more likely to deal with contextual instantiation of the policy. Peer-to-peer communication between PIEs may be envisioned for policy reconciliation purpose. However, this is left to the responsibility of the super-PIE.

In the following, we focus on the architecture presented in section 3.2.1, *i.e.* we consider that a single PIE provides policy instantiation for a single PDP which is in charge of pushing configurations to enforce policy to multiple PEPs.

3.3 Multiple Paths to Resources

In the following, we take inspiration from the concept of dependable computing, which appeared in the 1830's in the context of Babbage's calculating engine. In 1834, Dr. Dionysius Lardner [56] states that:

“The most certain and effectual check upon errors which arise in the process of computation, is to cause the same computations to be made by separate and independent computers; and this check is rendered still more decisive if they make their computations by different methods.”

Dependability has shown that redundancy and design diversity of the proposed services and resources are suitable solutions for fault-tolerance [52, 10]. Instead of deploying a single mail server, one could make the choice to provide a second one running different software, so that both would not be vulnerable to the same attacks, at least at

the software level. *Redundancy* deals with the use of redundant copies of hardware, data and programs, whereas *design diversity* prones the use of hardware and software components independently designed to meet similar requirements [9]. The most common technique to achieve design diversity is called “N-version programming” [4]. It consists in having software with identical initial specifications developed by different teams, different technologies, etc. The result is the independent generation of $N \geq 2$ software modules, called “versions”, which should not suffer from the same vulnerabilities.

In our approach, we build upon the fact that multiple paths to resources exist or may easily be provided in an operational environment. We bring up redundancy to our servers, clients, services, applications, or even information. We may also implement design diversity. Although this is an interesting idea to achieve fault-tolerance, we do not make use of “N-version programming”. Indeed, we do not want to develop new services or applications, we just use existing ones (COTS, or Components-Of-The-Shelf), because developing new applications is very costly, and because our major interest is threat response, and not fault-tolerance.

Our proposal is partly based on the fact that redundancy and design diversity are sometimes already present for various applications and services of the information system. We present the main parameters to manage multiple paths to resources through the use of a dynamic security policy, focusing on components, functionalities or information redundancy and diversity at three levels: (1) host, (2) service / application and (3) information.

3.3.1 Host level

Server side. Redundancy at this level is the deployment of a set of physical servers for a given service. Installing two mail servers maintains access to resource when one has been rendered unavailable because of a DoS (Denial-of-Service) attack, or in response to a confidentiality or integrity threat. Diversity in addition requires to deploy servers with different hardware, *e.g.* an Intel and a PowerPC architecture, in order to avoid both of the servers being threatened by similar hardware-linked vulnerabilities.

Client side. Client redundancy is implicit in a corporate environment where management of the workstations is centralized so that hosts rely on standard configurations, because each station indifferently ensures access to proposed services and required applications. In such an environment, users are not necessarily assigned to hosts, that is if a client is rendered physically unavailable, a user should be able to use indifferently another workstation to achieve his task. Some users may even own multiple workstations (*e.g.*, a desktop computer and a laptop computer). Diversity on the client side requires that users considered as “sensitive” have multiple workstations of different hardware.

3.3.2 Service / Application level

Diversity at the service / application level first depends on the operating system. Considering the OS used, software vulnerabilities are very likely to be different, as well as implementations of the TCP/IP stack, which renders hosts more or less sensitive to

a given attack. Dealing with application software, diversity can be partially obtained with multi-version software, *i.e.* by installing different versions of software, since vulnerabilities are likely to be specific to a given version or a restricted number of versions of software.

Server side. From the server point of view, applications are mainly services. Redundancy consists in deploying the same services over different servers. For instance, each mail server allows to use the POP service to read mail. Another kind of redundancy would be to launch multi-threaded POP, in order to assign a dedicated instance of the POP service to each user. Diversity at the service level may be achieved with the use of different services / protocols designed for a similar purpose. For instance, POP and IMAP are protocols which are both designed to provide mail reading. A mail server allowing both POP and IMAP by running adequate software would naturally provide alternatives when one of the services designed on a similar purpose is out of order.

Client side. Redundancy on the client side is achieved if the user is able to use different workstations, so that he may launch his application on different hosts. However, if a particular client software is threatened, it is very likely to be threatened whatever the workstation. Diversity is thus more suitable at the client level, allowing the user to access information through multiple similar (in term of service) applications. For instance, if Microsoft Outlook mail client is threatened, the user could access mail using Mozilla Thunderbird.

3.3.3 Information level

Server side. Redundancy is realized by replicating data. For instance, rather than having multiple mail services on multiple mail servers checking a single centralized mailbox, one could choose to automatically replicate the mailbox locally on each mail server. Diversity may be ensured via the management of different but similar information (in term of service), like different mailboxes. Thus, if a mailbox is forbidden by the response process, the user can still send email using another one.

Client side. Redundancy consists in having user accounts on different workstations. Diversity is obtained by configuring multiple accounts for users with different access rights. Such a technique provides at least degraded operating modes, instead of complete blocking of user accounts.

We observe that combinations of redundancy and diversity at the three levels provide multiple paths to resources. The ideal goal is to provide paths specific enough so that a threat affecting a path would not affect the other paths, or at least would not affect all of the other paths. In particular, varying both OS and service is more discriminant towards attacks, *e.g.* a web server running Microsoft IIS (Internet Information Service) under Windows will have very few common vulnerabilities with a web server running Apache under Linux. Obviously, this would be even better with two distinct hardware platforms [77].

Beyond availability, we also address confidentiality and integrity. It essentially depends on the characteristics of the considered similar services / protocols. For instance, we consider HTTP and HTTPS similar in term of service, since they both allow web access. However, HTTPS is more constraining than HTTP regarding confidentiality and integrity, since it uses certificates to establish the communication and communications are encrypted.

3.4 Conclusion

We presented in this chapter our proposal of an architecture of a threat response system. This architecture fits our requirements, consisting in allowing automated, dynamic and policy-based response. We have introduced the Policy Instantiation Engine (PIE) to the traditional model including a Policy Decision Point (PDP) and Policy Enforcement Points (PEPs). The global and generic definition of a policy is locally enforced. The policy is contextual, that is its rules are activated thanks to alerts reported by an Alert Correlation Engine (ACE), providing alerts characterizing attacks or actual intrusions.

We then showed through simple examples illustrating redundancy and design diversity that there often exists multiple paths to resources and services. Access control to these paths may be managed thanks to the dynamic policy. This is used to ensure automated and dynamic response to threat.

Chapter 4

Use of Or-BAC for threat response

Contents

4.1	Introduction	47
4.2	The Or-BAC model	48
4.2.1	Or-BAC advantages for threat response	48
4.2.2	Operational contexts	49
4.2.3	Mapping Or-BAC onto our architecture	51
4.3	Or-BAC extensions for threat response	52
4.3.1	Threat contexts	53
4.3.2	Minimal contexts	58
4.3.3	Context composition	58
4.3.4	Conflict resolution and priorities assessment	60
4.4	Conclusion	66

4.1 Introduction

We develop in this chapter how we make use of Or-BAC for threat response. We explain in a first section why Or-BAC is a good choice for our proposal. We insist on the organizational character of the policy, its abstraction allowing local enforcement conforming to global requirements, and the use of contexts to render the policy dynamic. We then explain in a second section how we extend Or-BAC not only to describe an operational policy, but also to ensure reaction requirements. In particular, we define threat contexts characterized by detected attacks or intrusions, we add a boolean context algebra, and we allow the definition of minimal requirements, used to constrain the policy implementation, so that some services/resources may not be degraded, whatever the threat level. We also explain how we manage conflict resolution between security rules and provide conflict resolution strategies enabling automated priority assessment.

4.2 The Or-BAC model

4.2.1 Or-BAC advantages for threat response

Most current security models such as DAC [46] or RBAC [71] can only be used to specify *static* security policies. When an intrusion occurs, the security administrator has to manually update the policy by removing obsolete security rules or inserting new security rules. Unfortunately, the time required for such a manual update is generally too long to represent an effective way to react to an intrusion. The administrator has also to update the policy again once the intrusion is circumvented to restore the policy in a state corresponding to a non intrusive context.

Our objective is to design a method to help the administrator in these tasks of updating the policy. For this purpose, we need a model to specify security policies that dynamically change when a threat is detected. In the absence of attack or intrusion characterizing threat, the policy to be applied is said to be an *operational* policy and corresponds to *nominal* contexts. Other contexts must be defined to specify additional security rules to be triggered when threats are detected. This is similar to provisional authorizations [53]; contexts are linked to the history of reported attacks and intrusions, and activate provisional security rules. Some of these security rules may correspond to *permissions* (positive authorizations) but more often they will represent *prohibitions* (negative authorizations). The prohibitions will be automatically deployed over the information system as a reaction to the threat. This may correspond to automatically insert a new deny rule in a firewall. The concept of provisional authorization provides means to envision an approach going significantly further than traditional access control. Indeed, provisional authorizations mean that access to resource is allowed provided certain conditions. For instance, we may require encryption to access a given resource by constraining the access control rule with a specific context related to a specific encryption requirement. Similarly, we may constrain access to a resource considering different types of authentication, that is, for instance, allow access only if the user is able to present a certificate to authenticate.

The security policy model must provide means to manage conflicts between permissions and prohibitions. In particular, the policy associated with a nominal context can include *minimal* security requirements. These minimal requirements must not be overridden, even when an intrusion is detected. For instance, they may include minimal availability requirements. Of course, these minimal requirements may conflict with contextual rules associated with the detection of a given intrusion. In this case, simple strategies such as prohibition takes precedence or permission takes precedence will not be appropriate to solve the conflict. Instead, the model must include the possibility to specify high level conflict management strategies to find the best compromise between conflicting rules [44].

The model must also provide an abstract and global view of the security policy. This is the purpose of the PIE (Policy Instantiation Engine) to manage this global security policy. The PIE will have to clearly separate the global policy from its implementation in the PEPs (Policy Enforcement Points). In particular, the conflicts are to be solved at the abstract level before generating PEPs configurations. Unfortunately, most security

models do not provide such a clear separation.

In this work, we suggest using an approach based on the Or-BAC model [62], since it fulfills the requirements formulated previously. Moreover, from an operational point of view, Or-BAC is also interesting because it abides to the Datalog restrictions [80], which means that we can prove that it is possible to decide in polynomial time that a conflict management strategy is effective.

4.2.2 Operational contexts

Let C be a set of contexts. We consider a set $OC \subseteq C$ of *operational* contexts. Operational contexts aim at describing traditional operational policy. Note that $c \in OC$ *is active* does not mean that there is no attack or intrusion, but that *it is possible that there is no attack or intrusion*. Indeed, *operational* contexts do not provide any information about threats.

4.2.2.1 Nominal contexts

For the sake of simplicity, we consider that, in the absence of characterized threat, that is in the absence of attack or intrusion, the organizational policy is defined using *nominal* contexts. Thus, we assume that *nominal* $\in OC$. However, in a more realistic setting, this policy may depend on other contexts [27], such as *temporal* contexts, *spatial* contexts, *user-declared* contexts, *prerequisite* contexts and *provisional* contexts.

Nominal contexts are always active by definition. This means that rules defined with nominal contexts are always considered active in the policy compilation process. However, it does not mean that nominal rules are always instantiated, since the conflict resolution step at the PIE level takes into account rule priorities, so that rules having higher priority than nominal contexts priority shadow nominal rules.

Listing 4.1 defines the generic *nominal* context, which is always active, “_” indicating that the statement is right for any subject, action and object.

Listing 4.1: Nominal context definition

```
hold (corp ,_ ,_ ,_ , nominal ) .
```

4.2.2.2 Temporal contexts

Temporal contexts are used to express temporal constraints for a subject making an action on an object. Typically, it refers to the time the subject is requesting for an access to the system. For instance, *working_hours* $\in OC$ defines policy rules activated during working hours only. Temporal contexts may be of great interest to balance confidentiality, integrity and availability requirements. Indeed, during working hours, one may want to put the light on availability, and to focus on confidentiality and integrity during non-working hours. An example of temporal context definition is given by listing 4.2.

Listing 4.2: Temporal context definition

```

hold (corp, __, __, __, working_hours) :-
    globalclock (DayClock, TimeClock),
    TimeClock >= '07:00:00',
    TimeClock < '20:00:00',
    DayClock != 'saturday',
    DayClock != 'sunday'.

```

4.2.2.3 Spatial contexts

Spatial contexts are used to express spatial constraints for a subject making an action on an object, especially dealing with the subject location. For instance, it may be used to distinguish *internal_access* from *external_access*, and thus provide a different policy implementation considering the location of the subjects. Note that spatial context may either be physical spatial contexts or logical spatial contexts. For instance, a physical spatial context may be *in_office*, whereas a logical spatial context may be *in_vlan0* or *in_dmz*. From a reaction point of view, this may permit to distinguish response to insider subjects from response to outsider subjects, which cannot be treated the same way, because enforcement points under control remain inside the administrative domain of the information system.

Spatial context definition requires the knowledge of the information system cartography, that is, inventory of the hosts, logical topology and service dependencies, and possibly geographical location.

Listing 4.3: Spatial context definition

```

hold (corp, S, A, O, internal_access) :-
    host_ip (S, Ip),
    ip_in_range (Ip, 192.168.1.0, 192.168.1.255).

```

4.2.2.4 User-declared contexts

According to the role a subject plays in the organization, he may be able to declare that is is making a particular activity in a given context. This kind of context is called user-declared contexts, and may allow the subject to benefit from specific permissions linked with the activity he is making. Thus, the activation of the context is not derived from dynamic external parameters of the environment, such as time or cartography, but directly declared by the user considering his current activity. For instance, to define new policy rules, the security administrator must be authorized to read and write in the policy repository. One may imagine a scenario in which such authorization is given only when the administrator declares to be in a context *policy_modification*. This allows a better partitioning of access control, providing authorizations only when necessary.

User-declared context definition requires the knowledge of subject purpose. According to [27], user-declared contexts require the definition of a *Purpose* sub-view, which can for instance be *system_administration*. Then, a *recipient* is any subject taking advantage of view *system_administration*. Additional constraints are then to be defined, relatively to the considered object, such as *topic* in the example (listing 4.4). In this case, we express the fact that in organization *corp*, subject *S* performs action *A*

on object O in context *policy_modification* if there is an object P belonging to view *system_administration* having S as recipient and *policy_management* as topic.

Listing 4.4: User-declared context definition

```
hold (corp , S , A , O , policy_modification) :-
    use (corp , P , system_administration) ,
    recipient (P , S) ,
    topic (P , policy_management) .
```

4.2.2.5 Prerequisite

Prerequisite contexts are used when specific constraints need to be satisfied to grant permission (or prohibition or obligation) to make an action on an object. For instance, a system administrator may be authorized to modify a policy, but only provided the considered repository is under his responsibility. Thus, context *admin_repository* may be used to express such a requirement.

Prerequisite context definition requires a system database. For instance, knowing who is responsible for a specific policy repository management allows us to control access according to such requirements, as illustrated in 4.5.

Listing 4.5: Prerequisite context definition

```
hold (corp , S , A , O , admin_repository) :-
    use (corp , O , policy_repository) ,
    is_responsible_for (S , O) .
```

4.2.2.6 Provisional

Provisional authorizations have been introduced in [53, 48]. The main idea is to authorize subjects to perform the desired access provided a certain condition is satisfied. For instance, a user may be authorized to access sensitive information provided access and actions realized are logged; a user may be allowed to authenticate provided he presents authentication means strong enough considering threat level, etc. An example of provisional context is *pki_auth*, which requires the use of a public key infrastructure to authenticate. This means that users must present a key to perform access.

Listing 4.6: Provisional context definition

```
hold (corp , S , A , O , pki_auth) :-
    authentication_capability (S , pki) .
```

4.2.3 Mapping Or-BAC onto our architecture

The mapping of Or-BAC on our architecture leads to figure 4.1. The PIE is aware of the *generic Or-BAC policy* (abstract Or-BAC rules), the *context definitions* that is how Or-BAC *hold* facts are activated, and additional *context data*, such as various information about the hosts, time, vulnerability references, etc. It ensures threat characterization, strategy application and policy compilation. The PDP receives policy

instances (concrete policy rules) and processes local decision to enforce the policy according to PEP capabilities. Note that additional data about PEP capabilities is also needed by the PDP, including PEP functionalities (*e.g.* firewall), services (*e.g.* http service) and implementations (*e.g.* netfilter, apache, etc.).

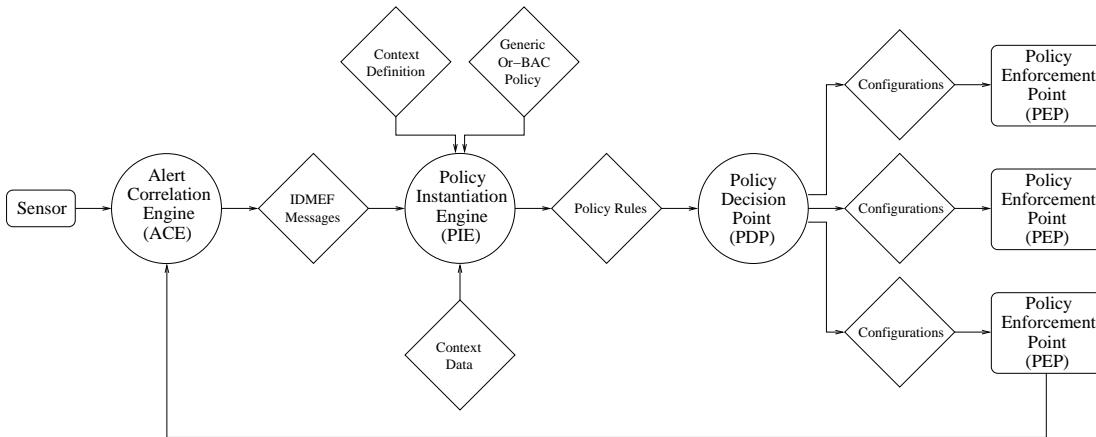


Figure 4.1: Threat response system architecture

We introduce in the next section the necessary extensions to Or-BAC in order to ensure response, especially dealing with threat context activation and deactivation, Or-BAC entities modeling to fit response use cases, and conflict resolution. We also discuss the need for defining minimal contexts ensuring minimal requirements.

4.3 Or-BAC extensions for threat response

The core of our proposal is to manage contexts according to threat information. *Threat* contexts refer to contexts used to characterize threats and to provide threat response. Examples of threat contexts may in fact refer to attacks or intrusions (successful attacks). For instance, *syn_flooding* is considered a threat context, and *pop_attack* is considered a threat context. Note that the ACE should be designed to report intrusions, but this does not exclude the knowledge of sub-parts of scenarios, *e.g.* elementary attacks, especially through the use of the *CorrelatedAlert* class of IDMEF.

Contexts belong to three categories: *operational*, *threat* and *minimal*. Contexts are organized hierarchically so that, when a conflict occurs, security rules associated with contexts higher in the hierarchy will override security rules associated with lower contexts. We assume that *operational* contexts are lower than *threat* contexts which are in turn lower than *minimal* contexts.

We present in this section how we introduce threat contexts to define not only an *operational* policy, but also a *reaction* policy, that is part of the policy which deals with threat. In particular, we discuss on which basis threat contexts should be activated. We propose a simple solution to deal with their deactivation, and provide examples of threat contexts. We then add a part of the policy which is called the *minimal* policy. It

aims at defining and activating *minimal* contexts which ensures minimal requirements. Then, we add the possibility to express composed contexts, through the definition of a boolean context algebra. We finally address conflict resolution and dynamic priority assessment in a last section, building upon [44].

4.3.1 Threat contexts

Threat contexts regroup every context which allows to characterize threat. For instance, *pop_intrusion*, *brute_force_pwd_attack* and *synflooding* are threat contexts. We consider a set $TC \subseteq C$ of *threat* contexts. A context $c \in TC$ is activated when a given threat is detected. This means that $c \in TC$ *is active* necessarily implies that *there is an attack or an intrusion*. It is associated to a set of new security rules that apply to fix the threat.

4.3.1.1 Threat context activation

Activation of threat contexts raises two major issues: (1) which information is available to characterize threats, and (2) what do we do with this information to characterize threats at the policy level. Context activation deals with the activation of complete *hold* facts, *i.e.* not only context, but also organization, subject, action and object. If c is a *threat* context, then subject s , action a and object o must be correctly mapped onto information available from threats, including threat source, threat classification and threat target. This allows a full characterization of the threat.

We use IDMEF messages to select contexts and policy rules to activate. Among the IDMEF message attributes, we particularly use:

CreateTime The CreateTime timestamp indicates the time at which the alert was created and is mostly relevant to ensure temporal consistency regarding alert processing and to manage threat context lifetime.

Assessment The Assessment attribute carries information related to the risk of the attacker's actions. This is used for response strategy and to compute threat context lifetime, depending on the severity and type of the alert.

Classification The Classification provides information about the mechanism of the attack. This is used to activate contexts, and many additional valuable information about role, activity and view may also be deduced from the vulnerability reference.

Target The Target attribute carries information about the victim. This is used to relate the alert to the views and activities of the Or-BAC policy rules, and possibly to activate contexts.

Source The Source attribute carries information about the attacker. This is relevant for roles in the Or-BAC policy rules if the attacker is an insider, and possibly for context activation.

4.3.1.2 Mapping alert information on hold predicates

Mapping alert information to context requires creating transformations from alert content to instantiated triples (*Subject, Action, Object*) by writing the appropriate *hold* predicates. Unfortunately, the naive mapping from *IDMEF.Source* to *Subject*, from *IDMEF.Classification* to *Action*, and from *IDMEF.Target* to *Object*, is far from sufficient, and this for three reasons:

1. We need a mapping that has variable granularity, to take into account the different scope of different attacks. For example, a distributed denial-of-service on all areas of the network needs to be handled differently than a targeted brute-force password-guessing attack.
2. Alert information is sometimes incomplete; sources can be inexistent, incomplete or wrong. Multiple classifications may provide inconsistent information, such as conflicting attack references, may cover multiple attacks, or may not be modeled in our system. We need to specify what happens when an alert is incomplete.
3. We also need to specify complex responses mechanisms, that take into account environmental information, expressing complex reaction scenarios. For example, a complete response system may require moving from HTTP to HTTPS, and hence opening and closing multiple network accesses, and starting and stopping multiple services.

		Subject	Action	Object	Context	Lifetime
Createtime	ntpstamp					X
Source	Node.name	X				
	Node.Address.address	X				
	Node.Address.netmask	x				
	User.Userid.name	X				
	Process.name	X			x	
	Service.name				x	
	Service.port				x	
Target	Node.name			X		
	Node.Address.address			X		
	Node.Address.netmask			x		
	User.Userid.name			x		
	Process.name			x	x	
	Service.name		X		x	
	Service.port		X		x	
Classification	Reference.name	x	x	x	X	
Assessment	Impact.severity				x	X
	Impact.type				x	X

Table 4.1: Mapping IDMEF classes on Or-BAC parameters

Table 4.1 lists the main elements which should be taken into account in IDMEF to provide relevant mappings. 'X' means that the considered information is very likely to be considered in the mappings. 'x' means that either information is less likely to be found or that it is of secondary interest regarding mappings. For example, beyond contexts, many elements may be deduced from the reference, like the targeted service.

For example, reference *CVE-2005-1133* implies that the attack targets service *pop3* (*tcp/110*). This introduces the possibility to discover information which are not explicitly present in the alerts. This may also be used to detect conflicts between information. However, we assume that alerts are correctly qualified, which means for example that the system cannot receive an alert with reference *CVE-2005-1133* and service *imap*. We do not also aim at considering such deduction if the information about the target service is already explicitly present in the Target class. Note that not all IDMEF parameters are taken into account in this table. A complete and systematic mapping of every IDMEF parameter is part of the future work.

4.3.1.3 Threat context deactivation

Deactivating threat contexts is used to revoke countermeasures once threats are no longer present. We currently manage static context lifetimes, computed thanks to IDMEF alerts assessment attributes. We use the severity and type attributes of the IDMEF *Assessment.Impact* class to derive the duration of the context activity according to the matrix defined in table 4.2.

Impact severity Impact type	info	low	medium	high	Comment
admin	1	2	4	8	This is the most severe case. We are not currently handling scans, as they do not result in compromise.
dos	1	2	3	4	
file	0	1	2	3	
recon	0	0	0	0	
user	0	1	2	4	
other	0	0	1	2	

Table 4.2: Intrusive context lifetime according to IDMEF impact severity and type, in minutes

When an alert occurs, it is asserted for a certain duration. The corresponding context is activated with the expiration date set according to the table. While this alert remains stored in the system, the context remains active. When the lifetime expires, the alert is removed from the database, and the context is deactivated, unless another instance of the alert has been received in the meantime. Both asserts and retracts should trigger a re-evaluation of the security policy.

The values of table 4.2 have been defined according to expert knowledge of the risks incurred by each protocol. We currently use the same matrix for evaluating the risk incurred by each access mechanism; the variation in risk associated with each individual protocol is handled by the proper setting of the impact severity attribute. Note however that this table is only an example and that values may differ depending on the considered application and the corresponding risk analysis.

4.3.1.4 Threat contexts examples

Attack (or intrusion) classes are associated with threat contexts. We propose here two examples of threats. For each threat, a security rule is defined in the reaction

policy. Activation of corresponding contexts (*hold* facts) leads to the instantiation of a policy instance in response to threat. We build upon figure 4.2 to model roles, activities, views and contexts. Ellipses represent abstract entities, whereas round boxes materialize concrete entities (*i.e.* instances of abstract entities).

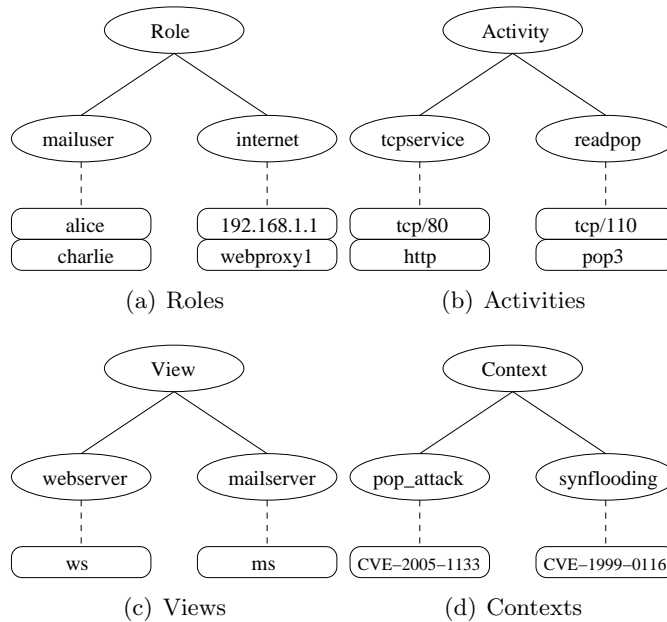


Figure 4.2: Or-BAC model for threat context examples

Syn-flooding attack. A Syn-flooding attack towards a webservers is reported by an IDMEF alert with (1) a classification reference equal to CVE-1999-0116 (corresponding to the CVE reference of a Syn-flooding attack) and (2) the target is attacked through a service whose name is *http* (or port is *tcp/80*) and (3) the target corresponds to a network node whose name is *ws*, then the *synflooding* context is active for *http* action on *ws* object. The corresponding translation in Prolog can be found in Listing 4.7.

Listing 4.7: `syn_flooding` context definition

```

hold(corp, _, http, ws, synflooding) :-
    alert(CreateTime, Source, Target, Classification),
    reference(Classification, 'CVE-1999-0116'),
    service(Target, http),
    hostname(Target, ws).
  
```

Notice that, since the intruder is spoofing its source address in a Syn-flooding attack, the subject corresponding to the threat origin is not instantiated in the *hold* predicate (represented by “_”).

When an attack occurs and a new alert is launched by the intrusion detection system, (a) new fact(s) *hold(org, s, a, o, c)* is (are) derived for threat context *c*. So, *c* is now active and the security rules associated with this context are triggered to react to the intrusion.

Notice that our approach provides *fine-grained* reaction. Consider a network where a given host *ws* is assigned to the role *webserver*. Assume that a Syn-flooding attack is detected against this host on port *tcp/80*, which corresponds to service *http*. In this case, we shall derive the following fact:

- *hold(org, _, http, ws, synflooding)*: means that host *ws* is now in the threat context *synflooding* through *http*.

Since the *synflooding* context is now active, security rules associated with this context are triggered. For instance, the following security rule matches the *synflooding* context:

- *security_rule(prohib, org, internet, tcpservice, webserver, synflooding)*: means that, in the threat context *synflooding*, *internet* is prohibited to perform *tcpservice* activity on the *webserver*.

However, only host *ws* (whose role is *webserver*) is in the context of *synflooding* through *http* (which is a *tcp* service). As a consequence, the reaction will not close every *tcp* service from the Internet to every web server. Instead, the reaction in this case will be limited to close *http* from the Internet to host *ws*.

Pop reconnaissance attack. An internal attacker is attempting a reconnaissance attack on a *pop3* server in order to determine valid users. The reference CVE-2005-1133 is an instance of such an attack for a *pop3* server in IBM iSeries AS/400.

The *pop_attack* context is defined by an IDMEF alert with (1) a classification reference equal to CVE-2005-1133 (corresponding to the CVE reference of a *pop* reconnaissance attack) and (2) the target is attacked through a service whose port is *tcp/110* (or name is *pop3*) by (3) a source that corresponds to a mail user whose name is *charlie* and (4) the target corresponds to a network node whose name is *ms*, then the *pop_attack* context is active for *charlie* subject making *tcp/110* action on *ms* object. Notice that we face here an internal attack and we consider that the diagnostic has revealed that the source is not a decoy, so we are able to instantiate the subject being the source in the *hold* predicate. The corresponding translation in Prolog can be found in Listing 4.8.

Listing 4.8: *pop_attack* context definition

```
hold(corp, charlie, 'tcp/110', ms, pop_attack) :-
    alert(CreateTime, Source, Target, Classification),
    reference(Classification, 'CVE-2005-1133'),
    hostname(Source, charlie),
    process(Target, 'tcp/110'),
    hostname(Target, ms).
```

In this case, we shall derive the following *hold* fact:

- *hold(org, charlie, tcp/110, ms, pop_attack)*: means that host *ms* is now in the threat context *pop_attack* through port *tcp/110*, the attacker being user *charlie*.

Since the *pop_attack* context is now active, security rules associated with this context are triggered. For instance, let us assume that there is the following security rule:

- *security_rule(prohib, org, mailuser, readpop, mailserver, pop_attack)*: means that, in the threat context *pop_attack*, a *mailuser* is prohibited to perform *readpop* activity on the *mailserver*.

This security rule is triggered once the *pop_attack* context is active. However, only host *ms* (whose role is *mailserver*) is in the context of *pop_attack* through port *tcp/110* (or *pop3* service, which are *readpop* actions) for subject user *charlie* (which belongs to the *mailuser* role). Alike the previous example, the reaction in this case will be limited to forbid port *tcp/110* to host *ms*, but for user *charlie* only.

These two examples illustrate the fact that, in our approach, we can associate threat contexts with *general* security rules. Fine-grained instantiation of the intrusion can be used to limit the reaction to those entities that are involved in the attack (as an intruder or a victim). The presented listings could be generalized by replacing constants by variables. For instance, in listing 4.7, it is possible to replace the constant *ws* by a variable, and similarly for constants *charlie* and *ms* in listing 4.8.

4.3.2 Minimal contexts

We consider the set $MC \subseteq C$ of *minimal* contexts. Minimal contexts define high priority exceptions in the policy, describing minimal operational requirements that must apply even when in case of characterized threat.

Minimal contexts are especially used to maintain availability. For instance, a minimal requirement can be to ensure that there is always an available path to resources, whatever the threat. We provide further details on *minimal* contexts in Chapter 5 and Chapter 7 through the email use case application.

4.3.3 Context composition

Expressing fine-grained contexts is of major interest, in particular to characterize threats. However, managing specific atomic contexts would rapidly become difficult since it would result in a huge number of definitions. A context algebra provides a way to combine atomic contexts through a boolean algebra. The following basic functions are provided to manipulate composed contexts:

Negation : $n(c) \leftrightarrow$ context *c* is **not** active

Conjunction : $\&(c1, c2) \leftrightarrow$ context *c1* **and** context *c2* are active

Disjunction : $v(c1, c2) \leftrightarrow$ context *c1* is active **or** context *c2* is active

This algebra allows the expression of composed contexts based on the composition of atomic contexts, ensuring an easy way to define fine-grained security rules. Note that

contexts entering in the composition of *composed* contexts are simply named *composing* contexts.

Composed contexts in Or-BAC have already been envisioned in [27]. However, managing security rules with composed contexts requires the ability to deal with such context priorities. We now analyze the possible combinations, and we explain how to define the priority of a composed contexts given its composing contexts. For each case, we give examples for a better understanding of composed context priorities.

Definition 1 *Since it is possible that there is no attack or intrusion in a negative context, the negation of a context, whatever its category, is an operational context.*

Property 1 *The priority of a negative context is equal to the priority of an operational context. Consequently, the priority of a negative context is lower than the priority of a threat context, and lower than the priority of a minimal context.*

Let us consider $c_1 \in OC$, $c_2 \in TC$ and $c_3 \in MC$. According to definition 1, one can state that $n(c_1) \in OC$, $n(c_2) \in OC$ and $n(c_3) \in OC$. Now, according, to property 1, one can state that $n(c_1)$, $n(c_2)$ and $n(c_3)$ have a priority of an operational context. Thus, they have a lower priority than threat and minimal contexts.

Ex. $n(\text{working_hours})$, like working_hours , is an operational context; $n(\text{pop_attack})$, negation of pop_attack , is an operational context. Thus, $n(\text{working_hours})$ and $n(\text{pop_attack})$ have both a lower priority than pop_attack , which is a threat context.

Definition 2 *The conjunction of two contexts belonging to the same category belongs to this category.*

Property 2 *The priority of the conjunction of two contexts belonging to the same category is the priority assigned to this category.*

Let us consider $c_1 \in OC$ and $c_2 \in OC$. According to definition 2, one can state that $\&(c_1, c_2) \in OC$. Now, according to property 2, one can state that $\&(c_1, c_2)$ has the priority of an operational context.

Ex. $\&(\text{working_hours}, \text{in_dmz})$ is the conjunction of a temporal (thus, operational) and a spatial (thus, operational) context. Consequently, $\&(\text{working_hours}, \text{in_dmz})$ is an operational context.

Now, let us consider $c_3 \in TC$ and $c_4 \in TC$. One can state that $\&(c_3, c_4) \in TC$ and that its priority is higher than operational, but lower than minimal.

Ex. $\&(\text{pop_attack}, \text{syn_flooding})$ is the conjunction of two threat contexts. Consequently, $\&(\text{pop_attack}, \text{syn_flooding})$ is a threat context.

Definition 3 *The conjunction of two contexts belonging to different categories belongs to the category of the composing context having the highest priority.*

Property 3 *The priority of the conjunction of two contexts belonging to different categories is the highest priority of the composing contexts.*

Let us consider $c_1 \in TC$ and $c_2 \in OC$. According to definition 3, one can state that $\&(c_1, c_2) \in TC$. Now, according to property 3, one can state that $\&(c_1, c_2)$ has the priority of a threat context.

Ex. $\&(pop_attack, working_hours)$ is the conjunction of a threat context and an operational (temporal) context. Since a threat context has a higher priority than an operational context, $\&(pop_attack, working_hours)$ is a threat context, and thus it has the priority of a threat context.

Dealing with the disjunction is more difficult, in particular with two contexts belonging to different categories. Consider $c_1 \in OC$ and $c_2 \in TC$. Determining to which category $v(c_1, c_2)$ belongs requires to consider which composing context among c_1 and c_2 is activating $v(c_1, c_2)$, since c_1 and c_2 do not have the same priority. If $v(c_1, c_2)$ is active because c_1 is active, this means that $v(c_1, c_2)$ is an operational context, like c_1 . On the contrary, if $v(c_1, c_2)$ is active because c_2 is active, this means that $v(c_1, c_2)$ is a threat context, like c_2 . Moreover, it is possible that $v(c_1, c_2)$ is active because c_1 and c_2 are both active. In this case, $v(c_1, c_2)$ belongs to the category of the composing context having the highest priority.

To avoid the issue of active context determination, we choose to automatically split the security rules defined with a disjunctive context into a set of equivalent rules, each one being defined for each composing context of the disjunction. We observe that a security rule defined with a disjunctive context $v(c_1, c_2)$ is logically equivalent to the conjunction of two security rules respectively defined with context c_1 and with context c_2 . Therefore, we first convert contexts to Disjunctive Normal Form (DNF), that is as a disjunction of conjunctions, and then write the set of equivalent rules.

Based on the defined algebra, it is possible to envision not only the composition of atomic contexts, but also the composition of composed contexts, so that one can define rules triggered by fine-grained contexts expressing accurately the security requirements. For instance, one could express a prohibition for a role *user* to make the activity *read_pop* on the view *mail_server* in the context:

$$\&(v(remote_access, \&(internal_access, n(working_hours))), pop_attack),$$

that is either in a context of pop attack **and** remote access, **or** in a context of pop attack **and** internal access on non-working hours.

4.3.4 Conflict resolution and priorities assessment

4.3.4.1 General principles

Before addressing the problem of conflict resolution and dynamic priority assessment, we should remind the notion of conflict among a set of access control rules.

Definition 4 *A conflict occurs among a set of access control rules when a subject is both permitted and prohibited to perform an action on an object.*

When a conflict occurs between two access control rules, a conflict resolution strategy must be defined to decide which rule takes precedence.

Definition 5 *A conflict resolution strategy (CRS) is a rule (or set of rules) which allows the system to decide which rule takes precedence when two rules are conflicting.*

In Or-BAC, one has to distinguish *prima facie* authorizations from *actual* authorizations, as explained in [44]. The abstract level defines on the one hand *prima facie* authorizations, which are not necessarily instantiated at the same time. Indeed, security rules are activated only given certain conditions, *i.e.* when the corresponding contexts hold for given (subject, action, object) triples. When such a context holds, an *actual* authorization is derived at policy compilation time at the concrete level. One should thus distinguish *actual conflicts*, occurring at the concrete level, from *potential conflicts*, characterized at the abstract level. Potential conflicts deal in fact with the coexistence of *prima facie* rules that lead to actual conflicts if their conditions of activation are simultaneously satisfied.

Definition 6 *A potential conflict appears between two prima facie authorizations if their simultaneous activation lead to actual conflict.*

Solving potential conflicts is realized using a global conflict resolution strategy.

Definition 7 *A global conflict resolution strategy (GCRS) is a rule (or a set of rules) which allows the system to decide which rule takes precedence when two rules are potentially conflicting.*

In [44], Cuppens & *al.* make the choice to implement conflict resolution at the abstract level, that is in fact, potential conflict resolution. This allows to ensure *a priori* that there will not exist any actual conflict at policy compilation time. In addition, since CRS is applied *a priori* to assess rule priorities, this simplifies the compilation process, reducing complexity regarding conflict resolution computation. With this approach, the policy instantiation engine considers security rules which are correctly ordered, so that it just has to apply the pre-computed priorities to solve actual conflicts.

Definition 8 *If all actual conflicts can be solved by a given strategy, it is simply called an effective strategy.*

Conflict resolution in Or-BAC is addressed through the definition of the *prioritized Or-BAC model*. It consists in the adjunction of a priority parameter in the security rules. The purpose of a conflict resolution strategy is thus to assess a partial order between potentially conflicting *prima facie* authorizations. In the following, we consider the two following potentially conflicting security rules for illustration purpose:

$$\begin{aligned} sr_1 &= sr(\text{permission}, org, r_1, a_1, v_1, c_1, p_1), \\ sr_2 &= sr(\text{prohibition}, org, r_2, a_2, v_2, c_2, p_2). \end{aligned}$$

An order of priority is defined over p_1 and p_2 , indifferently denoted respectively $\mathcal{P}(sr_1)$ and $\mathcal{P}(sr_2)$. If $p_1 \prec p_2$, we say that sr_2 takes precedence over sr_1 . Note that since all rules are not potentially conflicting, a partial order is sufficient to ensure an effective strategy.

Note also that p_1 and p_2 may be specified manually by the security operator. Sometimes, this is necessary, because there may exist cases where no CRS is able to address conflict resolution. However, the actual interest of a CRS is to simplify the work of the operator, since defining such priorities *a priori* is a deeply tedious task. Note that each time a CRS matches the given security rules, priorities which are automatically derived by this CRS override manually defined priorities.

A first constraint has to be defined regarding antisymmetry of \prec relation, in order to avoid inconsistencies between rule priorities, that is two strategies cannot state that both $p_1 \prec p_2$ and $p_2 \prec p_1$. This is represented by the following error: $p_1 \prec p_2 \wedge p_2 \prec p_1 \rightarrow error$.

In [44], conflict resolution is ensured through the definition of *separation constraints* and *inheritance mechanisms* in the Or-BAC model.

Separation constraints. Separation constraints are necessary because a subject can potentially be empowered in two different roles, an object can be used in two different views, an action can be considered in two different activities, and two different contexts may be activated simultaneously. Considering this, the fact is that every pair of sr_1 and sr_2 as previously defined may be potentially conflicting. On the contrary, defining for instance a separation constraint indicating that a subject cannot be empowered into both roles r_1 and r_2 allows to ensure that sr_1 and sr_2 will not generate any conflict. Note that such rules are said to be *unrelated rules*, since they are not applicable to the same conditions.

Inheritance. Inheritance deals with the notion of exception. Considering the defined hierarchies, sr_2 is an exception to sr_1 if it is lower in the hierarchy. In particular, sr_2 is a *strict exception* to sr_1 if r_2 is a sub-role of r_1 , a_2 is a sub-activity if a_1 , v_2 is a sub-view of v_1 and c_2 is a sub-context of c_1 . In such a case, we say that sr_2 is more *specific* than sr_1 , and that sr_2 takes precedence over sr_1 by mean of specificity. Such rules are said to be *redundant rules* in the case of similar decisions (both permissions, or both prohibitions). Note also that there is no need for all parameters (role, activity, view and context) to be strict exceptions. For instance, there may be some equal parameters between sr_1 and sr_2 . In some cases, it is also possible that some parameters of sr_1 are exceptions to the corresponding parameters of sr_2 and that other parameters of sr_2 are exceptions to the corresponding parameters of sr_1 . In such a case, rules are said to be *correlated* [3]. This does not mean that there is not potential conflict. However, this case is generally not managed by a CRS, since hierarchical specificity is not sufficient to decide which rule takes precedence. Priority assessment is thus often left to the responsibility of the operator.

4.3.4.2 Conflict resolution in our approach

Now, with respect to our approach, we need to consider additional parameters to ensure dynamic priority assessment and conflict resolution. In particular, we consider *criticity* to assess context priority between the three defined categories of contexts, namely *operational*, *threat* and *minimal*. We also need to consider context composition, both regarding *criticity* and *specificity*. Note however that we only consider here negation and conjunction, since we choose to split security rules according to disjunctions¹.

We define the following construction rules for contexts:

1. If c is an atomic context, then c is a well-formed context (WFC),
2. If c is an atomic context, then $\neg c$ is a WFC,
3. If c_1 is a WFC and c_2 is a WFC, then $c_1 \& c_2$ is a WFC.

Criticity. Criticity deals with context category. We define categories so that a rule declaring a *minimal* context must have a higher priority than a rule declaring a *threat* context, which must have a higher priority than a rule declaring an *operational* context. Let C be the set of well-formed contexts (WFC). We define the \mathcal{L}_c operator to assess the level of *criticity* of contexts, so that:

$$\mathcal{L}_c : C \rightarrow \{ope, threat, min\} \text{ avec } ope < threat < min$$

Definition 9 We define the *criticity relation* as follows: $c_1 <_c c_2 \leftrightarrow \mathcal{L}_c(c_1) < \mathcal{L}_c(c_2)$.

Criticity of composed context is assessed according to the resulting context category, as discussed in the previous section. We thus say that:

$$\mathcal{L}_c(c_1 \& c_2) = \text{Max}(\mathcal{L}_c(c_1), \mathcal{L}_c(c_2)) \text{ and } \mathcal{L}_c(\neg c) = ope.$$

Property 4 $c <_c c_1 \& c_2 \rightarrow (c <_c c_1 \vee c <_c c_2)$

Property 5 $c_1 \& c_2 <_c c \rightarrow (c_1 <_c c \wedge c_2 <_c c)$

Specificity. Specificity both deals with inheritance and context composition. Hierarchical specificity considers role, activity, view and context inheritances. It uses *sub_role*, *sub_activity*, *sub_view* and *sub_context* predicates to determine which rule is more specific. For instance, we say that r_2 is more specific than r_1 , a_2 is more specific than a_1 , v_2 is more specific than v_1 and c_2 is more specific than c_1 if: $sub_role(r_2, r_1)$, $sub_activity(a_2, a_1)$, $sub_view(v_2, v_1)$, $sub_context(c_2, c_1)$

Definition 10 We define the *specificity relation* for contexts as follows:

$$c_1 \leq_s c_2 \leftrightarrow sub_context(c_2, c_1)$$

$$c_1 <_s c_2 \leftrightarrow c_1 \leq_s c_2 \wedge \neg(c_1 = c_2)$$

¹Note in addition that even without this splitting technique, disjunctions may be expressed using combinations of negations and conjunctions, which also justifies the restriction to these two operators.

Regarding composition specificity, we say that:

$$c_1 \leq_s c_1 \& c_2 \wedge c_2 \leq_s c_1 \& c_2$$

Property 6 $c_1 \& c_2 \leq_s c \rightarrow (c_1 \leq_s c \wedge c_2 \leq_s c)$

Note. We do not have $c \leq_s c_1 \& c_2 \rightarrow (c \leq_s c_1 \vee c \leq_s c_2)$. For instance, let $c = a_1 \& a_2 \& a_3$ and $c_1 = a_1 \& a_2$ and $c_2 = a_3 \& a_4$. We have $c \leq_s c_1 \& c_2$, but we do not have $c \leq_s c_1$, nor $c \leq_s c_2$.

However, we assume that:

$$c \leq_s c_1 \& c_2 \rightarrow \exists c'_1, c'_2, c = c'_1 \& c'_2 \wedge c'_1 \leq_s c_1 \wedge c'_2 \leq_s c_2.$$

In the previous example, this is verified for $c'_1 = a_1 \& a_2$ and $c'_2 = a_3$.

Conflict resolution strategies. Now, to deal with criticality and specificity in our system, we define the two following strategies to assess rule priorities in case of potential conflicts. Rule types are denoted t_1 and t_2 , with $t_1, t_2 \in \{permission, prohibition\}$. We thus do not restrict to conflict resolution (cases where a rule is a permission and the other is a prohibition), but extend the approach to deal with redundancy. This means that if two matching rules recommend both a permission or both a prohibition, we assess which rule actually takes precedence, avoiding thus multiple instantiations of authorizations.

S1. *Criticality conflict resolution strategy*

$$\begin{aligned} &sr(t_1, org, r_1, a_1, v_1, c_1, p_1) \wedge \\ &sr(t_2, org, r_2, a_2, v_2, c_2, p_2) \wedge \\ &c_1 <_c c_2 \\ &\rightarrow p_1 \prec p_2 \end{aligned}$$

Note that the criticality strategy only considers context criticality to assess priorities. In fact, whatever the other parameters, the rule with the context of highest criticality always takes precedence.

S2. *Specificity conflict resolution strategy*

$$\begin{aligned} &sr(t_1, org, r_1, a_1, v_1, c_1, p_1) \wedge \\ &sr(t_2, org, r_2, a_2, v_2, c_2, p_2) \wedge \\ &sub_role(r_2, r_1) \wedge sub_activity(a_2, a_1) \wedge sub_view(v_2, v_1) \wedge c_2 \leq_s c_1 \wedge \\ ¬(r_1 = r_2 \wedge a_1 = a_2 \wedge v_1 = v_2 \wedge c_1 = c_2) \\ &\rightarrow p_1 \prec p_2 \end{aligned}$$

Note that *sub_role*, *sub_activity*, *sub_view* and *sub_context* are reflexive predicates. This means that we do not restrict to strict exceptions.

Now, we should prove that strategies *S1* and *S2* are not conflicting strategies, *i.e.* we shall not obtain conflicting decisions when applying these strategies.

Definition 11 *Strategies S_1 and S_2 are said to be conflicting iff:*

$$\exists sr_1, sr_2 \text{ security rules, } \mathcal{P}(sr_1)_{S_1} \prec \mathcal{P}(sr_2)_{S_1} \wedge \mathcal{P}(sr_1)_{S_2} \succ \mathcal{P}(sr_2)_{S_2}.$$

We define the two following constraints to deal with conflicting strategies:

Assumption 1 $\forall c_1, c_2$: atomic contexts, $c_1 <_c c_2 \wedge c_2 \leq_s c_1 \rightarrow error$

Assumption 2 $\forall c_1, c_2$: atomic contexts, $\neg c_1 <_c c_2 \wedge c_2 \leq_s \neg c_1 \rightarrow error$

Now, we should extend these assumptions to every well-formed context, as follows:

Lemma 1 $\forall c_1, c_2$: well-formed contexts, $c_1 <_c c_2 \wedge c_2 \leq_s c_1 \rightarrow error$

Proof. by recurrence on size of c_1 and c_2 .

Let c be an atomic context, then $Size(c) = Size(\neg c) = 1$.

$Size(c_1 \& c_2) = Max(Size(c_1), Size(c_2)) + 1$

1. Lemma 1 is verified if $Size(c_1) = Size(c_2) = 1$.

- *Case 1.* c_1 and c_2 atomic contexts: verified since Assumption 1,
- *Case 2.* $c_1 = \neg c'_1$ and c'_1, c_2 atomic contexts: verified since Assumption 2,
- *Case 3.* $c_2 = \neg c'_2$ and c'_2 atomic context: impossible to have $c_1 <_c \neg c'_2$ since $\mathcal{L}_c(\neg c'_2) = ope$.

2. Let us assume that Lemma 1 is verified if $Size(c_1) \leq n$ and $Size(c_2) \leq n$.

Now, let c_1 and c_2 contexts so that $Size(c_1) \leq n + 1$ and $Size(c_2) \leq n + 1$.

We then have $c_1 = c'_1 \& c'_2$ with $Size(c'_1) \leq n$ and $Size(c'_2) \leq n$.

Let us assume that $c_1 <_c c_2 \wedge c_2 \leq_s c_1$.

We then have $c'_1 \& c'_2 <_c c_2 \wedge c_2 \leq_s c'_1 \& c'_2$.

We choose c''_1 and c''_2 so that $c_2 = c''_1 \& c''_2 \wedge c''_1 \leq_s c'_1 \wedge c''_2 \leq_s c'_2$.

Note that $Size(c''_1) \leq n$ and $Size(c''_2) \leq n$.

We thus have:

$$c'_1 \& c'_2 <_c c''_1 \& c''_2 \wedge c''_1 \leq_s c'_1 \wedge c''_2 \leq_s c'_2$$

According to property 4, we deduce that:

$$(c'_1 \& c'_2 <_c c''_1 \vee c'_1 \& c'_2 <_c c''_2) \wedge c''_1 \leq_s c'_1 \wedge c''_2 \leq_s c'_2$$

Thus, thanks to property 5:

$$(c'_1 <_c c''_1 \wedge c'_2 <_c c''_1) \vee (c'_1 <_c c''_2 \wedge c'_2 <_c c''_2) \wedge c''_1 \leq_s c'_1 \wedge c''_2 \leq_s c'_2$$

- *Case 1.* $c'_1 <_c c''_1 \wedge c'_2 <_c c''_1 \wedge c''_1 \leq_s c'_1 \wedge c''_2 \leq_s c'_2$

We thus have:

$$c'_1 <_c c''_1 \wedge c''_1 \leq_s c'_1 \rightarrow error$$

- *Case 2.* $c'_1 <_c c''_2 \wedge c'_2 <_c c''_2 \wedge c''_1 \leq_s c'_1 \wedge c''_2 \leq_s c'_2$

We thus have:

$$c'_2 <_c c''_2 \wedge c''_2 \leq_s c'_2 \rightarrow error$$

In each case, we derive *error*, according to Assumption 1. Lemma 1 is thus proved by recurrence on size of c_1 and c_2 .

Theorem 1 *Strategies S_1 and S_2 are not conflicting strategies.*

Proof. Let us consider that strategies S_1 and S_2 are conflicting strategies. Thus there exists two security rules sr_1 and sr_2 such that $\mathcal{P}(sr_1)_{S_1} \prec \mathcal{P}(sr_2)_{S_1}$ and $\mathcal{P}(sr_2)_{S_2} \prec \mathcal{P}(sr_1)_{S_2}$.

The only way to derive $\mathcal{P}(sr_1)_{S_1} \prec \mathcal{P}(sr_2)_{S_1}$ is to apply strategy S_1 , as previously defined. We thus can conclude that $c_1 <_c c_2$, where c_1 and c_2 are the respective contexts of security rules sr_1 and sr_2 .

Similarly, the only way to derive $\mathcal{P}(sr_2)_{S_2} \prec \mathcal{P}(sr_1)_{S_2}$ is to apply strategy S_2 , as previously defined. We thus conclude that $c_2 \leq_s c_1$.

Applying Lemma 1, we derive *error*, which is a contradiction since we assume that constraints cannot be violated.

We thus prove Theorem 1.

4.4 Conclusion

In this chapter, we showed why and how Or-BAC can be used to enable automated threat response. We first explained the advantages of the Or-BAC model. It provides means to define authorizations which are dynamically activated considering current context; it provides a high-level of abstraction, allowing a global definition of the security policy, which is enforced locally; it ensures through conflict resolution at the abstract level that no conflict occurs at policy compilation. We proposed to consider alerts reported by intrusion detection systems as contextual data enabling threat context characterization. Contexts are structured in three categories, namely *operational* contexts, *threat* contexts and *minimal* contexts. Operational contexts define the traditional access control policy, whereas minimal contexts allow us to declare high priority contexts, ensuring the preservation of minimal requirements, whatever the characterized threat. We provided the possibility to combine contexts, through the use of a boolean algebra. The corresponding mechanisms to derive composed context priorities were given considering composing context categories. Finally, we showed how to manage conflict resolution and dynamic priority assessment through the use of two conflict resolution strategies based on *criticity* and *specificity* of rule parameters.

Chapter 5

Application: email use case

Contents

5.1	Introduction	67
5.2	Presentation of the email use case	68
5.3	Modeling roles	72
5.4	Modeling views	74
5.5	Modeling activities	76
5.6	Modeling contexts	79
5.6.1	Operational contexts	79
5.6.2	Threat contexts	80
5.6.3	Minimal contexts	81
5.7	Modeling policy	82
5.7.1	Operational policy	82
5.7.2	Reaction policy	83
5.7.3	Minimal policy	87
5.8	Modeling structured entities	88
5.9	Conclusion	92

5.1 Introduction

We present in this chapter a case study, whose aim is to illustrate how policy-based response can be achieved for the email service through multiple available paths to mail, including diversity of servers, services and client applications.

In a first section, we present the email use case. Then, we successively show in the four following sections how one can model roles, activities, views and contexts. For each of these four entities, we discuss model refinement possibilities. An important point to address deals with granularity at the abstract level. On the one hand, abstraction facilitates policy definition, avoiding the exhaustive listing of numerous security rules at the concrete level. On the other hand, fine-grained abstract entities allows the

expression of specific requirements in the security policy definition. It may render the definition of the policy more tedious, including more rules, which increases policy compilation complexity. A compromise has to be found between very specific abstract-level entities, expressing very fine-grained rules, and very generic abstract-level entities, minimizing the number of rules to define and compute.

We thus analyze how one can refine roles, activities, views, and also contexts, using the email case study for illustration purposes. This discussion should allow better understanding of the choices made in the email application, but also provide means to envision different applications, regarding considered requirements. A section then explains how one can define a sample email policy, declined over three sub-policies, namely the *operational* policy, the *reaction* policy and the *minimal* policy. Finally, a last section introduces the need for structuration of concrete entities, and gives the corresponding Prolog templates to correctly describe subjects, actions and objects.

5.2 Presentation of the email use case

Access to mail is provided to users through the use of three remote exchange servers. Each server is in charge of a third of the total number of mailboxes; they do not provide mailboxes redundancy, but simply static load balancing. Users can use three different client applications to deal with email, namely *outlook*, *thunderbird* and *firefox*, over four different mechanisms. The outlook mail client accesses the exchange servers through native Microsoft protocols. Thunderbird accesses POP and IMAP extensions of the same exchange servers. Finally, firefox accesses the OWA (Outlook Web Access) extension of the same exchange servers. In normal operation, all these four modes are active and allow parallel access to the same information, the consistency being preserved by the backend exchange servers.

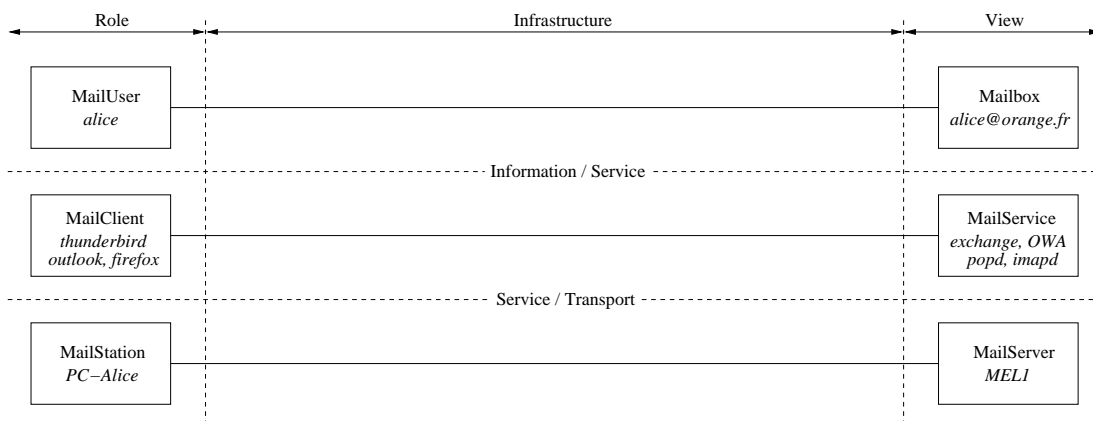


Figure 5.1: Email use case: simplified layer model

Figure 5.1 provides a description of the various entities involved in this use case. This description is layered to ease understanding of these entities. A request for access encapsulates information belonging to all three layers from top to bottom. Figure 5.2

is a simplified vision of a network datagram, which illustrates the successive encapsulations. IP addresses ensure communication between mail servers and mail stations. Hence, they are considered at the transport layer. The ports target specific applications on each host. Thus, they are used to identify data for the service layer. Finally, the data part of the packet, namely the TCP/UDP payload, contains commands of the service protocol, with arguments providing data for the information layer.

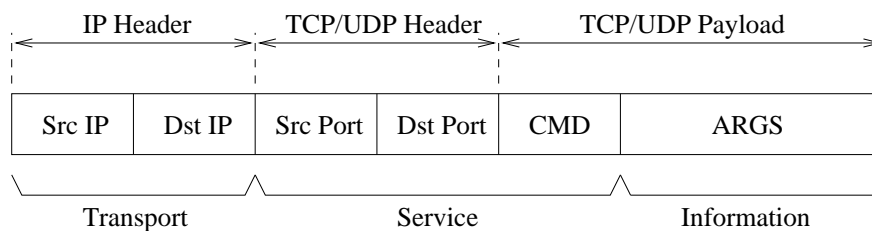


Figure 5.2: Simplified network datagram structure

Horizontal layer segmentation. The information layer at the top models interactions between users and information they are likely to access. In the use case, users want to access their mail messages. The middle layer represents the system intermediaries, typically programs, that make reading mail possible. In the use case, mail clients such as Microsoft OutlookTM, Mozilla Thunderbird or Mozilla Firefox, used as support for webmail access, interact with mail servers such as Microsoft ExchangeTM. The bottom layer represents the communication channels, enabling communication between machines; in the use case, this enables the exchange of TCP/IP packets between mail stations and mail servers.

Vertical segmentation. In addition to the layer segmentation, we bring up in figure 5.3 a vertical separation taking infrastructure into account. Indeed, in addition to the classic Subject/Object (and Role/View) duality, there are a large number of infrastructure components which are either necessary to ensure service access and thus communications, or which are to be traversed by traffic, especially those dealing with monitoring and security. Without these infrastructure functions, access is at best impaired and at worst impossible. Hence, they represent an attractive target for attackers and a possibility of countermeasure for the defender, and we wish to extend the classic policy model of subjects and objects by representing these components.

In the use case, user workstations rely on DNS to identify the target machine. User logins and access to information rely on Active Directory to identify and authenticate users, and associate user logins with mailboxes. They also need to traverse firewalls and intrusion detection / prevention systems. Note that it is not necessary to create new concepts to model these infrastructure entities; subjects and objects apply to them as well. Their presence in the model simply improves the understanding of the security policy and widens opportunities for threat response, since any combination of these

elements can be leveraged. In particular, we consider the Active Directory database as a PEP enabling access control on user accounts, and firewalls as PEPs allowing network / transport level (and possibly application-level) filtering.

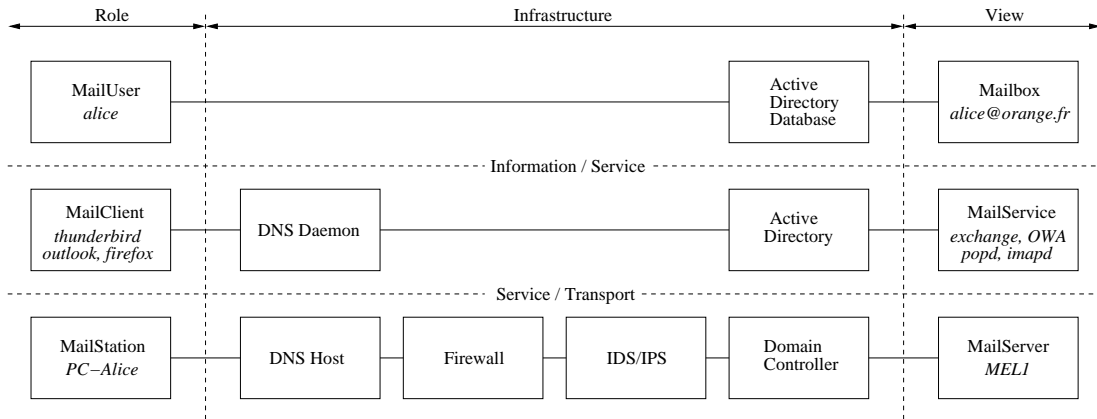


Figure 5.3: Email use case: layer model with infrastructure components

Note that the use case presented in figure 5.3 does not necessarily mention all potential infrastructure components. In particular, we do not represent here any potential router between client and server hosts, which may be used for instance to redirect traffic on response purpose. We also do not represent DHCP clients/servers, which are necessary for the hosts to obtain their network configuration at boot time and be able to further communicate and use services. This means that on a network where IP addresses are attributed dynamically through the use of DHCP, if the DHCP server is compromised, the only hosts able to communicate are those which were already connected and active on the network, and this for a limited time (DHCP lease). We assume here that service dealing with infrastructure components is correctly delivered. Hence, the presented policy only considers access control between entities at the extremities, namely those described as roles and views in figure 5.3, *i.e.* mail users and mailboxes, mail clients and mail services, and mail stations and mail servers.

Figure 5.4 recaps the email use case architecture with the different paths for the mail activity. Protocols are depicted as diamonds, client software as rounded boxes and server software as circles. Note that figure 5.4 provides a logical representation of the functionalities (web service, exchange service and SMTP service). These functionalities may be grouped on a same component (server). However, we consider the same representation at the physical level, that is one component (server) is deployed for each functionality (service), since separation of the functionalities allows additional potential actions. For instance, one can choose to isolate the web server in response to a threat, without having any side-effect on the two other servers, except the fact that OWA is no more accessible. We mention here only one Exchange server, since interactions of the three Exchange services are similar to the OWA web server and the SMTP server.

1. **Protocols.** Required and available protocols for the mail use case include reading and writing email through the use of HTTP for Outlook Web Access (OWA), Netbios¹ for native Microsoft Exchange, POP and IMAP (reading only), and SMTP (writing only).
2. **Server software.** Server software associated with these protocols regroup three main servers: a web server to access OWA, a SMTP server for mail writing, and the exchange server which deals with mailboxes repository, native Exchange, and POP and IMAP through Exchange connectors (POPC and IMAPC), implemented as services running conjointly to the Exchange service.
3. **Client software.** As mentioned previously, we consider at least three mail clients in the use case, which are in fact three kinds of mail clients: those allowing OWA (*i.e.* web browsers, *e.g.* Firefox), those allowing native Exchange mailing (namely Outlook), and those allowing POP, IMAP or native SMTP (*e.g.* Thunderbird).

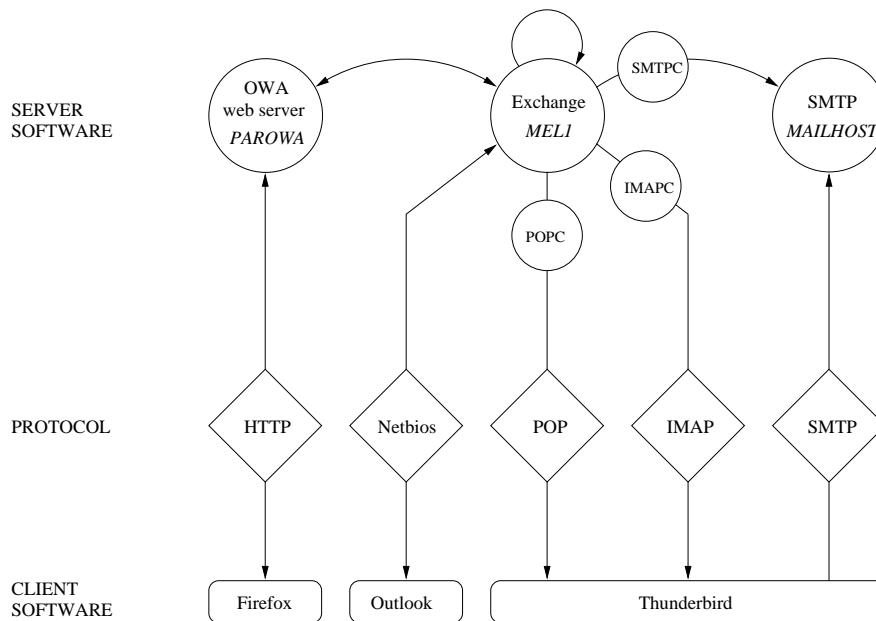


Figure 5.4: Email use case: logical architecture

Note that writing mail to internal email addresses is represented as a loop on figure 5.4, since Exchange manages this kind of mail internally. However, sending a mail written to an external email address is realized through the use of the dedicated SMTP

¹Behind the keyword “Netbios”, we mean the four corresponding protocols used by Microsoft for client / server communications, namely *loc-srv*, *netbios-ns*, *netbios-dgm* and *netbios-ssn*. They are respectively used for service localization for Remote Procedure Calls (RPC), name service, datagram service and session service. This explains why we consider in the following four TCP ports to filter native Exchange access (TCP/135, TCP/137, TCP/138 and TCP/139).

server. Interface with the Exchange server is simplified in the figure to a SMTP connector (SMTPC). However, one should notice that it also requires the presence of a Microsoft SMTP virtual server to relay SMTP traffic, which we consider here to be installed on the same host than the Exchange server.

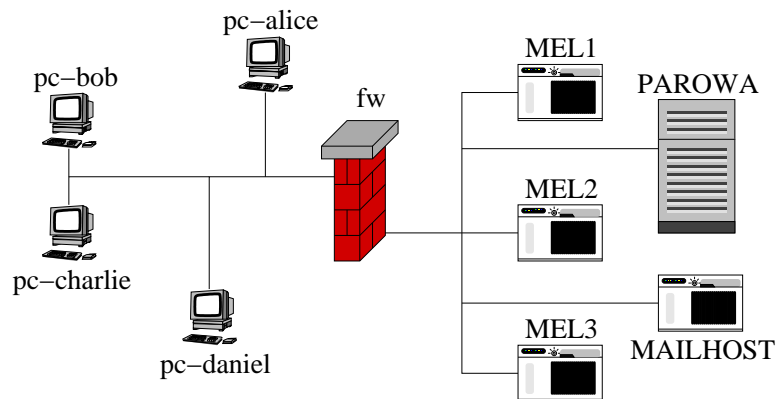


Figure 5.5: Email use case: physical architecture

Note also that alike infrastructure dependencies, *e.g.* DNS, figure 5.4 clearly introduces additional dependencies in the email use case. Indeed, one can notice that the web server for OWA is in fact a client of the Exchange service. Similarly, the Exchange service uses the SMTP service in order to send mail. Minimal requirements thus have to be defined to ensure that response does not provide unwilling side-effects, such as stopping the Exchange process. Indeed, this would automatically prevent users from reading mail². Similarly, shutting down the SMTP service would mean that it is no more possible to send mail. Note however a distinction between sending mail and writing mail. Indeed, even in case of SMTP service ceasing, it is still possible to write mail offline on the Exchange server. Then, once the SMTP service is reestablished, mail delivery is ensured as delayed sending of the stored email. Moreover, loosing the dedicated SMTP service does not prevent users from writing and sending internal mail.

5.3 Modeling roles

Regarding the presented architecture, we describe roles in figure 5.6, segmented according to the three layers of figure 5.1. Note that while this segmentation is not absolutely necessary - we could attach the various components that are in the layers to the *Mail-Role* nodes -, it provides additional segmentation that will prove useful for defining and analyzing countermeasures. Accordingly, the various concrete objects (*i.e.* subjects) are attached to the appropriate abstract nodes.

²Note however that even if all reading paths to the remote mail service were blocked, it would still be possible to read mails which would be stored locally on the client hosts. Although we will not consider this in our minimal requirements, another policy may take it into account, considering thus a minimal policy where users have at least the possibility to read their local mail repository. Such a policy should

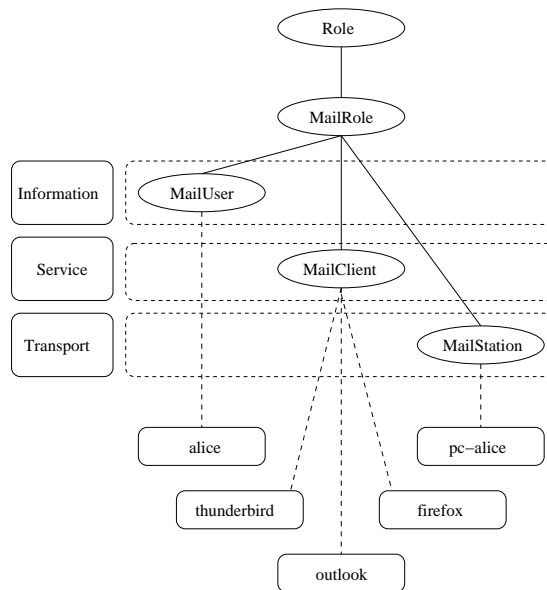


Figure 5.6: Email use case: roles modeling

The illustrated hierarchy of roles is expressed by the corresponding Or-BAC Prolog facts in listing 5.1. The *role* statements allow the definition of roles. The *empower* statements are used for subject to role assignments. Finally, hierarchy is obtained through the use of *sub_role* statements.

Listing 5.1: Definition of roles

```

role(pie, 'MailRole').
role(pie, 'MailUser').
role(pie, 'MailClient').
role(pie, 'MailStation').
sub_role(pie, 'MailUser', 'MailRole').
sub_role(pie, 'MailClient', 'MailRole').
sub_role(pie, 'MailStation', 'MailRole').

empower(pie, 'alice', 'MailUser').
empower(pie, 'thunderbird', 'MailClient').
empower(pie, 'outlook', 'MailClient').
empower(pie, 'firefox', 'MailClient').
empower(pie, 'PC-Alice', 'MailStation').
  
```

Note that listing 5.1 is biased; in any realistic deployment, there should be many more concrete entities than abstract entities. The ratio of one to one in the listing is not representative of a realistic setting. In particular, for the sake of simplicity, we only take into account a single user, since it is enough to illustrate the principle.

About role granularity. We present here a very simple arborescence of role modeling, granularity being only provided considering the three layers, namely transport, service and information. Let us now discuss possible role refinements.

obviously ensure that mails are effectively downloaded and stored locally by mail stations.

- **MailUser.** Refinement of the *MailUser* role may be envisioned regarding service. In particular, one could distinguish *PopUser*, *ImapUser*, *OWAUser*, etc. We consider that this does not make sense in our case study, since users are already grouped in the mail activity and should all be able to access the desired service in the operational context. However, in a more realistic use case, it is possible that mail users would be divided according to different minimal requirements considering for instance their hierarchy or their function in the corporation.
- **MailClient.** Refinement of the *MailClient* role is used to distinguish the capabilities of client applications. For instance, Mozilla Thunderbird would be considered as a *PopClient*, *ImapClient*, and *SmtplibClient*, but not a *OWAClient*. We do not consider this refinement to avoid confusion.
- **MailStation.** Refinement of the *MailStation* role may also be considered according to service, distinguishing roles like *PopStation* and *ImapStation*. Alike users, we make the choice to avoid such a refinement, since stations are supposed to permit every proposed service.

5.4 Modeling views

Roles and views have symmetric behavior respectively of subjects and objects, with one little difference. Although we do not feel the need for additional segmentation concerning the role / subject hierarchy, we provide refinement at the *MailService* node level with respect to the various services used. This is useful for better definition of security rules, ensuring better consistency and fine-grained rules. Similarly, we refine the *MailServer* node to distinguish OWA web servers, Exchange servers and SMTP servers, which does not assume the same functionalities.

Listing 5.2: Definition of views

```

view(pie, 'MailView').
view(pie, 'Mailbox').
view(pie, 'MailService')
view(pie, 'MailServer')
sub_view(pie, 'Mailbox', 'MailView').
sub_view(pie, 'MailService', 'MailView').
sub_view(pie, 'MailServer', 'MailView').
sub_view(pie, 'OWAService', 'MailService').
sub_view(pie, 'POPService', 'MailService').
sub_view(pie, 'ExchgService', 'MailService').
sub_view(pie, 'IMAPService', 'MailService').
sub_view(pie, 'SMTPService', 'MailService').
sub_view(pie, 'OWAServer', 'MailServer').
sub_view(pie, 'ExchgServer', 'MailServer').
sub_view(pie, 'SMTPServer', 'MailServer').

use(pie, 'alice@orange.fr', 'Mailbox').
use(pie, 'IIS6.0', 'OWAService').
use(pie, 'ExchPOP3', 'POPService').
use(pie, 'Exchange2003', 'ExchgService').
use(pie, 'ExchIMAP4', 'IMAPService').
use(pie, 'ESMTP6.0', 'SMTPService').
use(pie, 'PAROWA', 'OWAServer').
use(pie, 'MEL1', 'ExchgServer').
use(pie, 'MEL2', 'ExchgServer').
use(pie, 'MEL3', 'ExchgServer').
use(pie, 'MAILHOST', 'SMTPServer').

```

The hierarchy of views shown in figure 5.7 is expressed by the corresponding Or-BAC Prolog facts in listing 5.2. The *view* statements allow the definition of views. The

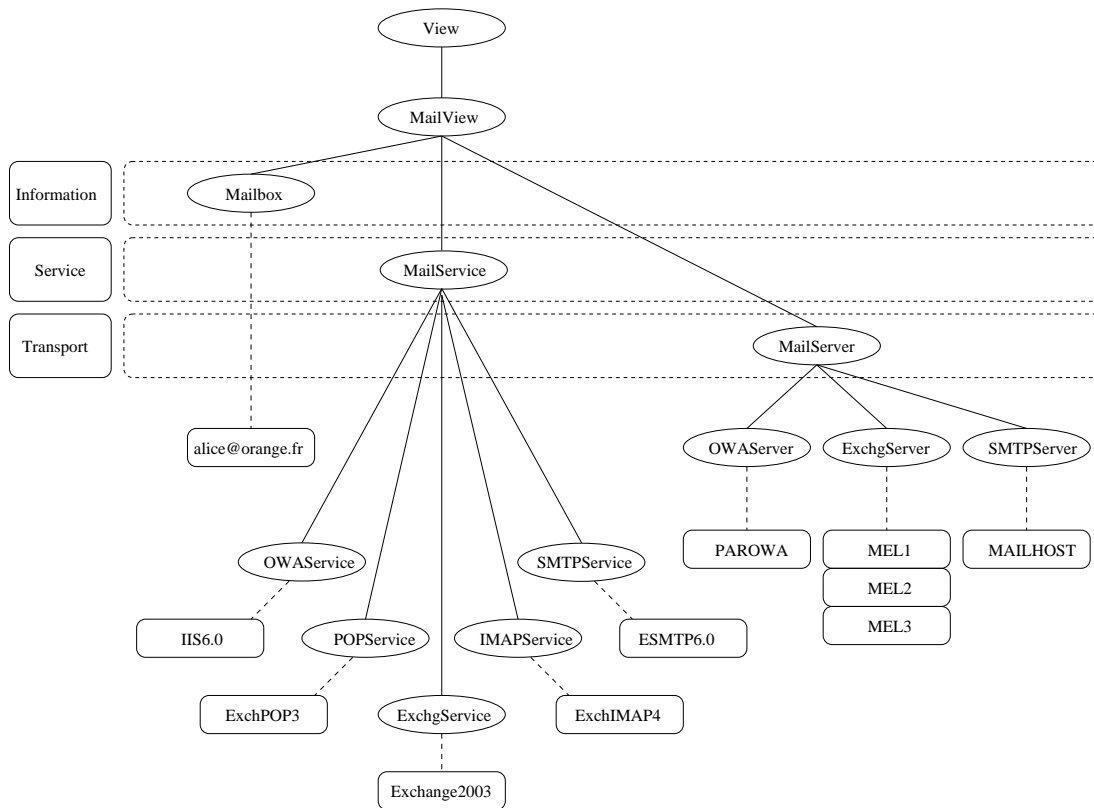


Figure 5.7: Email use case: views modeling

use statements are used for object to view assignments. Finally, hierarchy is actually obtained through the use of *sub_view* statements. Note that alike roles, the listing is biased, showing only one mailbox, for the sake of simplicity.

About view granularity. Refinements regarding views may be interpreted as follows:

- **Mailbox.** Refinement is possible mainly if the policy takes into account access control regarding mailing lists, or more generally lists of mailboxes. In such a case, if some policies have to consider for instance the hierarchy or function of users in the corporation, this may be envisioned. In our case study, we keep it simple, avoiding such distinctions.
- **MailService.** The *MailService* view can be refined according to the different services, namely *PopService*, *ImapService*, *SmtplibService*, etc. This is clearly one of the most important entity to refine, since it may permit the definition of fine-grained security rules, controlling access at the service level. We thus choose to implement this refinement in the email use case.

- MailServer.** The *MailServer* view is also likely to need refinement, to consider the different physical entities composing an email environment. In our case study, there are multiple servers depending on the proposed services (OWA, Exchange / POP / IMAP and SMTP). Given a threat, one may for instance choose to react on Exchange servers, but not on OWA and SMTP servers. It is thus important to provide this possibility through the necessary refinement at the abstract level. This is also necessary because there may be dependencies between these views, as illustrated by OWA and Exchange.

5.5 Modeling activities

Activities are described in figure 5.8. We have segmented mail under three activities: access to the mailbox (login), mail reading and mail writing. This segmentation is introduced with respect to the response system, where options such as preventing new sessions but letting existing sessions continue, or letting users read but not write mail, are opening up additional opportunities to focus for the response system and limit the response with respect to threat.

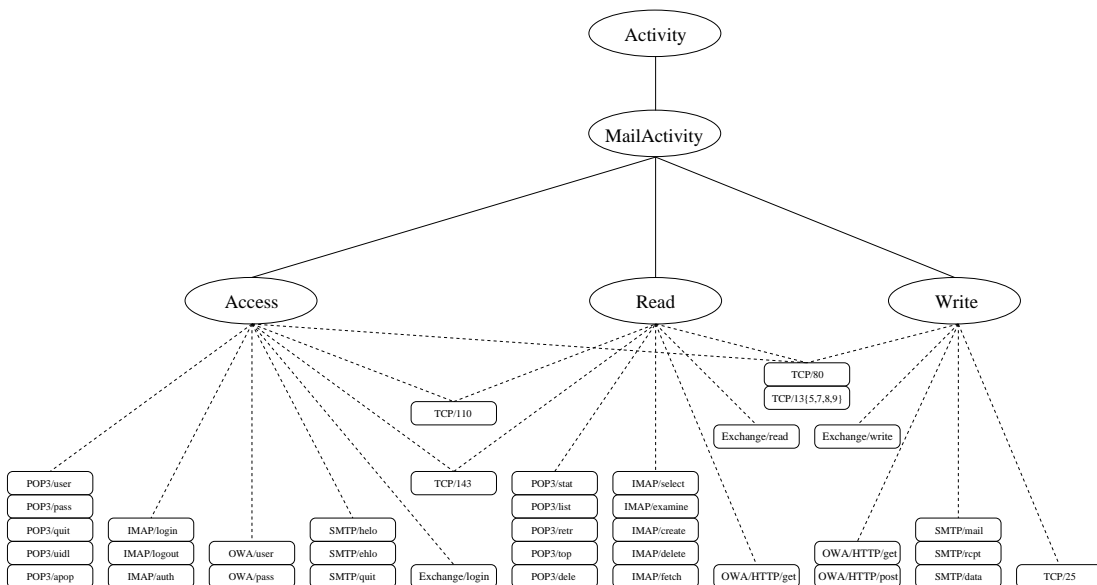


Figure 5.8: Email use case: activities modeling

Actions are modeled at the information level by retaining the various application commands used by the protocols for communications between the mail client and the mail server, and at the service and transport layer by the protocols and ports involved. Since protocols and ports are both found in IP packets and on machines (as bound ports), it is sufficient to model them with a single concrete object. For convenience, we are using a regular-expression-like notation for the Microsoft protocols, since they need to be configured together to enable the activity to succeed. Protocol commands for

POP, IMAP and SMTP are taken from Request for Comments (RFC) 1939, 2060 and 2821 respectively, although we have only introduced here a subset of these commands. Commands for OWA are limited to HTTP GET and POST commands, since OWA is web-based. Since the Exchange/Outlook dialog is not a public standard, we have introduced meta-keywords to represent commands that correspond to the various mail-related activities of Exchange.

In figure 5.8, we also acknowledge that this granularity does not apply to all possible activities. For example, the direct connection between the Microsoft Outlook mail reader and the Microsoft Exchange mail server limits our capability to separate read and write activities at the network layer, because all three sub-activities use the same set of ports. This is the same for the separation between access and read at the network layer. The access activity, however, is separable at the service layer, by manipulating the active directory server to prevent exchange login requests to succeed, or by filtering out IMAP and POP3 login requests, and thus forbid access.

Activities are described in listing 5.3 with respect to figure 5.8.

Listing 5.3: Definition of activities

```

activity (pie, 'MailActivity').
activity (pie, 'Access').
activity (pie, 'Read').
activity (pie, 'Write').
sub_activity (pie, 'Access', 'MailActivity').
sub_activity (pie, 'Read', 'MailActivity').
sub_activity (pie, 'Write', 'MailActivity').

consider (pie, 'TCP/80', 'Access').
consider (pie, 'TCP/110', 'Access').
consider (pie, 'TCP/13{5,7,8,9}', 'Access').
consider (pie, 'TCP/143', 'Access').
consider (pie, 'TCP/80', 'Read').
consider (pie, 'TCP/110', 'Read').
consider (pie, 'TCP/13{5,7,8,9}', 'Read').
consider (pie, 'TCP/143', 'Read').
consider (pie, 'TCP/25', 'Write').
consider (pie, 'TCP/80', 'Write').
consider (pie, 'TCP/13{5,7,8,9}', 'Write').
consider (pie, 'Exchange/login', 'Access').
consider (pie, 'POP3/user', 'Access').
consider (pie, 'POP3/pass', 'Access').
consider (pie, 'POP3/quit', 'Access').
consider (pie, 'POP3/uidl', 'Access').
consider (pie, 'POP3/apop', 'Access').
consider (pie, 'IMAP/login', 'Access').
consider (pie, 'IMAP/logout', 'Access').
consider (pie, 'IMAP/auth', 'Access').
consider (pie, 'SMTP/helo', 'Access').
consider (pie, 'SMTP/ehlo', 'Access').
consider (pie, 'SMTP/quit', 'Access').
consider (pie, 'OWA/user', 'Access').
consider (pie, 'OWA/pass', 'Access').
consider (pie, 'Exchange/receive', 'Read').
consider (pie, 'POP3/stat', 'Read').
consider (pie, 'POP3/list', 'Read').
consider (pie, 'POP3/retr', 'Read').
consider (pie, 'POP3/top', 'Read').
consider (pie, 'POP3/dele', 'Read').
consider (pie, 'IMAP/select', 'Read').
consider (pie, 'IMAP/examine', 'Read').
consider (pie, 'IMAP/create', 'Read').
consider (pie, 'IMAP/delete', 'Read').
consider (pie, 'IMAP/fetch', 'Read').
consider (pie, 'OWA/HTTP/get', 'Read').
consider (pie, 'Exchange/send', 'Write').
consider (pie, 'SMTP/mail', 'Write').
consider (pie, 'SMTP/rcpt', 'Write').
consider (pie, 'SMTP/data', 'Write').
consider (pie, 'OWA/HTTP/get', 'Write').
consider (pie, 'OWA/HTTP/post', 'Write').

```

The normal behavior of certain protocols puts demands on the model. For in-

stance, ports $TCP/13\{5, 7, 8, 9\}$ are grouped since all activities related to the Outlook-Exchange connections cannot be differentiated easily at the network level. However, the service level does differentiate access, read and write activities.

Figure 5.9 depicts how activities are achieved in the use case, activities *Access*, *Read* and *Write* being respectively represented by letters *A*, *R* and *W*. The only activities mentioned on the figure are port numbers, allowing to control access from clients to services at the network / transport level. Higher-level actions, like *POP3/user* or *SMTP/helo* are not represented on the figure. However, they may be used as well to enforce filtering through the use of application-level firewalls (enabling to filter specific commands), or possibly located close to services, dealing with service reconfigurations to permit or not the use of such commands given the context. Server-side service-level objects are mentioned in italics in the corresponding circles on the figure. In our use case, the OWA web server, the Exchange server and the SMTP server respectively run Microsoft IIS 6.0, Microsoft Exchange Server 2003 6.5, and Microsoft ESMTP MAIL Service 6.0. In addition, POPC and IMAPC connectors are the POP3 server and the IMAP4rev1 server extensions of Microsoft Exchange 2003.

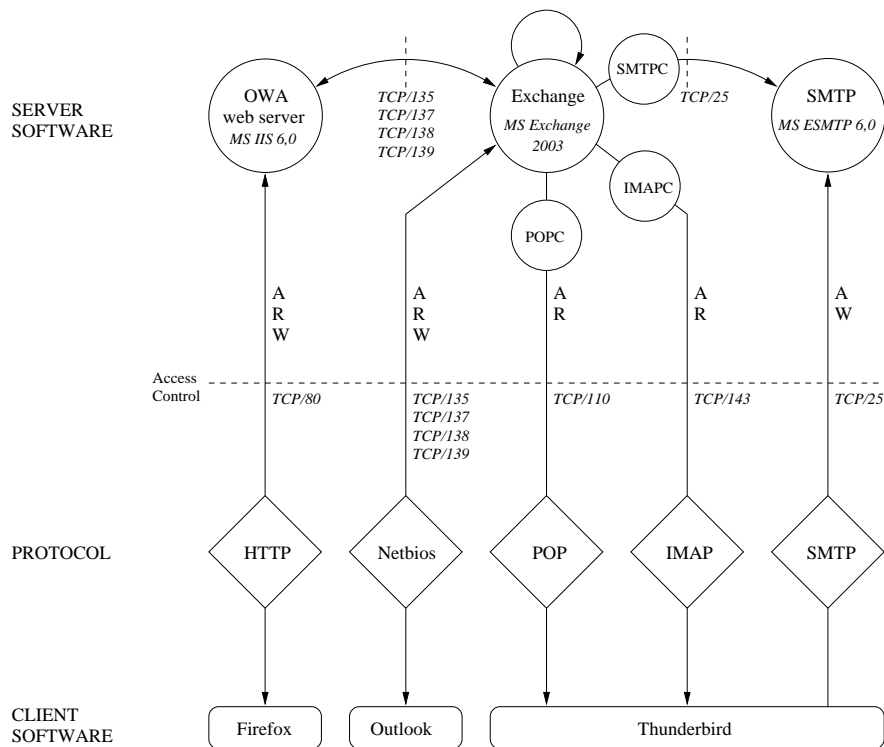


Figure 5.9: Email use case: logical architecture, actions and activities

About activity granularity. Basic activities, namely *Access*, *Read* and *Write*, can be refined according to services, distinguishing *AccessPop* from *AccessImap* for instance, but it is redundant with the refinement of the views. It is clear that,

regarding enforcement, couple $(Access, PopService)$ will be interpreted similarly as $(AccessPop, PopService)$, since the information about the service to consider is given in the view in the former. In the model, the latter is more rigorous since it describes precisely which actions are considered. We come back later on this refinement, which appears to be useful at the strategy level. However, we do not mention refinement on figure 5.8, for the sake of simplicity, and since actions are already visually grouped by service sub-activity.

5.6 Modeling contexts

Context are hierarchically defined as presented in figure 5.10.

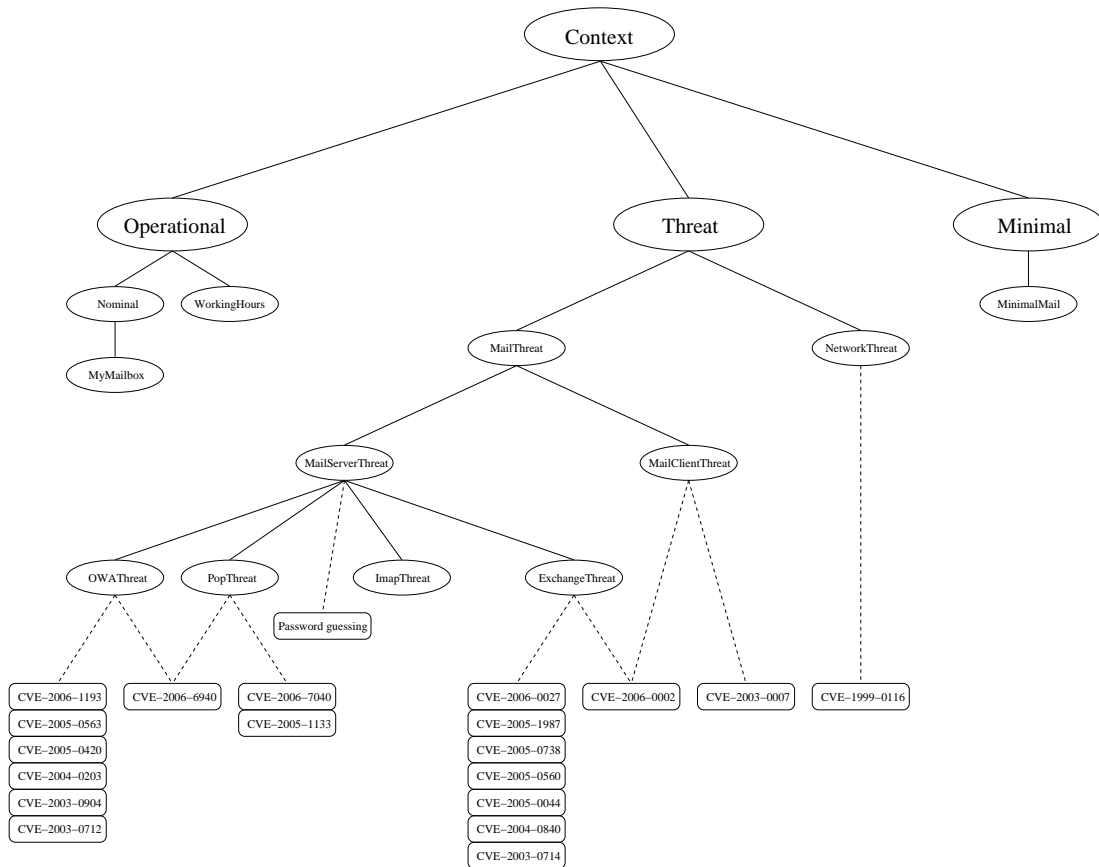


Figure 5.10: Email use case: contexts modeling

5.6.1 Operational contexts

For the normal operation of our system, we use *operational* contexts only, namely the *Nominal* context, which interest is to specify default requirements (default permit),

and the *MyMailbox* context, whose objective is to ensure that the user can only access his mailbox. The definition of the *MyMailbox* context is necessary to define a policy exception to the *Nominal* context, because access to the mailbox relies on access to the information system, and there is a link between the authentication data (login, password) and the mailbox. The *sub_context* relation in listing 5.4 implies that the *MyMailbox* context takes precedence by mean of specificity over the *Nominal* context.

We also define a temporal context, namely *WorkingHours*, to specify minimal requirements. The root *Operational* context is defined mainly for priority assessment, especially dealing with criticality. There is obviously no reason to manage security rules defining the global *Operational* context. However, evaluating criticality requires the knowledge of the category of context, which is obtained thanks to the root in the hierarchical graph.

Listing 5.4: Email operational contexts

```
context (pie, 'Operational').
context (pie, 'WorkingHours').
context (pie, 'Nominal').
context (pie, 'MyMailbox').
sub_context (pie, 'WorkingHours', 'Operational').
sub_context (pie, 'Nominal', 'Operational').
sub_context (pie, 'MyMailbox', 'Nominal').
```

5.6.2 Threat contexts

Threat contexts are triggered by alerts, considering the vulnerability references reported in the IDMEF messages. References and their attributes are found in a vulnerability database, like the National Vulnerability Database³ (NVD). We define in our case study a global *MailThreat* context, including *MailServerThreat* and *MailClientThreat* contexts. Regarding server threats, we refine contexts to obtain four sub-contexts with respect to the four different mechanisms of the use case, namely *OWAThreat*, *PopThreat*, *ImapThreat* and *ExchangeThreat*. Listing 5.5 shows the complete definition of threat contexts in figure 5.10.

Listing 5.5: Definition of threat contexts

```
context (pie, 'Threat').
context (pie, 'MailThreat').
context (pie, 'MailServerThreat').
context (pie, 'MailClientThreat').
context (pie, 'OWAThreat').
context (pie, 'PopThreat').
context (pie, 'ImapThreat').
context (pie, 'ExchangeThreat').
context (pie, 'NetworkThreat').
sub_context (pie, 'MailThreat', 'Threat').
sub_context (pie, 'NetworkThreat', 'Threat').
sub_context (pie, 'MailServerThreat', 'MailThreat').
sub_context (pie, 'MailClientThreat', 'MailThreat').
sub_context (pie, 'OWAThreat', 'MailServerThreat').
sub_context (pie, 'PopThreat', 'MailServerThreat').
sub_context (pie, 'ImapThreat', 'MailServerThreat').
sub_context (pie, 'ExchangeThreat', 'MailServerThreat').
```

Note that listing 5.5 also includes a *NetworkThreat* context, since network threats may affect any service, including the mail service. Such a context may thus be useful in the definition of the email policy.

³<http://nvd.nist.gov>

About threat context granularity. Defining fine-grained rules regarding roles, activities and views is important to provide a relevant policy, but since rules are activated through contexts, context refinement is also crucial to better qualify threat, and thus to trigger most adequate response. There are at least three adjustment variables allowing to refine threat contexts, as shown in the following:

1. **Service.** Threats are classified by service (or mechanism) which is targeted by the considered attack (*pop*, *imap*, *exchange* or *owa*). This is the approach we consider in the email case study. References are thus classified among as many categories as there are services, and possibly others, dealing with non-service-level threats, like synflooding attacks.
2. **Type of attack.** Threats can also be classified considering the type of attack (*admin*, *dos*, *recon*, etc.). The type of attack is reported in the IDMEF Assessment class, but for a consistent and exhaustive list, one should consider the creation of a classification given references.
3. **Severity of alert.** Threats can finally be treated differently given the severity of the alert (*info*, *low*, *medium* or *high*). Severity can help achieving the most adequate choice regarding scale and level of response. A high severity may orient the choice to a large response, taking into account other potentially threatened hosts or services. Low-level response, like complete blocking of service access, may also be preferable for a high severity alert. On the opposite, low severity alerts should imply very specific countermeasures, limiting the risk of undesirable side-effects.

Context composition may achieve the necessary refinement considering these three adjustment variables. For instance, composed contexts like $\&(pop_attack, recon)$ or $\&(\&(pop_attack, recon), medium)$ better qualify the threat than elementary context *pop_attack*. This may prove to be useful to express more fine-grained policy rules. We do not consider such refinements in our email use case. We limit ourselves to the service parameter to characterize threats. The main part of response strategy relies at the concrete level. Parameters like type of attacks and severity of alerts may thus as well be considered at the concrete level to activate adequate *hold* facts with respect to threat.

5.6.3 Minimal contexts

Minimal requirements are specified through *minimal* contexts, associated with the highest context priority (*i.e.* the highest criticality), in order to satisfy requirements which must not be overridden. In the email use case, a minimal requirement is that at least one path to mail should be preserved whatever the current threat(s). This is defined through the *MinimalMail* context in figure 5.10 according to listing 5.6.

Listing 5.6: Definition of minimal context

```
context(pie, 'Minimal').
context(pie, 'MinimalMail').
sub_context(pie, 'MinimalMail', 'Minimal').
```

The minimal contexts are clearly part of the organizational policy, *i.e.* they are not directly linked with security. They consist in requirements which are specific to the organization, and often related to performance, convenience, or business constraints, especially regarding availability of services and resources.

5.7 Modeling policy

Given the presented models for roles, activities, views and contexts, we now show how one can define a simple email policy taking threat response and minimal requirements into account. We provide in this section means to define the corresponding three sub-policies (*operational*, *reaction* and *minimal* policies).

5.7.1 Operational policy

We provide here two sample solutions to define an email operational policy. First, we present the closed policy approach, and secondly, we provide the open policy approach.

5.7.1.1 Closed email policy

Within the presented model, the operational policy is expressed by four simple rules, given in listing 5.7. This closed policy (“default deny”) prohibits the mail activity to users through the *Nominal* context (first security rule), and permits explicitly the mail activity provided the user is accessing his own mailbox (*MyMailbox* context). Since there is no additional constraint at lower levels, that is any mail client should be allowed to access any mail service, and any mailstation should be allowed to access any mailserver in the operational policy, we define the second and third security rules. A potential conflict occurs between the first and the last security rules, and the last one takes precedence by means of specificity of the *MyMailbox* context over the *Nominal* context.

The *Nominal* context is always active (default policy), according to the first *hold* statement. The second *hold* statement ensures that the subject (*MailUser*) is actually linked with the object (*Mailbox*) in Active Directory, to ensure that users are allowed to access their mailbox only.

Listing 5.7: Sample operational policy 1 for the email use case (closed policy)

```
security_rule(prohibition , pie , 'MailUser' , 'MailActivity' , 'Mailbox' , 'Nominal' ).
security_rule(permission , pie , 'MailClient' , 'MailActivity' , 'MailService' , 'Nominal' ).
security_rule(permission , pie , 'MailStation' , 'MailActivity' , 'MailServer' , 'Nominal' ).
security_rule(permission , pie , 'MailUser' , 'MailActivity' , 'Mailbox' , 'MyMailbox' ).

hold(pie , __ , __ , __ , 'Nominal' ).

hold(pie , Subject , Action , Object , 'MyMailbox' ) :-
    empower(pie , Subject , 'MailUser' ),
    consider(pie , Action , 'MailActivity' ),
    use(pie , Object , 'Mailbox' ),
    active_directory_entry(Subject , Object ).
```

5.7.1.2 Open email policy

We provide in listing 5.8 another way to define the aforementioned sample operational policy. We describe here an open policy, *i.e.* “default permit”. The *Nominal* context defines three permissions according to the three layers, whereas the *MyMailbox* context defines an exception (prohibition), *i.e.* a more specific rule regarding role (*MailUser*) and view (*Mailbox*). Note the use of context negation to prohibit access to mailboxes of other users. The $n(MyMailbox)$ context has the same specificity as the *MyMailbox* context, and thus takes precedence over the *Nominal* context.

Note also the definition of the last *hold* statement in the listing, which is the predicate enabling evaluation of negative context definitions.

Listing 5.8: Sample operational policy 2 for the email use case (open policy)

```
security_rule(permission, pie, 'MailUser', 'MailActivity', 'Mailbox', 'Nominal').
security_rule(permission, pie, 'MailClient', 'MailActivity', 'MailService', 'Nominal').
security_rule(permission, pie, 'MailStation', 'MailActivity', 'MailServer', 'Nominal').
security_rule(prohibition, pie, 'MailUser', 'MailActivity', 'Mailbox', n(MyMailbox)).

hold(pie, __, __, __, 'Nominal').

hold(pie, Subject, Action, Object, 'MyMailbox') :-
    empower(pie, Subject, 'MailUser'),
    consider(pie, Action, 'MailActivity'),
    use(pie, Object, 'Mailbox'),
    active_directory_entry(Subject, Object).

hold(Org, Subject, Action, Object, n(Context)) :-
    not(hold(Org, Subject, Action, Object, Context)).
```

5.7.2 Reaction policy

5.7.2.1 Defining an email reaction policy

The *reaction* policy deals with new policy instances to enforce when threats are characterized (activation of *hold* facts depending on incoming alerts). In this section, we explain how we define a reaction policy with respect to the email application. We then give examples of threat-related *hold* facts, used for actual policy instances derivation in case of characterized threats.

We give in listing 5.9 the basic email reaction policy. It expresses that in case of threat towards mail, prohibitions have to be instantiated with respect to the threat.

Listing 5.9: Reaction basic policy for the email use case

```
security_rule(prohibition, pie, 'MailRole', 'MailActivity', 'MailView', 'MailThreat').
```

However, listing 5.9 only expresses the global response concept, but it is too generic to provide correct policy instantiation and enforcement. We thus make use of the defined model to refine the reaction policy. The first step is to refine the generic rule according to services. This affects activity and context, as shown by listing 5.10. With such a refinement, we avoid inconsistencies like the definition of a rule dealing with the *Read* activity for the *SmtplibService*. This is clearly a non-sense, since SMTP is a protocol which allows mail writing, but not reading.

Listing 5.10: Refining the basic reaction policy

```

security_rule(prohib, pie, 'MailRole', 'AccessOWA', 'MailView', 'OWAThreat').
security_rule(prohib, pie, 'MailRole', 'ReadOWA', 'MailView', 'OWAThreat').
security_rule(prohib, pie, 'MailRole', 'WriteOWA', 'MailView', 'OWAThreat').

security_rule(prohib, pie, 'MailRole', 'AccessPop', 'MailView', 'PopThreat').
security_rule(prohib, pie, 'MailRole', 'ReadPop', 'MailView', 'PopThreat').

security_rule(prohib, pie, 'MailRole', 'AccessImap', 'MailView', 'ImapThreat').
security_rule(prohib, pie, 'MailRole', 'ReadImap', 'MailView', 'ImapThreat').

security_rule(prohib, pie, 'MailRole', 'AccessExchange', 'MailView', 'ExchangeThreat').
security_rule(prohib, pie, 'MailRole', 'ReadExchange', 'MailView', 'ExchangeThreat').
security_rule(prohib, pie, 'MailRole', 'WriteExchange', 'MailView', 'ExchangeThreat').

security_rule(prohib, pie, 'MailRole', 'AccessSmtpt', 'MailView', 'SmtptThreat').
security_rule(prohib, pie, 'MailRole', 'WriteSmtpt', 'MailView', 'SmtptThreat').

```

Now, let us consider figure 5.11. Intra-level rules, *i.e.* rules associated with roles, activities and views of a same layer, clearly make sense regarding enforcement. However, it is unclear whether inter-level rules, *i.e.* rules which mix different levels, are relevant to our needs. In our application, we want to ensure that the mail user is allowed to access his mailbox, that the used mail client is allowed to access the considered mail service, and that the used mail station is allowed to access the considered mail server.

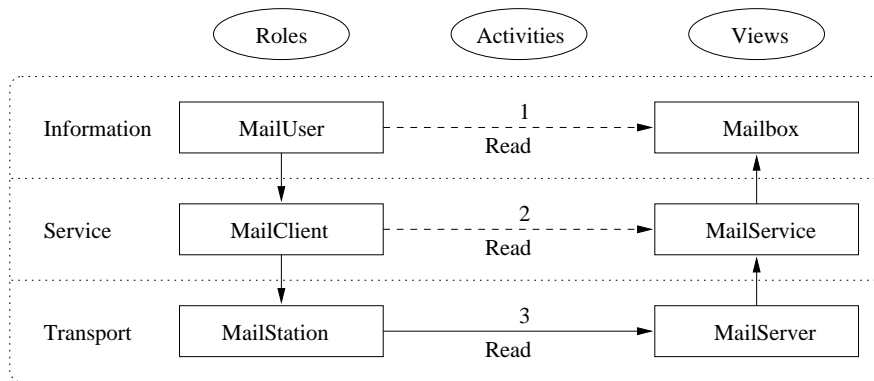


Figure 5.11: Email use case: layer model for the Read activity

Similarly as for operational policy definition, we should thus refine *MailRole* and *MailView* according to the three layers, obtaining thus listing 5.11 for the POP3 service. With such refined abstract rules, we ensure that we will avoid inconsistent policy instances at the concrete level, associating for instance an IMAP action with a POP3 service.

Listing 5.11: Refining even more the reaction policy

```

security_rule(prohib, pie, 'MailUser', 'AccessPop', 'Mailbox', 'PopThreat').
security_rule(prohib, pie, 'PopClient', 'AccessPop', 'PopService', 'PopThreat').
security_rule(prohib, pie, 'MailStation', 'AccessPop', 'PopServer', 'PopThreat').

security_rule(prohib, pie, 'MailUser', 'ReadPop', 'Mailbox', 'PopThreat').
security_rule(prohib, pie, 'PopClient', 'ReadPop', 'PopService', 'PopThreat').
security_rule(prohib, pie, 'MailStation', 'ReadPop', 'PopServer', 'PopThreat').

```

5.7.2.2 Discussion

Regarding presented policies, a question however arises about the necessity of refining abstract rules since it is not to be instantiated since there does not exist any matching *hold* fact. Let us for instance consider listing 5.12. The policy expressed here deals with application-level firewall rules application, enforced through the filtering of application commands between two hosts. We successively analyze in the following each defined *hold* fact with respect to the corresponding security rules.

Listing 5.12: Refining for better strategy assessment

```

security_rule(prohib, pie, 'MailStation', 'Read', 'MailServer', 'PopThreat') (1)
security_rule(prohib, pie, 'MailStation', 'ReadPop', 'MailServer', 'PopThreat') (2)
security_rule(prohib, pie, 'MailStation', 'ReadPop', 'ExchgServer', 'PopThreat') (3)

hold(pie, 'pc-charlie', 'POP3/retr', 'mel1', 'PopThreat') (a)
hold(pie, 'pc-charlie', _, 'mel1', 'PopThreat') (b)
hold(pie, 'pc-charlie', _, _, 'PopThreat') (c)

```

- **hold (a).** This *hold* fact explicitly defines all three concrete entities (subject, action and object), recommending thus a very specific countermeasure. It prohibits only the *POP3/retr* command between hosts *pc-charlie* and *mel1*. Considering the defined hierarchical graphs, this *hold* fact matches all three security rules. Whatever the defined rule at the abstract level, the result of the policy compilation will be the desired one with such an elementary *hold* fact.
- **hold (b).** Strategy may require not only to block the *POP3/retr* command, but also all POP3 commands which deal with the POP3 read activity. In this case, the action instantiated in the *hold* is the *any* variable. Rule (1) is no longer correct. Many actions are considered in the generic *Read* activity, which do not deal with the POP3 service. In particular, this will result in the prohibition of IMAP reading activities as well, which is not desired here. Only rules (2) and (3) lead to the desired result, since the activity is constrained enough to match only POP3-related actions.
- **hold (c).** Strategy may also recommend to enlarge response to other potentially threatened servers. For instance, a successful attack towards the *mel1* server is likely to work also for *mel2* and *mel3*. In such a case, a strategy is to enlarge response at the object level with the *any* variable. Rule (1) is not correct, since activity is not refined enough. Rule (2) is not correct anymore, since it does not consider the necessary view refinement to focus only on servers which deal with the POP3 service. Note that this is rather a model relevance issue than an actual negative side-effect regarding enforcement, since requiring to block POP3 reading commands on servers which do not provide the POP3 service will simply fail.

We observe in fact that there is a trade-off between the number of defined security rules and the specificity of *hold* facts. Even with generic rules, if *hold* facts correctly and precisely describe the current situation, there is simply no *hold* fact which will match an abstract rule which is not correct regarding enforcement. For instance, SMTP does not deal with any action considered in the activity *Read*, thus there is no reason that

a *hold* fact could associate both the *Read* activity and the SMTP service. One may thus choose between (1) defining generic abstract rules and define adequate constraints on *hold* facts to derive correct policy instances, or (2) provide abstract rules which are specific enough to possibly minimize constraints on *hold* facts.

We state that this trade-off mainly deals with strategy, and especially scale of response, as we will further explain in Chapter 6. We consider part of the response strategy the possibility to enlarge the scale of response through the use of the *any* Prolog variable, represented by '___'. Instead of providing response on a single concrete entity, the response will bear on all the subjects empowered in the matched role, all the actions considered in the matched activity, or all the objects used in the matched view.

Note that other operational constraints may explain refinement choices. In particular, in listing 5.11, it is likely that the operator will define *MailClient* instead of *PopClient*, for at least two reasons: (1) mail clients generally offer multiple protocols compliance, (2) it is difficult to enforce policy at the client-side, and users are generally assumed responsible for their own station and software (at least, it is possible to make recommendations, but it is difficult to force them use a specific software, with a specific configuration.).

5.7.2.3 Sample threat-related hold facts

We give in listing 5.13 sample definitions of threat-related *hold* facts, based upon information reported by alerts.

Listing 5.13: Sample threat-related hold facts

```
hold(corp,_, 'TCP/25', 'SMTP1', 'SmtptThreat') :-
    alert(CreateTime, Source, Target, Classification),
    reference(Classification, 'CVE-2005-0560'),
    service(Target, 'TCP/25'),
    hostname(Target, 'SMTP1').

hold(corp, 'outlook',_,_, 'MailClientThreat') :-
    alert(CreateTime, Source, Target, Classification),
    reference(Classification, 'CVE-2003-0007'),
    process(Source, 'outlook').
```

The first hold statement is related to an implementation fault allowing to exploit a buffer overflow resulting in the possibility for remote attackers to execute arbitrary code on the server. In such a case, since the server is clearly threatened, a suitable countermeasure is a victim-centric countermeasure. Thus, one would probably choose to block SMTP access for all subjects at the network level (port TCP/25), which is materialized by '___' in the Or-BAC subject field in listing 5.13.

The second hold statement is related to an operating fault in Microsoft Outlook 2002 used with V1 Exchange Server Security certificates, which results in the sending of messages in plaintext. This is both a server threat and a client threat. However, one can deal with such a threat by preventing the use of Outlook. Users are thus *de facto* prevented from native outlook access to the Exchange server. Such a client-side response both preserves the availability of the mail server and ensures that confidentiality issues brought up by the alert are correctly addressed. Note that according to the fact that the only constraint is to block the subject, *e.g.* the Outlook mail client, action and object and not instantiated since it applies to all relevant occurrences.

Note that these examples of *hold* facts are here defined totally statically, *i.e.* a definition is necessary for each considered vulnerability reference. In fact, it is here just for the sake of illustration, but we aim at providing means to factorize and automate this process at much as possible. Further elements going in this way are provided both in Chapter 6, dealing with response strategy and Chapter 7, providing explanations about the current implementation of the approach.

5.7.3 Minimal policy

The *minimal* policy deals with minimal requirements which are to be preserved whatever the threat. Minimal requirements are specified through *minimal* contexts, associated with the highest context priority (highest criticality). In the email use case, a minimal requirement is that at least one path to mail should be preserved whatever the current threat(s). When all paths to mailboxes are blocked by the reaction policy, OWA must be permitted with a higher priority by the minimal policy. OWA presents the advantage of being an indirect way to access mail, that is through the use of a web server being the actual client of the Exchange server. The administrator (or the operator) has a better control over the OWA web server than over a user mailstation. Moreover, with respect to figure 5.5, one can observe that PAROWA and MEL1 benefit from a direct access, isolated from client stations by a firewall. In the *MinimalMail* context, user stations can only pass through port TCP/80 of the firewall, which is very desirable to be kept open with regard to the multiple web services likely to be offered on a desktop environment.

Since the Exchange process is the core of the email application, especially because it centralizes mail repositories, we also formulate the requirement that no service-level response can shutdown the Exchange process. Otherwise, mail would be forbidden to all users. We assume that alternative countermeasures exist if Exchange is threatened, especially regarding access control (*e.g.* firewall rules, which allow to block some hosts without leading to total loss of service for others).

Listing 5.14 recaps this sample minimal policy for email access. According to the first *hold* statement, *MinimalMail* is active only when all four mechanisms to access mail are conjointly threatened. In such a case, the three first security rules effectively recommend to keep OWA open. The last security rule, associated with the last *hold* statement, ensures that the Exchange process is always kept active, whatever the considered *MailRole* and *MailActivity*.

Listing 5.14: Sample minimal policy for the email use case

```

security_rule(permission, pie, 'MailRole', 'AccessOWA', 'MailView', 'MinimalMail').
security_rule(permission, pie, 'MailRole', 'ReadOWA', 'MailView', 'MinimalMail').
security_rule(permission, pie, 'MailRole', 'WriteOWA', 'MailView', 'MinimalMail').

security_rule(permission, pie, 'MailRole', 'MailActivity', 'ExchgService', 'MinimalMail').

hold(pie, Subject, __, 'OWA1', 'MinimalMail') :-
    hold(pie, Subject, __, 'MEL1', 'PopThreat'),
    hold(pie, Subject, __, 'MEL1', 'ImapThreat'),
    hold(pie, Subject, __, 'MEL1', 'ExchangeThreat'),
    hold(pie, Subject, __, 'MEL1', 'OWAThreat').

hold(pie, __, __, 'Exchange2003', 'MinimalMail').

```

5.8 Modeling structured entities

Or-BAC concrete entities have been described along this chapter as elementary information, *e.g.* host names, port numbers, application names, etc. In fact, there is a need for structured entities modeling, especially for implementation purpose. The objective is clearly to regroup information which qualify a particular concrete entity (subject, action or object), may it be related to various identifiers (*e.g.* hostnames may be considered equivalent to IP addresses), or complementary information which may be useful for strategy or enforcement (*e.g.* netmask, process path or pid, user identifiers, etc.). We thus describe higher-level structured Or-BAC objects (*i.e.* non-elementary), namely *subjects*, *actions* and *objects*, which regroup common characteristics given the layer they belong to, namely *information*, *service* and *transport*.

Structuration of objects is done via XML-encoding. Let us consider the IDMEF *Node* class. Figure 5.12 recaps the corresponding XML schema. Table 5.1 gives an example of a *Node* sub-part which may be contained by an alert and the corresponding representation in Prolog⁴. In fact, it simply consists in a Prolog recursive list, each XML class being introduced with the use of the keyword *element*. Now, given this notation, one can manage XML-encoded data in Prolog. Importing XML data from the Prolog engine results in the creation of such a list, and exportation is successful if such a list is provided to the Prolog engine. In the following, we build upon the Swi-Prolog XML representation to provide structured Or-BAC objects, which are thus directly exportable in XML when necessary.

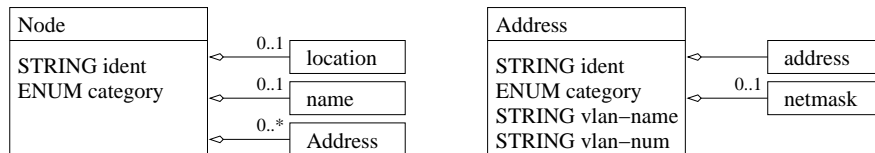


Figure 5.12: XML schema of the IDMEF Node class

<pre><Node category='unknown'> <name>mell</name> <Address category='ipv4-addr'> <address>192.168.1.50</address> </Address> </Node></pre>	<pre>[element('Node', [category='unknown'], [element('name', [], ['mell']), element('Address', [category='ipv4-addr'], [element('address', [], ['192.168.1.50'])])])])]</pre>
--	---

Table 5.1: IDMEF and Swi-Prolog Node examples

- **Subjects.** Subjects in our use case are modeled according to roles *MailUser*, *MailClient* and *MailStation* depending on the considered layer. They may thus

⁴Our experiments are based on the Swi-Prolog implementation of the Prolog language <http://www.swi-prolog.org>.

be related to users or groups, processes, or nodes, as represented in figure 5.13. A sample subject is shown in table 5.2.

1. **Information-level subjects**, namely *users* or *groups* represent information-level entities actually accessing resources. Users and groups are characterized by a *name* and an identifier, namely *Uid* or *Gid*. They are mapped onto *Source.User* class in IDMEF alerts.
2. **Service-level subjects**, namely *processes*, represent client software entities accessing service software entities. Processes are characterized by a *name* and an identifier, namely *pid*, and possibly a *path* locating the application. They are mapped onto *Source.Process* class in IDMEF alerts. Information about service-level subjects may also be inferred from the IDMEF *Classification.Reference* class as explained in the following in 6.3.3.
3. **Transport-level subjects**, namely *hosts*, represent client hardware entities accessing server hardware entities. Hosts are characterized by a DNS *name*, an IP *address* and a *netmask*. They are mapped onto *Source.Node* class in IDMEF alerts.

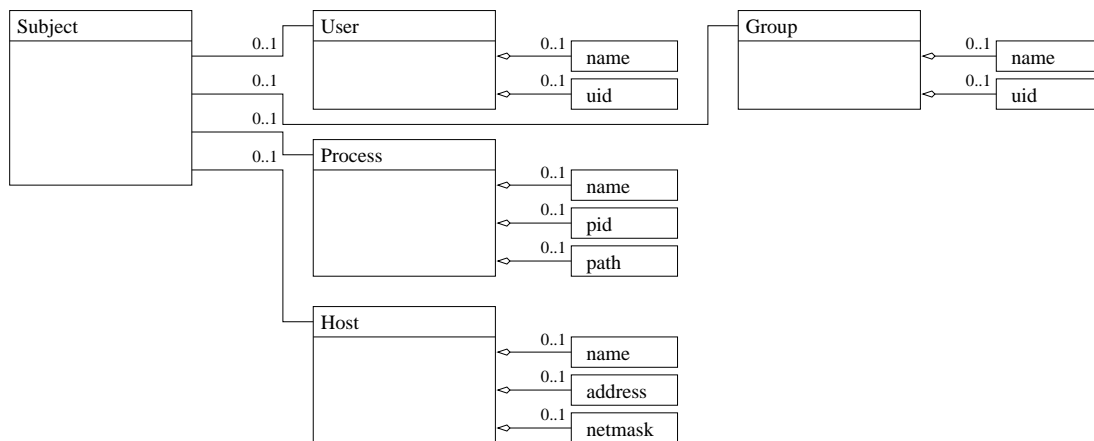


Figure 5.13: Modeling structured subjects

<pre> <Subject> <Host> <name>mel1</name> <address>192.168.1.50</address> <netmask>255.255.255.0</netmask> </Host> </Subject> </pre>	<pre> [element('Subject', [], [element('Host', [], [element('name', [], [mel1]), element('address', [], ['192.168.1.50']), element('netmask', [], ['255.255.255.0'])])])]</pre>
---	--

Table 5.2: XML and Swi-Prolog transport-level subject examples

- **Objects.** Objects are modeled according to views *Mailbox*, *MailService* and *MailServer* depending on the considered layer. They may thus be related to

users (*mailboxes*), processes, or nodes, as represented in figure 5.14. A sample object is shown in table 5.3.

1. **Information-level objects**, namely *mailboxes*, represent actually accessed resources. Mailboxes are characterized by an email *address*, and possibly a *path* locating the mail repository. They are mapped onto *Target.User* and *Target.File* in IDMEF alerts.
2. **Service-level objects**, namely *processes*, represent server software entities accessed by client software entities. Processes are characterized by a *name* and an identifier (*pid*), and possibly a *path* locating the application. They are mapped onto *Target.Process* class in IDMEF alerts. Information about service-level objects may also be inferred from *Classification.Reference* as explained in 6.3.3.
3. **Transport-level objects**, namely *hosts*, represent server hardware entities accessed by server hardware entities. Hosts are characterized by a DNS *name*, an IP *address* and a *netmask*. They are mapped onto *Target.Node* class in IDMEF alerts.

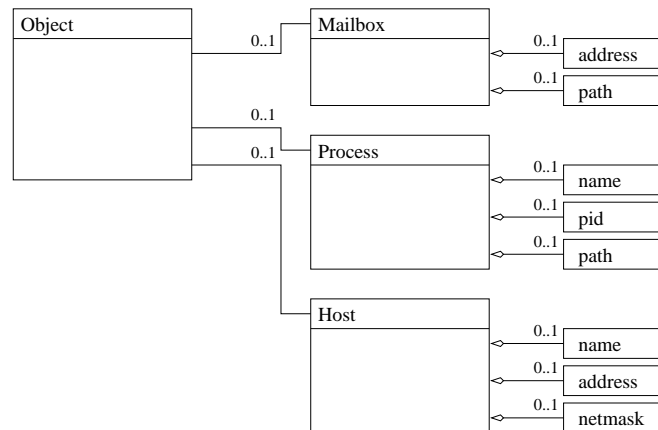


Figure 5.14: Modeling structured objects

<pre> <Object> <Process> <name>popd</name> <pid></pid> <path>'/etc/popd'</path> </Process> </Object> </pre>	<pre> [element('Object', [], [element('Process', [], [element('name', [], [popd]), element('pid', [], []), element('path', [], ['etc/popd'])])])]</pre>
---	--

Table 5.3: XML and Swi-Prolog service-level object examples

Note here that the *pid* is not defined in the object. We consider a generic instance for each application, but we do not want to describe every instance actually

running through the knowledge of its pid. The objective is clearly to identify a process, and we consider that its name and possibly path is sufficient. However, the pid may be of interest for enforcement, in particular to precisely identify the process to kill. We state that if necessary, the information may be found in the alert requiring such a response, or at least in an additional database reflecting the information system cartography, especially the inventory of the processes actually running on the hosts.

- **Actions.** Actions in our use case are modeled according to activities *Access*, *Read* and *Write*, but this does not provide any particular structure to actions. Structuration is again provided by the three layers. However, actions are structured similarly at the service level than at the transport level, as explained in the following. A sample action is shown in table 5.4.

1. **Information-level actions**, *i.e.* applicative *commands*, represent commands which are executed by mean of applicative protocols. Commands are characterized by a *protocol* (*e.g.* POP3) and a command *name* (*e.g.* retr). Regarding IDMEF alerts they may be mapped either an *AdditionalData* since no other IDMEF class provides this kind of data natively, or inferred from *Classification.Reference*.
2. **Service-level actions**, *i.e.* *services*, represent the port actually binded by the server software. Service is characterized by a *name* (*e.g.* POP3) and a *port* number (*e.g.* TCP/110). Service-level actions are found in the *Target.Service* class. They can also be sometimes inferred from the IDMEF *Classification.Reference* class.
3. **Transport-level actions**, *i.e.* *service* represent the destination port used to transport packets. Service is characterized by a *name* (*e.g.* POP3) and a *port* number (*e.g.* TCP/110). Transport-level actions are found in the *Target.Service* class. They can also be sometimes inferred from *Classification.Reference*.

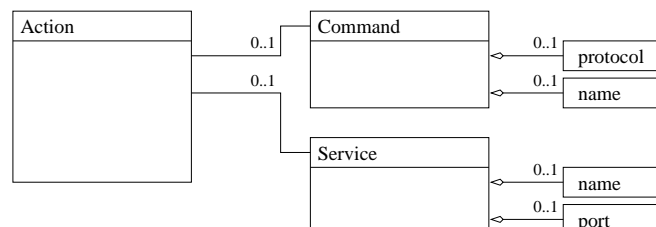


Figure 5.15: Modeling structured actions

<pre><Action> <Command> <protocol>POP3</protocol> <name>retr</name> </Command> </Action></pre>	<pre>[element('Action', [], [element('Command', [], [element('protocol', [], [POP3]), element('name', [], [retr])],)])]</pre>
--	---

Table 5.4: XML and Swi-Prolog applicative-level action examples

5.9 Conclusion

In this chapter, we showed through a realistic application dealing with email access that it is possible to model threat response through the use of an Or-BAC policy. We explained how Or-BAC provides the possibility to describe roles, activities and views grouping concrete objects according to the desired strategy. In particular, we showed that a fine-grained definition of abstract entities, especially activities, facilitates response scalability management.

We then proposed to model the policy according to three sub-policies, namely the *operational* policy, the *reaction* policy and the *minimal* policy. The operational policy is the traditional access control policy. Sometimes, minimal requirements are included, which we model as the minimal policy. Finally, the significant part of the proposal relies on modeling a reaction policy, *i.e.* a part of the policy which deals with response according to detected threat. We explained how contexts may be modeled and activated with respect to these three sub-policies. Finally, we provided sample definitions of the policy, related to the email use case.

Along this chapter, we discussed refinement of Or-BAC abstract entities regarding strategy, namely roles, activities, views, but also contexts. We observed that some refinements are of interest, especially to facilitate the management of scale of response. However, we stated that the most important part of response strategy relies at the concrete level, *i.e.* in the instantiation of Or-BAC *hold* facts. Indeed, the crucial issue is clearly to consider information available into alerts and activate Or-BAC *hold* facts with respect to threat, and ensuring adequate response at policy compilation time. Dealing with information at the concrete level, we finally introduced in a last section the need for structured concrete entities.

Chapter 6

Mapping Policy Instances onto Alerts: Response Strategy

Contents

6.1	Introduction	93
6.2	Response strategy taxonomy	96
6.2.1	Direction	96
6.2.2	Target layer	97
6.2.3	Scale of response	98
6.3	Information to consider	99
6.3.1	Information available into alerts	99
6.3.2	Information available from models	101
6.3.3	Information deduced from alerts	101
6.4	Conclusion	107

6.1 Introduction

We have proposed in Chapter 3 a generic architecture to provide response to threat, based on the continuous re-evaluation of the security policy parametered by alerts reported by intrusion detection systems. Chapter 4 describes how the Or-BAC model can be used to fit the dynamic response requirements, and Chapter 5 brings up an application with the email use case. We show that it is possible to derive new policy instances considering alerts. However, such an automated system would present significant disadvantages without any assurance that deployed countermeasures actually respond to threat. We remind that regarding alerts, we consider out-of-scope their characterization, and rely on the fact that alerts reported by the Alert Correlation Engine may be trusted as actual alerts, *i.e.* that there is no false alert. This hypothesis is crucial in our work to avoid undesirable side-effects, such as self-inflicted denial-of-service.

However, reliable alerts are not the only condition to ensure automated response to threat. Indeed, a second major point is that selected countermeasures must be adequate

regarding threat, which means that it is essential (1) to correctly assess threat, and (2) to select the most appropriate countermeasure.

Threat assessment is strongly linked with the relevance and the consistency of the alerts given by the ACE. The PIE should indeed receive alerts characterizing as well as possible what is happening, that is detailed information about source, type and target of the attack / intrusion.

Regarding countermeasures themselves, the Common Criteria for Information Technology Security Evaluation [2] propose to evaluate the confidence required by an owner that countermeasures minimize risk to assets through two parameters: *sufficiency* and *correctness* of the countermeasures (see figure 6.1), defined as follows:

- **Sufficiency:** if the countermeasures do what they claim to do, the threats to the assets are countered;
- **Correctness:** the countermeasures do what they claim to do.

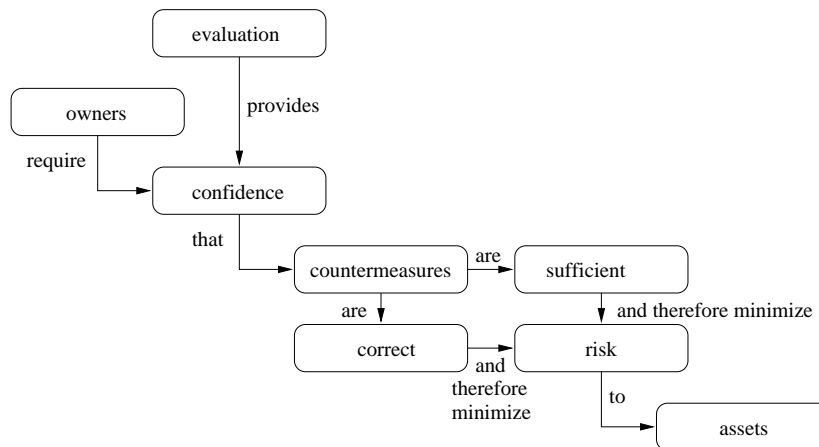


Figure 6.1: Evaluation concepts and relationships in the Common Criteria

Sufficiency and correctness parameters do not take into account variables different from security, that is for instance, performance and convenience. A countermeasure could be both sufficient and correct, regarding risk, but the CC approach does not guarantee that there is no undesirable side-effect from a performance or convenience point of view. In fact, we shall also consider that sometimes, exposing assets to risk is necessary, because beyond security requirements, one also has to consider that the first requirement of an information system is to provide service and resources. This means that one should balance risk with actual effects of the countermeasures, to ensure that response does not lead to a situation which is even worse than what it would have been in the absence of response.

We do not consider correctness. In fact, we assume that countermeasures actually do what they claim to do. At least, we assume that policy validation is part of the future work. We also assume that balancing risk with actual effects of the countermeasures

is implicitly realized thanks to *minimal* contexts, ensuring requirements which cannot be overridden, even in case of intrusion.

We focus on sufficiency, that is we would like to ensure that selected countermeasures actually counter considered threats to assets, if they do what they claim to do. Hence, we discuss here how one can map policy instances corresponding to countermeasures onto alerts. Figure 6.2 shows the distinction between the two functionalities of the PIE, namely the Threat and Response Characterization Engine (TRCE) and the Policy Core Engine (PCE). The PCE is responsible for potential conflict resolution (dynamic priority assessment) and policy compilation, which has already been discussed in chapter 4. We thus focus here on threat and response characterization, which purpose is clearly to satisfy the sufficiency property.

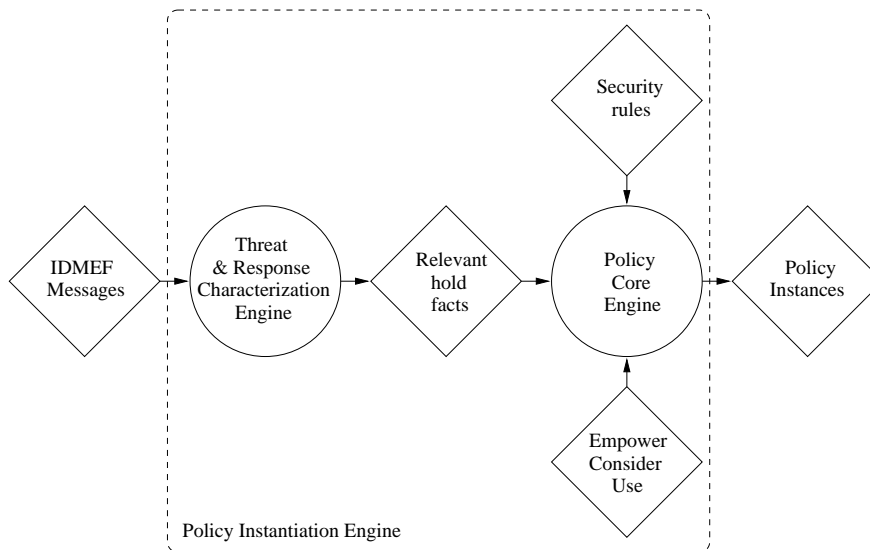


Figure 6.2: Detailed view of the PIE

In the following we identify three main functions of the TRCE, namely *syntactic mappings*, *enrichment* and *strategy application*. We define these three functions as follows:

- **Syntactic mappings.** This phase consists in parsing IDMEF messages to assess every valuable information available in alerts. Syntactic mappings provide identification of the threat source and target, and the kind of threat, especially characterized by a vulnerability reference.
- **Enrichment.** This phase either deals with completion of information missing in alerts or additional information which may be deduced from available ones. Completion of information may especially consider the model of the world, including the definition of the policy, which provides data about subjects, actions and objects which are considered in the information system model. Additional

information may be in particular deduced from vulnerability references, which refer to many properties, including for instance the target resource.

- **Strategy application.** This phase provides the crucial decisional function of the system. Depending on information provided by both syntactic mappings and enrichment phases, but also considering possibly user-defined parameters, strategy application ensures that activated *hold* facts are relevant regarding threat, but also with respect to the requirements of the administrator. For this purpose, vulnerability references are to be strongly considered, in order to trigger threat context, but also possibly to orient strategy. Other parameters inherited from the syntactic mappings may also enter the strategy process, especially dealing with the IDMEF *Assessment* class.

In a first section, we present a simple response strategy taxonomy, to explain the scope of our approach regarding response strategy. Then, in a second section, we discuss the kind of information one can consider to characterize threat and provide adequate response. We distinguish information which are available from alerts, dealing with syntactic mappings, from information which are available from model and information which can be deduced from alerts, dealing with enrichment and possibly strategy application.

6.2 Response strategy taxonomy

We provide here a simple response strategy taxonomy, relevant to our purpose, which is illustrated by figure 6.3.

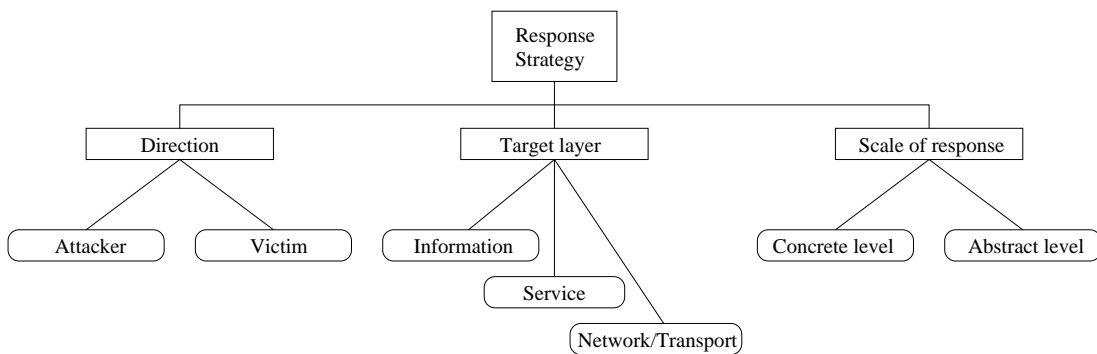


Figure 6.3: Response strategy taxonomy

6.2.1 Direction

Response can focus on two main directions: attacker and victim.

Attacker-centric response. Attacker-centric response focuses on the attacker, *i.e.* the threat agent. The objective is to contain the threat origin. We distinguish here two kinds of threat origins: those coming from the inside, namely internal attackers, which are under the administration domain, and those coming from the outside, namely external attackers, which are generally not under the operator's control.

- **Internal attackers.** Alerts are often likely to report internal users as the source of a fault which in fact reveals to be a non-malicious fault, *i.e.* a side-effect of the normal use of the information system. However, actual internal attackers, and thus potential malicious activity from the inside, must also be considered, at least for two reasons. First, it is easier to attack from inside, and second, outsiders generally first compromise an inside host to realize an internal attack.
- **External attackers.** Attacker-centric response for external attackers is not so easy to implement than for internal ones. It is often difficult to identify the real source of the attack. Most of the time, sensors report as source the web proxy relaying the traffic from the outside. Moreover, even when attackers are clearly identified, they are mainly out of range for the operator. Attacker-centric response in case of external attackers mainly deals with access control and filtering at the border of the information system (administration domain boundaries).

Whatever the attacker (internal or external), the major issue is that attackers usually use spoofing, in order to masquerade their identity (*e.g.* mainly their IP address). Decoy sources have to be considered in response strategy, since a response targeting a decoy source may present undesirable side-effects regarding legitimate users.

Victim-centric response. Victim-centric response focuses on the assets, either user assets or service assets.

- **User assets.** User assets deal with data resources, such as files and mailboxes. Victim-centric response includes access control to such assets.
- **Service assets.** Reconfiguring services is another way to realize victim-centric response. According to the defined policy, services may be shutdown, (re-)started or even reconfigured with more or less functions enabled (*e.g.* reconfiguring the Apache web service without its PHP module when it is threatened).

6.2.2 Target layer

Response can be provided at different target layers according to (1) the kind of threat, (2) the information available in the alerts, and (3) the desired strategy. These layers are those presented in the email use case: Network / transport level, service level, and information level.

1. **Kind of threat.** Some alerts refer to very low-level attacks, *e.g.* syn-flooding attacks, which is a network / transport level attack, whereas others correspond to high-level attacks, *e.g.* cross-site scripting attacks or brute-force password guessing attacks, which deal with application-level protocols.

2. **Information available.** Depending on the information available to the ACE, alerts produced may be more or less informative. If an alert reports information at the network level only, it is difficult to envision better response than blocking the source or destination.
3. **Desired strategy.** Regarding the kind of threat and the available information, one can sometimes choose among multiple countermeasures. If all three layers of the model are available, different countermeasures can be applied in parallel or sequence at the information level, at the service level, or at the network level.

Network / transport level. At the network / transport level, response consists in access control lists and filtering rules. One may want to block traffic from or to a given host, based on IP addresses and port numbers for more fine-grained responses.

Service level. At the service level, response deals with service shutdown, restart and reconfiguration (for instance to restart a given service on a different port).

Information level. At the information level, access control to application resources is enforced by access rights modification, temporary deleting of user entries in Active Directory, applicative filtering (*e.g.* to prohibit specific applicative commands), etc.

6.2.3 Scale of response

The response strategy should also deal with the scale of response. Even if source and destination are fully qualified in the alert, a very targeted response is not necessarily the best adequate considering threat. Let us consider an attack exploiting a vulnerability of a given service instance. Such an attack could probably affect other instances of this service, if any exists. In such a case, the best suitable response would probably be to consider all instances of the considered service, that is take countermeasures not only on the host or service under attack, but also to take measures on other hosts or services which are potential victims of a similar attack. Similarly, it is sometimes relevant to take measures on a group of users, or on a whole sub-network or network, instead of a single user or a single host. Hence, we distinguish here concrete-level response from abstract-level response as explained in the following.

Concrete-level response. We refer to concrete-level response when response deals with specific concrete entities. For instance, a very targeted response filtering a specific command of a given service (*POP/user*) is considered to be a concrete-level response, as well as a response preventing a particular user from accessing a specific mailbox.

Abstract-level response. We refer to abstract-level response when response is enlarged to a set of objects dealing with a same abstract entity. For instance, in some cases, instead of blocking the use of the specific *POP/user* command, one may choose to filter all POP access related commands, which are grouped under activity *AccessPop*. Abstract-level response is only a denomination characterizing the scale of response,

while response itself is actually enforced at the concrete level, that is instantiating as many concrete policy rules as there are objects considered among the given abstract entity.

6.3 Information to consider

6.3.1 Information available into alerts

We analyze here the different sub-classes of IDMEF, discussing the relevance of available information with respect to response. The objective is both to characterize threat and to provide adequate countermeasures expression.

6.3.1.1 Createtime (alert timestamp)

It ensures temporal consistency of the alerts. Received alerts may be obsolete regarding response since they may accidentally or purposefully be delayed. The response system uses *Createtime* to determine whether the alert should be responded to.

6.3.1.2 Source

According to the formulated ACE requirements, the *Source* class represents attack source, the attacker, or at least what is considered the origin of the attack. The *Source* class provides the following different sub-classes and attributes:

- **spoofed.** The *spoofed* attribute evaluates if we can trust the source as being the actual origin of the threat. In a wide majority of cases, it is however very likely to be unknown, since determining whether the source is spoofed or not is far from trivial.
- **Node.** The *Node* class includes node *name*, *address* and *netmask*. It is used to instantiate subjects at the network / transport level. We only consider in our approach IPv4 addresses, but the approach may be extended to other kinds of addresses (*e.g.* IPv6 or MAC addresses), according to the *category* attribute.
- **User.** The *User* class contains the *name* and *number* (uid) associated with the source user. It is used for subjects instantiation at the information level.
- **Process.** The *Process* class provides information about a process executed on the host. It reports a client software instance as a subject, like *thunderbird* in the email use case example. A process is characterized by a *name*, a *path* and possibly a *pid* (process identifier).
- **Service.** The *Service* class provides the *name* and *port* of the considered service. Since the source mostly represents the client side, the information is mainly relevant for fine-grained access control including source port.

6.3.1.3 Target

According to the formulated ACE requirements, the *Target* class represents the attack target, that is the victim. The *Target* class provides the following sub-classes and attributes:

- **Node.** The target *Node* class allows instantiation of objects at the network / transport level. It mostly provides physical locations for policy enforcement at the server side, allowing for instance to filter traffic considered to be threatening a given server.
- **User.** The target *User* class mostly represents a threatened user account, like a mailbox. It instantiates entities at the information level.
- **Process.** The target *Process* class provides *name*, *path* and possibly *pid* of the considered targeted process (*e.g.* exchange). It allows objects instantiation at the service level.
- **Service.** The target *Service* class provides the *port* number(s) and *name* of targeted service. It instantiates actions at the service and network levels.
- **File.** The target *File* class reports information about a file object which has been created, modified or deleted on the target. In our system, it is used to report accessed mailboxes (file used for mail storage on the server). Hence, it is generally assigned to an object at the information level.

6.3.1.4 Classification

The *Classification* class provides the alert *Reference*, expressed according to a vulnerability database, such as CVE, Bugtraq or OSVDB.

- **Reference.** It characterizes threat contexts. As mentioned in the email use case, CVE-2005-1133 activates the context *pop_attack*. However, this is not the only available information in the reference. One can deduce from the description of CVE-2005-1133 that the targeted object is a *pop3* service and that the action used to realize the attack is the *POP/user* applicative action. Moreover, such an attack may be realized by any subject, without any particular system requirement, except the possibility to communicate with a pop server on port tcp/110. This reference classifies the threat as a disclosure of information. The reference is thus both used for threat context activation, and to orient mapping strategy, thanks to information which can be deduced.

6.3.1.5 Assessment

Regarding the *Assessment* class, we are interested in the *Impact* sub-class.

- **Impact.** It includes the *severity* and *type* of the alerts, and *completion* of the attempt. Regarding mappings, *severity* is especially used to manage scale of

response. For instance, the response system can be configured so that a response which is specific to a subject must be extended to all subjects if the severity is high. The *type* is mostly relevant to orient strategy. The *completion* parameter is also considered in strategy, to mitigate response, reducing for instance scale of response if the attempt has not been completed (no effective intrusion), or even to avoid response in such a case.

- **Confidence.** The *Confidence* class brings up information about the confidence the analyzer evaluates to the reported alert. We assume that confidence is optimum, but in a more realistic application, this may be used to prioritize some alerts towards others.

6.3.2 Information available from models

Some information may lack in the alerts. For instance, an alert may report only a user *name* as a subject, or only a host *address*. Models like the policy definition and/or an up-to-date model of the information system complete the *User* structured entity by adding the *uid* parameter, and the *Host* structured entity by adding the *name* and *netmask* parameters. Such an enrichment is useful for policy compilation, in order to correctly match concrete entities defined in the policy, and for enforcement. For instance, a misqualified alert may report only a process *name*. However, the knowledge of at least the *pid* or the *path* may be necessary for actual enforcement. This information can be gathered thanks to an up-to-date model of the information system, given the considered host, which IP address should be available from alert.

6.3.3 Information deduced from alerts

A lot of information can be deduced from vulnerability references reported by alerts, resulting from a search in the vulnerability databases. This can be used both for enrichment of explicitly available data, and to orient strategy.

6.3.3.1 Threat assessment and response characterization

Additional input for threat analysis. In the following, we consider threats which are relevant to the environment of the email use case. We have carried out a simple analysis using the National Vulnerability Database¹ (NVD) as an additional input the security officer can make use of to orient strategy. We have used successively the keywords “exchange” (table 6.1), “pop3” (table 6.2), “imap” (table 6.3), “outlook” (table 6.4), “thunderbird” (table 6.5) and “firefox” (table 6.6). We also give a few additional generic attacks or network attacks in 6.7. The tables referenced present a partial result of the search. Column 1 gives the CVE identifier of the vulnerability. Column 2 evaluates the relevance of this vulnerability to the use case; the vulnerability is highly relevant if the vulnerable software is clearly present in the use case, low if the vulnerability is clearly absent from the use case, and medium if we could not

¹<http://nvd.nist.gov>

determine it clearly. Column 3 contains the Common Vulnerability Scoring System² (CVSS) score for the vulnerability, computed from the CVSS vector. The CVSS vector itself is partially explicated in the impact column, where the (*C*) indicates complete impact and the (*P*) indicates partial impact of the vulnerability on either availability, confidentiality or integrity. Finally, columns 5, 6 and 7 analyze the model abstract or concrete entities that are impacted by the vulnerability.

The tables contain only cursory information about each vulnerability. The reader is referred to the NVD for a detailed description of the vulnerabilities associated with each CVE reference, and in particular a textual description and the complete CVSS vector. The tables only show partial results from our search, and the search term themselves are not exhaustive; a thorough examination should include all installed software (we only included software presented in the use case). It should take into account the detection capability, i.e. the fact that an attempt to use the vulnerability will result in an alert provided by one of the sensors; if an attack using one of these vulnerabilities cannot be detected, then no response can be included in the threat response system. Also, in the specific case of software vulnerabilities, patching applications will change the relevance of the vulnerability for the threat response system, and thus may change the associated response. We do consider though that there will be some time between vulnerability discovery and patching where such response will be important, and that non-software vulnerabilities (*e.g.* password guessing) are relevant to the threat response system. The proposed analysis merely illustrates the technology, but must of course be updated when new vulnerabilities are discovered.

Objective. The objective is clearly to deal with Or-BAC *hold* statements activation. Contexts are mapped onto references, and analysis of abstract or concrete entities may provide valuable information for the instantiation of *subjects*, *actions* and *objects*. However, the reader should note that we first focus on information directly available into alerts before taking into account the interpretation of the references given in the tables. For instance, if an alert reports a threat towards the *exchange* process, we do not need to get the information in the table. The objective of the tables is to orient the response strategy. They may also provide data in case of misqualified alerts. For instance, an alert which would report only network / transport-level information and a vulnerability reference does not mean that one is compelled to provide brutal response at the network / transport-level. In such a case, such tables provide data needed for more fine-grained and thus better adapted response.

Analysis output. Analysis results in a list of the roles, activities and views, or subjects, actions and objects potentially impacted by the vulnerability. Obtained triples thus directly provide entities on which the system should act to provide response. The definition of such triples corresponds to our understanding of the information available in the description of each vulnerability. We use the *any* keyword to denote that any

²<http://www.first.org/cvss/cvss-guide.html>

entity can participate in the vulnerability (the value of this field is not important in exploiting the vulnerability directly), italics to denote concrete entities, and plaintext to denote abstract entities. Note however that even when the table mentions an abstract entity, the related Or-BAC *hold* facts are instantiated at the concrete level.

CVE Identifier	Relevance	CVSS Severity	Impact	Activity (Action)	View (Object)	Role (Subject)
CVE-2006-1193	High	1,9	Unauthorized modification	Read	<i>OWA</i>	Mailuser
CVE-2006-0027	High	7	User access, Confidentiality (P), Integrity (P), Availability (P), DoS	Write, Transfer	<i>Exchange</i>	<i>any</i>
CVE-2006-0002	High	7	User access, Confidentiality (P), Integrity (P), Availability (P), DoS	Write, Transfer	<i>Exchange</i>	<i>any</i>
				Read	<i>any</i>	<i>Outlook</i>
CVE-2005-1987	High	7	User access, Confidentiality (P), Integrity (P), Availability (P), DoS	Write, Transfer	<i>Exchange</i>	<i>any</i>
CVE-2005-0738	High	3,3	DoS	Read	<i>Exchange</i>	Mail user
CVE-2005-0563	High	3,3	Allows unauthorized modification	Read	<i>OWA</i>	<i>any</i>
CVE-2005-0560	High	7	Unauthorized access, Confidentiality (P), Integrity (P), Availability (P), DoS	Write Transfer	<i>Exchange</i>	<i>any</i>
CVE-2005-0420	High	7	Unauthorized access, Confidentiality (P), Integrity (P), Availability (P), DoS	Access	<i>OWA</i>	<i>any</i>
CVE-2005-0044	High	7	User access, Confidentiality (P), Integrity (P), Availability (P), DoS	Write, Transfer	<i>Exchange</i>	<i>any</i>
				Read	<i>any</i>	<i>any</i>
CVE-2004-0840	High	10	Admin access, Confidentiality (C), Integrity (C), Availability (C), DoS	Write, Transfer	<i>Exchange</i>	<i>any</i>
CVE-2004-0203	High	10	Unauthorized access, Confidentiality (P), Integrity (P), Availability (P), DoS	Read	<i>OWA</i>	<i>any</i>
CVE-2003-0904	High	5,6	Unauthorized access, Confidentiality (P), Integrity (P), Availability (P), DoS	Access	<i>OWA</i>	<i>any</i>
CVE-2003-0714	High	8	Unauthorized access, Confidentiality (P), Integrity (P), Availability (P), DoS	Write Transfer	<i>Exchange</i>	<i>any</i>
CVE-2003-0712	High	7	User access, Confidentiality (P), Integrity (P), Availability (P), DoS	Read	<i>OWA</i>	<i>any</i>

Table 6.1: Example of Exchange threats considered relevant for the use case

6.3.3.2 Discussion

Table 6.1 contains more references than the others, since the exchange process, which is the core of the presented use case, contains more vulnerabilities than client-side entities. Also, exchange supports multiple activities, thus increasing complexity. We do not provide any table for the SMTP service, since vulnerabilities listed for Exchange

CVE Identifier	Relevance	CVSS Severity	Impact	Activity (Action)	View (Object)	Role (Subject)
CVE-2006-7040	Low	3,3	DoS	<i>TOP</i>	<i>POP3</i>	<i>any</i>
CVE-2006-6940	Low	10	Admin access, Confidentiality (C), Integrity (C), Availability (C), DoS	Write, Read, Transfer	<i>OWA</i> , <i>POP3</i>	<i>any</i>
CVE-2005-1133	Low	3,3	Unauthorized disclosure of information	<i>POP/user</i>	<i>POP3</i>	<i>any</i>
CVE-1999-0822	Low	10	Admin access Confidentiality (C), Integrity (C), Availability (C), DoS	<i>POP/auth</i>	<i>POP3</i>	<i>any</i>

Table 6.2: Example of pop3 threats considered relevant for the use case

CVE Identifier	Relevance	CVSS Severity	Impact	Activity (Action)	View (Object)	Role (Subject)
CVE-2007-1825	Medium	7	Unauthorized access Confidentiality (P), Integrity (P), Availability (P),	Write	<i>HTTP</i>	WebClient
CVE-2007-1675	Low	10	Admin access Confidentiality (C), Integrity (C), Availability (C),	<i>IMAP/login</i>	<i>IMAP</i>	<i>any</i>
CVE-2006-7039	Low	2,3	Unauthorized disclosure of information	Read	<i>IMAP</i>	<i>any</i>
CVE-2006-2917	Low	2,8	Unauthorized disclosure of information, Unauthorized modification	<i>IMAP/create</i> <i>IMAP/select</i> <i>IMAP/delete</i> <i>IMAP/rename</i> <i>IMAP/copy</i> <i>IMAP/append</i> <i>IMAP/list</i>	<i>IMAP</i>	<i>any</i>
CVE-2006-1255	Low	7	Unauthorized access Confidentiality (C), Integrity (C), Availability (C)	<i>IMAP/login</i> <i>IMAP/select</i>	<i>IMAP</i>	<i>any</i>

Table 6.3: Example of imap threats considered relevant for the use case

CVE Identifier	Relevance	CVSS Severity	Impact	Activity (Action)	View (Object)	Role (Subject)
CVE-2006-6659	High	2,3	DoS	Read	<i>any</i>	<i>Outlook</i>
CVE-2006-4868	High	8	Admin access Confidentiality (C), Integrity (C), Availability (C), DoS	Read	<i>any</i>	<i>Outlook</i>
CVE-2006-2055	High	2,3	Allows unauthorized disclosure of information	<i>any</i>	<i>any</i>	<i>Outlook</i>
CVE-2006-1305	High	1,9	DoS	Read	<i>any</i>	<i>Outlook</i>
CVE-2003-0007	High	3,3	Allows unauthorized disclosure of information	Write	<i>any</i>	<i>Outlook</i>

Table 6.4: Example of outlook threats considered relevant for the use case

impact the SMTP service of Exchange. In addition, the Exchange table also reports some vulnerabilities which are linked with OWA.

We give in the following our interpretation of the provided tables. Analysis includes in particular consideration of threat side, *i.e.* client or server, and determination of concrete or abstract entities.

Prevalence of any for subjects and roles at the server-side. Service vulnerabilities, given in table 6.1 for Exchange, table 6.2 for POP3, and table 6.3 for IMAP, include a fine-grained description of objects / views. References in these tables rarely give information about the client-side, except in three main cases:

CVE Identifier	Relevance	CVSS Severity	Impact	Activity (Action)	View (Object)	Role (Subject)
CVE-2007-1558	High	7	Unauthorized access, Confidentiality (P), Integrity (P), Availability (P),	APOP,	any	Thunderbird
CVE-2007-1282	High	10	Admin access Confidentiality (C), Integrity (C), Availability (C),	Read	any	Thunderbird
CVE-2007-0777	High	10	Unauthorized disclosure of information Unauthorized modification, DoS,	Read	any	Thunderbird
CVE-2007-0775	High	3,9	User account access Confidentiality (P), Integrity (P), Availability (P),	Read	any	Thunderbird
CVE-2006-6505	High	5,6	Unauthorized access Confidentiality (P), Integrity (P), Availability (P),	Read	any	Thunderbird

Table 6.5: Example of Thunderbird threats considered relevant for the use case

CVE Identifier	Relevance	CVSS Severity	Impact	Activity (Action)	View (Object)	Role (Subject)
CVE-2007-2162	High	3,3	DoS	Read	any	Firefox
CVE-2007-1970	High	2,3	Unauthorized disclosure of information	Read	any	Firefox
CVE-2007-1762	High	2,3	Unauthorized modification	Read	any	Firefox
CVE-2007-1736	High	7	Unauthorized disclosure of information, Unauthorized modification, DoS,	Read	any	Firefox
CVE-2007-0994	High	5,6	User account access, Confidentiality (P), Integrity (P), Availability (P),	Read	any	Firefox

Table 6.6: Example of Firefox threats considered relevant for the use case

CVE Identifier	Relevance	CVSS Severity	Impact	Activity (Action)	View (Object)	Role (Subject)
Password guessing	High	-	User access	Access	any	any
CVE-1999-0116	Medium	3,3	DoS	any	any	any

Table 6.7: Example of generic threats and network threats considered relevant for the use case

1. Some references (*e.g.* CVE-2006-0002) impact both server side and client side. The NVD information is unclear on whether it is the client or the server which is responsible for the exploited vulnerability. It may sometimes be both, especially dealing with Outlook interfaced with Exchange. Indeed, since they are both provided by a same vendor (Microsoft), some specification or implementation faults are likely to exist both client-side as well as server-side (*e.g.* protocol specification or implementation faults). Sometimes, we also face vulnerabilities which are exploited at the server side, for instance executing arbitrary code, but triggered through the action of the client. Indeed, many references dealing with mail are based on the sending of crafted messages which aim at exploiting a vulnerability in functions linked with the *Read* activity at the client-side (*e.g.* parsing or interpreting libraries, alike HTML or VML parsing). In such cases, although the actual impact is server-sided, the question which arises regarding response is to decide at strategy level whether it is better to block reading at the client side or writing at the server side.

2. The faulty component has been identified, but the table does not take into account natural dependencies. For example, all the table entries listing *OWA* as the object should list *firefox* as the subject, since it is the email client (web browser) used to connect to *OWA*, or at least the role *Webclient*, if the model includes multiple web clients. Taking these dependencies into account in the table is complex because we then need to determine whether the client software is impacted by the flaw and how; for all the cross-site scripting vulnerabilities, it is unclear to us whether *firefox* would effectively be vulnerable and whether the user making the final decision would interact in a dangerous way with the server. We err on the side of caution and consider that this information should not be part of the decision process.
3. Some vulnerabilities refer to attacks which require prerequisites to qualify at least the role attribute, *e.g.* *CVE-2006-1193* and *CVE-2005-0738*, requiring to be logged in with role *Mailuser*. *CVE-2005-0738* is exploited by deleting or moving a folder with specific properties, which requires *a priori* login to the service. In some other circumstances, like *CVE-2007-1825*, the role may be identified, namely the subject must be a *Webclient*, since it deals with the *HTTP* activity.

Prevalence of any for objects and views at the client-side. Client-side vulnerabilities are given in table 6.4 for Outlook, table 6.5 for Thunderbird, and table 6.6 for Firefox. Client application vulnerabilities provide a fine-grained description of subjects / roles, but rarely explicit server-side information. However, some cases already given in the Exchange table clearly show that client-side related threats may implicate a specific client allowing an attack on a specific server (*e.g.* especially considering Outlook / Exchange). In such cases, dependencies may be considered, and subject / roles would be likely to appear in the table.

Prevalence of concrete entities as subjects and objects. Our search terms were mostly server-oriented, hence it is quite natural that we obtain server software vulnerabilities attached to software objects. Similarly, regarding client-oriented tables, we obtain client software vulnerabilities attached to software subjects. Exceptions appear at the server side in cases like *CVE-2006-0002*, where both Exchange and Outlook are impacted by the vulnerability.

Prevalence of abstract entities as activities. Many vulnerability descriptions do not refer to a specific action from the user. Exceptions appear when vulnerabilities involve specific application commands, like *POP/auth* or *IMAP/login* for instance. In the other cases, we can identify a general activity that the user is performing, but not the exact action ; this would probably be possible if we analyzed attack code, but it is unlikely that a security administrator will have the time to do this from a security report highlighting security risks. He will have to base his decision on the vulnerability description, and we have done so as well.

Apparition of a “Transfer” activity. Many vulnerability descriptions indicate that the vulnerability can be exploited by sending a message from a remote location.

The *Write* activity does not fully reflect this, as mail servers also exchange mail with other outside servers. In our use case, we had not taken this dimension into account originally. This has led to the creation of the *Transfer* activity. With respect to server-side threats, *Write* and *Transfer* activities are very close ; client-side threats (as shown for example in *CVE-2003-0007*) do imply that the dialog is between a registered user and an internal server, and do not implicate the *Transfer* activity.

Specific handling of low-level threats. Network / transport-level threats have a much broader impact than the ones at the higher layers. The only example of low level threat in table 6.7, synflood *CVE-1999-0116*, can be carried out regardless of the role, activity or view, provided that all subjects can inject traffic at the transport layer. Threats related to traffic injection (land, ping-of-death, ...) are difficult to include in the model unless specific network-level access-control policies are in place (such as authentication of DHCP requests or network level policy enforcement such as DHCP switching as practiced by Ungoliant³).

Table 6.7 also includes an information-level threat, the password guessing activity. This attack occurs against the *Access* activity, through any of the mail clients, against any of the mail servers, because the synchronization mechanism replicates the same account information throughout all channels.

6.4 Conclusion

In this chapter, we discussed how one can map policy instances onto alerts reported by intrusion detection system.

Our approach to response strategy takes into account (1) the direction of response, distinguishing attacker-centric response from victim-centric response, and client-side response from server-side response, (2) the target layer of response, allowing possibly fine-grained response at the information level, or at the service level, or at the network / transport level, and (3) the scale of response, acting on specific concrete entities or more generally on whole abstract entities.

We classified information to consider in three categories: (1) those available in alerts, (2) those available from model, and (3) those which can be deduced from alerts. We show in particular that an analysis of the vulnerability references reveals valuable information regarding strategy. Such an analysis is currently realized by the security operator to orient response strategy and define the appropriate mappings. We shall provide further details about strategy (especially mappings) through our implementation case study in the next chapter.

³<http://ungoliant.sourceforge.net/>

Chapter 7

Implementation

Contents

7.1 Introduction	109
7.2 Implementation case study	110
7.2.1 Modified email use case	110
7.2.2 Definition of the security policy	111
7.3 Policy Instantiation Engine	113
7.3.1 Objectives	114
7.3.2 Prolog response engine	114
7.3.3 Perl sequencer	120
7.3.4 Experiment: injecting a set of alerts	122
7.4 Policy Decision Point	127
7.4.1 Objectives	127
7.4.2 Considered countermeasures	127
7.4.3 Policy enforcement	128
7.5 Conclusion	131

7.1 Introduction

In this chapter, we present a prototype of a policy-based response system, based on the requirements formulated in Chapters 3, 4, 6. Figure 7.1 represents the implementation workflow, the ACE being considered as a black box reporting IDMEF alerts.

We first expose the case study considered for implementation purpose. This case study is not exactly similar to the model provided in Chapter 4. It is a simplified model which we explain in a first section, and which is very similar to the one presented in [35, 33, 34]. The core of the implementation work relies on the development of the PIE, in a second section, as a proof-of-concept of the presented idea. We thus propose a prototype based on a Perl sequencer which deals with a Prolog engine to translate alerts and additional contextual data into policy instances to enforce for threat response. In a

third section, we discuss the specification and implementation of a PDP, which should be able to translate policy instances reported by the PIE into configurations to push to the PEPs according to local strategy. In this section, we provide examples of PEPs and show how actual policy enforcement can be achieved.

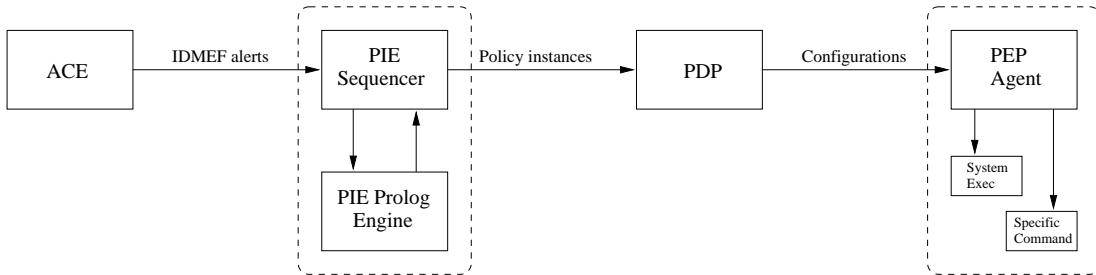


Figure 7.1: Implementation workflow

7.2 Implementation case study

7.2.1 Modified email use case

We consider here a generic architecture built upon three channels for accessing mail, namely the classic pop and imap protocols with their application of choice and outlook using the proprietary exchange protocol. As shown in figure 7.2, two UNIX processes (popd and imapd) and one Windows process (exchange.exe) are hosted on the same server. This is not an inconsistency, since with the help of virtualization technology, running two operating systems simultaneously on a same host is relevant. Contrary to the previous use case, we assume here that the three services are rigorously similar regarding mail, *i.e.* each one includes a copy of the mailbox repository. Hence, all three are kept synchronous, and changes in the same account using one of the access mechanisms are immediately seen using the others.

The *organization* is here is simply the *pie*, *i.e.* we consider that the defined policy is relevant to the PIE, which is the global entity responsible for the policy evaluation and instantiation. We degrade the previously presented model and do not explicitly take into account the three layers model, for the sake of simplicity. However, these three layers are found implicitly in the application, through the deployment of sample countermeasures at the three considered layers.

Or-BAC entities are modeled as follows:

- **Roles** are limited to the *mail_user* role and the *mail_station* role, *i.e.* the users accessing mail, and the hosts which are used for this purpose. Subjects include thus *alice* and *charlie*, as well as *pc-alice*, and *pc-charlie*, which represent the hosts used by Alice and Charlie to access mail.
- **Views** are limited to the *mail_box* and the *mail_server* view, *i.e.* respectively the files which represent the mailbox repositories, and the servers which host these

files and mail services. The single server object *mel1* provides the three mail services, and hosts mailboxes */var/spool/mail/alice* and */var/spool/mail/charlie*. Modeling files which actually materialize mail repositories provide another adjustment variable regarding access control.

- **Activities** are mainly dealing with the *read_mail* activity. Declined among the three channels through activities *read_pop*, *read_imap* and *read_exchange*, it results in network / transport-level actions, like *tcp/110* and *tcp/143*, and in service-level actions, like */etc/popd* and */etc/imapd*. We also consider information-level actions, namely file permissions. We introduce the *write_mail* activity, to distinguish *read* and *write* access permissions to files, but we do not consider network / transport and service-level actions for the *write_mail* activity, for the sake of simplicity. This should however be added for actual deployment in realistic settings, similarly as for *read_mail* sub-activities.
- **Contexts** are again divided into *operational*, *threat* and *minimal* contexts.

The objective of the use case is to show how the system reacts to a few sample alerts, managing multiple paths to resources so that minimal requirements are always met.

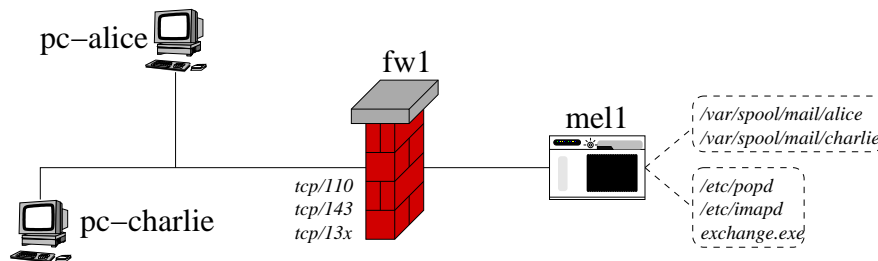


Figure 7.2: Implementation: case study layout

7.2.2 Definition of the security policy

Listing 7.1: Implementation: email access control policy

```
sr(permission, pie, mail_station, read_mail, mail_server, nominal).
sr(permission, pie, mail_user, read_mail, mailbox, nominal).
sr(permission, pie, mail_user, write_mail, mailbox, nominal).
sr(prohibition, pie, mail_user, read_mail, mailbox, n(mymailbox)).
sr(prohibition, pie, mail_user, write_mail, mailbox, n(mymailbox)).

sr(prohibition, pie, mail_station, read_pop, mail_server, pop_attack).
sr(prohibition, pie, mail_station, read_imap, mail_server, imap_attack).
sr(prohibition, pie, mail_station, read_exchange, mail_server, exchange_attack).
sr(prohibition, pie, mail_user, write_mail, mailbox, mail_threat).

sr(permission, pie, mail_station, read_pop, mail_server, &(minimal_mail, working_hours)).
```

The security policy is shown in listing 7.1. The policy states that it should always exist a way to read mail during working hours, but not necessarily on non-working hours. To avoid the case for which the system would close all possible paths to mail

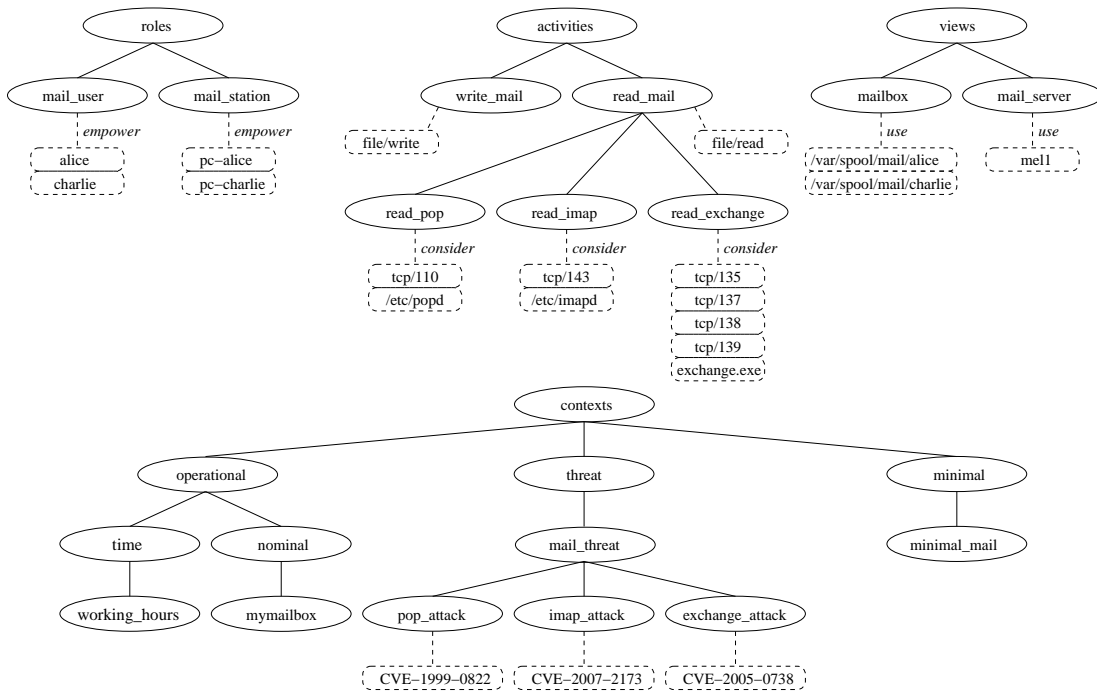


Figure 7.3: Implementation: Or-BAC abstract and concrete entities

(self-inflicted denial-of-service), we define an exception with a high-level priority. This is realized thanks to the *minimal_mail* context, permitting pop reading in this case study. Note that choosing the path to re-open is very subjective and left to the expertise of the security officer. In particular, it may depend on the probability for an application to be successfully attacked. Since, the “Exchange” keyword provides a large amount of results in the NVD, it is a good choice to avoid Exchange as a minimal service. Indeed, Exchange is widely used and is thus very likely to interest attackers. Then, a choice has to be made between pop and imap. We arbitrarily suppose in the case study that our implementation of pop is less sensitive to attacks than our implementation of imap.

This security policy then specifies that any attack against one of the email access mechanisms invalidates the access mechanism being attacked, and that by default, mail stations have access to all mechanisms to read mail. This simple expression is obtained by taking into account that each rule also applies to children in the graphs. We also consider that some mail threats may imply that a mail user is physically prevented from reading or writing his mailbox, *i.e.* his reading or writing permission on his mail repository file is invalidated. The *write_mail* activity is modeled to illustrate such information-level response, without any minimal requirement.

Note that this concise expression is generic and adaptable to multiple physical architectures. If we had multiple mail servers spread per location instead of a centralized mail server farm, we would express the same policy. However, we would change the deployment strategy at the PDP level and have a different list of PEPs.

Listing 7.2 defines the *hold* predicates relevant to the described policy. The temporal

Listing 7.2: Implementation: context definitions

```

hold(pie, Subject, Action, Object, Context) :-
    alert(CreateTime, Source, Target, Classification),
    map_context(Classification, Context),
    map_syntax(Source, Target, RawSubject, RawAction, RawObject),
    map_enrichment(RawSubject, RawAction, RawObject, EnrSubject, EnrAction, EnrObject),
    map_strategy(EnrSubject, EnrAction, EnrObject, Subject, Action, Object).

hold(pie, Subject, _, Object, minimal_mail) :-
    hold(pie, Subject, _, Object, pop_attack),
    hold(pie, Subject, _, Object, imap_attack),
    hold(pie, Subject, _, Object, exchange_attack).

hold(pie, _, _, working_hours) :-
    globalclock(DayClock, TimeClock),
    TimeClock >= '07:00:00',
    TimeClock < '20:00:00',
    DayClock != 'saturday',
    DayClock != 'sunday'.

hold(pie, _, _, nominal).

hold(pie, Subject, Action, Object, mymailbox) :-
    empower(pie, Subject, mail_user),
    (consider(pie, Action, read_mail);
     consider(pie, Action, write_mail)),
    use(pie, Object, mailbox),
    active_directory_entry(Subject, Object).

```

working_hours context is modeled in a straightforward way, as is the *nominal* context. The *mymailbox* context ensures that there exists an entry between subject (mail user) and object (mailbox) in active directory. The context *minimal_mail* is active when all three email access mechanisms are attacked. Hence, during working hours, when the $\&(minimal_mail, working_hours)$ context is active, the policy expresses that the pop access is re-opened ensuring continued availability of email information.

7.3 Policy Instantiation Engine

In this section, we give a sample implementation of the PIE, distinguishing its two main functionalities, namely the TRCE and the PCE. Figure 7.4 illustrates the workflow allowing the mapping from alerts to *hold* facts relevant for adequate response to assessed threat. Note that this workflow does not reflect a generic specification of the TRCE, but just the current implementation, including *syntactic mapping*, *enrichment* and *strategy application*.

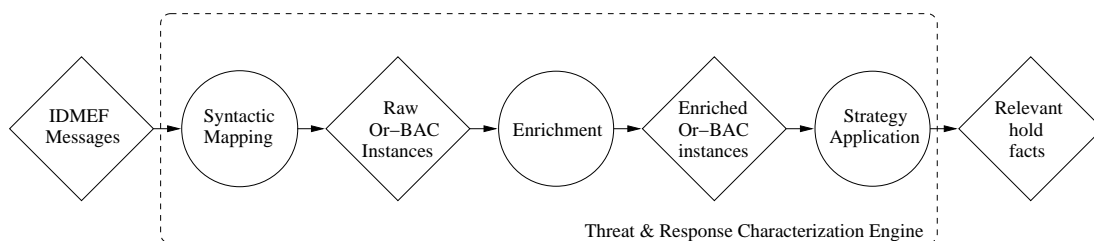


Figure 7.4: Implementation: Threat & Response Characterization Engine workflow

In the following, we first recap the objectives of the PIE. Then, we give elements

about current implementation, first regarding the TRCE functionality, and then, dealing with the PCE functionality.

7.3.1 Objectives

- **Implement the Or-BAC model with threat context definition.** The first objective is to implement the Or-BAC model according to its specification, which one can refer in [62]. The prototype must also provide extensions regarding threat contexts and context composition, according to requirements formulated in Chapter 4.
- **Read IDMEF alerts to trigger threat contexts.** The prototype must be able to read IDMEF alerts and consider all available information. Information have to be mapped in a phase of syntactic mapping on *subject, action, object*.
- **Enrich information considering policy definition.** The prototype must consider the policy definition to deduce some data which would not be available in alerts and possibly add new relevant (*subject, action, object*) triples.
- **Ensure adequate response to threat thanks to relevant strategy.** Information reported by syntactic mapping and enrichment do not necessarily reflect adequate response. A next step consists in evaluating whether or not information should be degraded to provide best relevant response.
- **Ensure dynamic policy compilation.** Activation of threat contexts must dynamically trigger policy compilation to generate new policy instances to enforce.
- **Manage threat context deactivation.** Contexts activated considering incoming alerts have to be deactivated with respect to alert lifetime strategy. Retracting alerts from the system when they are not relevant anymore automatically deactivates associated contexts, and thus provides adequate policy instances to enforce.
- **Ensure minimal requirements.** The prototype must include definition of specific contexts ensuring the fulfillment of minimal requirements, especially regarding availability of services and resources.
- **Conflict resolution and automated priorities assessment.** The prototype should be able to solve conflicts at the abstract level. For this purpose, it should implement criticity and specificity strategies to automatically assess rule priorities.

7.3.2 Prolog response engine

7.3.2.1 Modeling Or-BAC concrete and abstract entities

We consider here the description of structured objects of Chapter 4, simplified and adapted to the implementation case study. We model hosts and users as *subjects*, to describe mail stations and mail users. We model services and processes as *actions*, to

describe port numbers and instance of service applications. Finally, we model hosts and files as *objects*, to describe mail servers and mailboxes.

Or-BAC abstract entities are defined using the *role*, *activity* and *view* predicates. Hierarchies are obtained thanks to *sub_role*, *sub_activity* and *sub_view* predicates.

Abstract-to-concrete assignments are declared through *empower*, *consider* and *use* predicates. Listing 7.3 provides examples with respect to structured objects previously introduced.

Listing 7.3: Implementation: empower / consider / use examples

```
empower(pie, [element('Subject', [], [element('User', [],
[element('name', [], ['alice']), element('uid', [], [''])])])]), mail_user).

empower(pie, [element('Subject', [], [element('User', [],
[element('name', [], ['charlie']), element('uid', [], [''])])])]), mail_user).

empower(pie, [element('Subject', [], [element('Host', [],
[element('name', [], ['pc-alice']), element('address', [], ['192.168.1.12']),
element('netmask', [], ['255.255.255.0'])])])]), mail_station).

empower(pie, [element('Subject', [], [element('Host', [],
[element('name', [], ['pc-charlie']), element('address', [], ['192.168.1.13']),
element('netmask', [], ['255.255.255.0'])])])]), mail_station).

consider(pie, [element('Action', [], [element('Service', [],
[element('name', [], [pop3]), element('port', [], ['110/tcp'])])])]), read_pop).

consider(pie, [element('Action', [], [element('Process', [],
[element('name', [], [popd]), element('pid', [], ['']),
element('path', [], ['/etc/popd'])])])]), read_pop).

consider(pie, [element('Action', [], [element('Fileright', [],
[element('name', [], [read])])])]), read_mail).

consider(pie, [element('Action', [], [element('Fileright', [],
[element('name', [], [write])])])]), write_mail).

use(pie, [element('Object', [], [element('Host', [],
[element('name', [], [mell]), element('address', [], ['192.168.2.50']),
element('netmask', [], ['255.255.255.0'])])])]), mail_server).

use(pie, [element('Object', [], [element('File', [],
[element('name', [], [charlie]), element('path', [], ['/var/spool/mail/charlie']),
element('host', [], [mell])])])]), mailbox).
```

7.3.2.2 Parsing alerts

Reading IDMEF files is realized through the `load_xml_file` function of Swi-Prolog. It interprets an XML file as a recursive list of elements, as explained in Section 5.8. The `member` function retrieves any element `Elem` from a given list `List`. For instance, the `Node` sub-class of a given IDMEF alert is accessed by function `map_node_name`, which allows to get node name, and function `map_node_address`, which provides the category and value of address, as shown in listing 7.4.

Listing 7.4: Implementation: parsing IDMEF alerts

```
N = [element('Node', [category='unknown'], [element('name', [], [mell]),
element('Address', [category='ipv4-addr'], [element('address', [],
['192.168.1.50'])])])].

map_node_name(N, Name) :-
    member(element(name, [], [Name]), N).

map_node_address(N, AddressCategory, Address) :-
    member(element('Address', AddressAtt, AddressClass), N),
    member(category=AddressCategory, AddressAtt),
    member(element(address, [], [Address]), AddressClass).
```

A super-function called `map_idmeffile` gets all main classes of an IDMEF message, namely *CreateTime*, *Source*, *Target*, *Classification*, *Assessment*, *AdditionalData* and *CorrelationAlert*. Each of these main classes is readable through the use of more fine-grained mapping functions, alike those presented in listing 7.4.

In the prototype, *subjects* are mapped onto *IDMEF.Source* class and *objects* onto *IDMEF.Target* class. This fits a wide majority of the alerts reported by intrusion detection systems, and assume thus that the ACE reports alerts complying with this assumption.

7.3.2.3 Syntactic mapping

Syntactic mapping phase extracts (*subject, action, object*) triples from IDMEF messages, *i.e.* raw Or-BAC instances. According to the case study, the implementation considers three sample syntactic mappings, given by listing 7.5.

Listing 7.5: Implementation: syntactic mapping examples

```
map_syntax(Source, Target, Classification, Subject, Action, Object) :-
    map_subject_node(Source, Subject),
    map_action_service(Target, Action),
    map_object_node(Target, Object).

map_syntax(Source, Target, Classification, Subject, Action, Object) :-
    map_subject_node(Source, Subject),
    map_action_process(Target, Action),
    map_object_node(Target, Object).

map_syntax(Source, Target, Classification, Subject, Action, Object) :-
    map_subject_fileuser(Target, Subject),
    map_action_fileright(Target, Action),
    map_object_file(Target, Object).
```

7.3.2.4 Enrichment

Enrichment functionalities are implemented through the `map_enrichment` function, defined as follows:

1. Missing information is enriched from policy definition, *i.e.* *subjects*, *actions* and *objects* which are declared in the policy. This completes non-fully qualified entities into alerts.
 - Subjects are qualified in alerts by an IP address and/or a DNS name. If both information are not provided, the policy definition completes the missing data.
 - Actions are generally qualified into alerts by UDP/TCP ports, or possibly the associated protocol name. Port number and name are equivalent and related through the relevant `/etc/services` entry. Thus, if one is missing, completion is possible. Processes are also described by associations of name and path. The pid is not considered in this application, policy enforcement being ensured through the knowledge of the process path.
 - Objects are managed in the implementation exactly in the same way as subjects for host objects. For file objects, we consider that alerts report full paths, to avoid confusion between potential files with a same name.

Completion is not always strictly necessary regarding enforcement, but the policy compilation deals with matching of subjects, actions and objects that have been fully qualified at the policy definition step, and enrichment facilitates further matching between subjects, actions and objects obtained from alerts with those declared in the policy definition.

2. Additional information may be deduced by association between enforcement levels. For instance, network / transport-level action *tcp/110* is linked with service-level action */etc/popd*. If only one of these two actions is found in the alert, the response system is able to instantiate as well the associated action. This does not mean that enforcement will deal with both actions, but this lets this possibility at the strategy application step.

7.3.2.5 Strategic mapping

We consider four strategy parameters: (1) response classes, (2) target layers, (3) spoofed sources and (4) alert severity.

1. **Response classes.** Response classes allow part of the determination of direction and scale of response. Depending on the references, the `response_class` predicate defines whether actual response should consider *subject*, *action* and *object* as reported by the enrichment phase, or to degrade one or more of them to focus on attacker or victim, and to enlarge scale of response. The decision here is binary, since degrading entities is realized by replacing the explicit entity by the Prolog variable `'_'`, which results in the consideration of any entity matching a given security rule at the abstract level. Response classes may be classified in eight categories, namely `'_'`, `'s'`, `'a'`, `'o'`, `'sa'`, `'ao'`, `'so'`, and `'sao'`. `'_'` means that there is no defined constraint on subject, action and object, whereas `'sao'` means that the *hold* fact is instantiated precisely with the subject, action and object provided by enrichment. Response classes also deal with response level, as mentioned in the following.

Listing 7.6 gives two examples of response classes, using `'x'` to qualify elements which are not considered in the strategy. For instance, in listing 7.6, the first statement recommends a `'sao'` strategy, whereas the second statement recommends an `'ao'` strategy.

Listing 7.6: Response classes: examples

```
response_class(CVE-1999-0822,s,a,o).
response_class(CVE-2005-0738,x,a,o).
```

2. **Target layers.** Function `response_layer` analyzes subjects, actions and objects to assign the target layer of response, either network, service or information. This allows the `response_class` function to assign different classes of response considering the layer.

Listing 7.7 gives examples of target layers assessment and illustrates the possibility to define different classes of response at different layers. Subjects, actions and

objects are assigned layers. For instance, an IP address (subject or object) and a port number (action) are considered at the network layer. A process (action) is considered at the service layer. A user (subject), a fileright (action) and a file (object) are considered at the information layer. Regarding response classes, we observe here that *CVE-1999-0822* is considered differently at the network and service layer. The two last statements recommend a 'sao' strategy at the network layer, and an 'ao' strategy at the service layer. Since no *response_class* fact is defined at the information layer, the system will consider here response only at the network and service layers.

Listing 7.7: Response levels: examples

```

response_layer (Subject , Action , Object , network) :-
    subject_layer (Subject , network) ,
    action_layer (Action , network) ,
    object_layer (Object , network) .

response_layer (Subject , Action , Object , service) :-
    subject_layer (Subject , network) ,
    action_layer (Action , service) ,
    object_layer (Object , network) .

response_layer (Subject , Action , Object , information) :-
    subject_layer (Subject , information) ,
    action_layer (Action , information) ,
    object_layer (Object , information) .

response_class (CVE-1999-0822,s,a,o,network) .
response_class (CVE-1999-0822,x,a,o,service) .

```

3. **Spoofer source.** Considering the 'spoofed' IDMEF parameter allows us to degrade strategy regarding subject. In such a case, instead of responding specifically on subject reported by enrichment, the strategy considers '_' as subject, so that every potentially threatening subject is taken into account in the response. Indeed, a spoofed source often means a host which has been compromised by the actual attacker to realize his attack. In such a case, we consider that other hosts may be compromised as well, and we prefer for instance to deploy filtering rules considering every potentially threatening source instead of only dealing with the current attack.
4. **Alert severity.** It is considered to enlarge scale of response in our system. In particular, in our experiments, when severity is set to *high*, response is forced to all matching subjects (subject '_'), and not only to a specific subject, since we consider that the consequences of the corresponding attack are likely to be more serious. Enlarging scale of response is thus another way to take into account other potentially threatening sources.

7.3.2.6 Context activation

Listing 7.8 shows how we manage activation of *hold* facts related to threat contexts. Note that the *map_context* function in our implementation simply associates the vulnerability reference found in the *Classification* class with the corresponding threat context. However, as mentioned in 4.3.1.1, context mapping could take into account

additional parameters if necessary, including targeted service for instance. The last three functions evaluate composed context activation¹.

Listing 7.8: Implementation: strategy

```

hold(pie, Subject, Action, Object, Context) :-
    alert(MsgId, CreateTime, Source, Spoofed, Target, Decoy, Classification, Assessment),
    map_context(Classification, Context),
    map_syntax(Source, Target, Classification, RawSubject, RawAction, RawObject),
    map_enrichment(RawSubject, RawAction, RawObject, EnrSubject, EnrAction, EnrObject),
    map_strategy(EnrSubject, EnrAction, EnrObject, Source, Spoofed, Target, Decoy,
                Classification, Assessment, Subject, Action, Object).

hold(A, B, C, D, &(E1, E2)) :-
    hold(A, B, C, D, E1),
    hold(A, B, C, D, E2).

hold(A, B, C, D, v(E1, E2)) :-
    hold(A, B, C, D, E1),
    hold(A, B, C, D, E2).

hold(A, B, C, D, n(E)) :-
    not(hold(A, B, C, D, E)).

```

7.3.2.7 Conflict resolution and priority assessment

Priority assessment strategies (S1 and S2) are implemented through the definition of the *levelSup* predicate, according to 4.3.4. The *levelSup* predicate considers potentially conflicting security rules, ensuring thus conflict resolution at the abstract level. Conflicts are then avoided at the concrete level through the use of *higher_permitted* and *higher_prohibited* predicates, as shown by listing 7.9. The former indicates that there exists a higher prioritized concrete permission (*is_permittedL*), and similarly for the latter with prohibitions (*is_prohibited*). These predicates are then used for policy instantiation, as shown in the following section.

Listing 7.9: Implementation: Priority assessment and conflict resolution

```

higher_permitted(Subject, Action, Object, PriorityLevel1) :-
    is_permittedL(Subject, Action, Object, PriorityLevel2),
    levelSup(PriorityLevel2, PriorityLevel1).

higher_prohibited(Subject, Action, Object, PriorityLevel1) :-
    is_prohibitedL(Subject, Action, Object, PriorityLevel2),
    levelSup(PriorityLevel2, PriorityLevel1).

```

7.3.2.8 Policy instantiation

Policy instantiation (or compilation) is the last step assumed by the PIE in the response process. Concrete permissions and prohibitions are instantiated according to listing 7.10. According to previously assessed priorities in *security_ruleL* statements and to currently active *hold* statements, concrete prioritized policy instances (*is_permittedL* and *is_prohibitedL*) are derived through the two first statements. Actual concrete authorizations are obtained thanks to the two last statements (*is_permitted* and *is_prohibited*), which ensure that a permission is instantiated only if there does not exist any prohibition with a higher priority, and respectively, a prohibition is instantiated only if there does not exist any permission of higher priority.

¹Note that we provide here the function associated with context disjunction, which is not used in this use case since we choose to segment security rules according to disjunction to facilitate automated priority assessment and conflict resolution.

Listing 7.10: Implementation: Policy instantiation

```

is_permittedL(Subject, Action, Object, PriorityLevel) :-
    security_ruleL(permission, Org, Role, Activity, View, Context, PriorityLevel),
    empower(Org, Subject, Role),
    consider(Org, Action, Activity),
    use(Org, Object, View),
    hold(Org, Subject, Action, Object, Context).

is_prohibitedL(Subject, Action, Object, PriorityLevel) :-
    security_ruleL(prohibition, Org, Role, Activity, View, Context, PriorityLevel),
    empower(Org, Subject, Role),
    consider(Org, Action, Activity),
    use(Org, Object, View),
    hold(Org, Subject, Action, Object, Context).

is_permitted(Subject, Action, Object) :-
    is_permittedL(Subject, Action, Object, PriorityLevel),
    not(higherProhibited(Subject, Action, Object, PriorityLevel)).

is_prohibited(Subject, Action, Object) :-
    is_prohibitedL(Subject, Action, Object, PriorityLevel),
    not(higherPermitted(Subject, Action, Object, PriorityLevel)).

```

7.3.3 Perl sequencer

The PIE main program is a Perl sequencer which interfaces with the Prolog engine. We give here only a short description of the functionalities ensured by the sequencer:

1. Receive IDMEF alerts from the ACE, query the Prolog engine for alert parsing², and assert new alerts in the Prolog engine,
2. Compute alerts deathtimes and retract alerts from the Prolog engine when their lifetime is over,
3. Compile and export new policy instances on changes, *i.e.* considering asserted and retracted alerts, but also considering a synchronized clock for operational policy adapting, especially regarding temporal contexts,
4. Send policy instances to the PDP for translation and enforcement.

7.3.3.1 Formatting policy

Policy instances are exported in XML, relying on the structured entities resulting from policy compilation, namely subjects, actions and objects, associated to the type of the policy instances, namely permission or prohibition. An `export_policy` function is in charge of correctly formatting policy instances at the Prolog level, *i.e.* generating suitable tables of *elements*, so that the native Swi-Prolog `xml_write` function automatically writes tables of elements into policy instances similar to listing 7.11. The Perl sequencer sends such policy instances to the PDP, which extracts information useful for translation and enforcement.

²Note that alert parsing is realized through the use of Prolog in the prototype, but it could as well be ensured in the Perl program.

Listing 7.11: PIE output: sample policy instance

```
<PolicyInstance type="prohibition">
  <Subject>
    <Host>
      <name>pc-charlie</name>
      <address>192.168.1.13</address>
      <netmask>255.255.255.0</netmask>
    </Host>
  </Subject>
  <Action>
    <Service>
      <name>pop3</name>
      <port>tcp/110</port>
    </Service>
  </Action>
  <Object>
    <Host>
      <name>mel1</name>
      <address>192.168.2.50</address>
      <netmask>255.255.255.0</netmask>
    </Host>
  </Object>
</PolicyInstance>
```

7.3.3.2 Policy updates

The Perl sequencer is responsible for the temporal aspect of policy management. In particular, it decides when a new policy implementation must be applied, *i.e.* when new policy instances have to be sent to the PDP for enforcement in the PEPs. Policy compilation considers the whole policy, and not only policy changes due to current threat. In practice, re-deploying the whole policy - *i.e.* all policy instances including operational requirements - each time an alert is handled is impossible.

Mechanisms must be implemented in order to address the update issue. At least two solutions may be envisioned. First, the PIE may be aware of the current policy expression, and thus be able to deploy only the instances which differ in the new policy expression from the current one. Second, the PDP, as a local decisional entity, may be able to keep track of which configurations are currently enforced, and thus which one has to be changed to reflect the new policy expression. This is part of the future work.

Current implementation of the PIE updates the policy according to a whole IDMEF logfile, *i.e.* a new expression of the policy is provided considering all alerts included in this file, and considering alerts independently. We consider that dealing with each alert independently is a costly approach since we re-evaluate the whole policy at each time. This explains our preference to decrease the granularity at the file level. The security officer may be able to choose the maximum size of such a logfile, may it be a maximum number of alerts or a maximum time window depending on the desired reactivity.

The PIE is also in charge of retracting alerts from the Prolog engine each time an alert lifetime expires. This automatically triggers a re-compilation of the policy, so that corresponding threat contexts, and thus policy instances related to threat response, may be relaxed. Note that synchronous updates of the policy implementation are also necessary, especially to take into account temporal context changes.

7.3.4 Experiment: injecting a set of alerts

We now discuss a simple experiment consisting in providing to the PIE three alerts corresponding to the three provided services: POP, IMAP, and Exchange.

7.3.4.1 Considered alerts

In this case study, we manage response through the deployment of three simple countermeasures: filtering, process start / stop management and file access rights management. As presented in figure 7.2, a network firewall filters traffic between mail stations and the mail server. A PEP agent is running on the firewall host, as well as on the mail server, to run the commands required by the PDP to enforce the new policy. On the mail server, the agent is able to start and stop mail services and to reconfigure access rights of the mail repository files. Reconfiguring file rights in this application is enforced on UNIX mail repositories (those used by pop and imap services). We assume that the process in charge of synchronizing the repositories also updates permissions to Exchange database with respect to rights enforced on UNIX files.

Let us consider a set of sample vulnerabilities, namely references *CVE-1999-0822*, *CVE-2006-1255* and *CVE-2005-0738*, respectively threatening POP3, IMAP and Exchange services. We respectively associate these vulnerabilities to *Alert1*, *Alert2* and *Alert3* to facilitate further discussion. We consider that these vulnerabilities apply to the services considered in this use case, that is configurations are actually affected by these vulnerabilities.

Alert1. CVE-1999-0822 is a buffer overflow attack towards the Qpopper POP3 service, which allows remote root access via the POP3/auth command. Detection can for instance be achieved using a regular expression in a Snort rule matching the POP3/auth command with a too long argument in the content of the packet payload. Since we do not consider such fine-grained commands in the implementation case study, we cannot provide a countermeasure centered on POP3/auth specifically. We thus choose to close the POP service when such an alert is incoming. This means both that a firewall rule may be deployed to block destination port tcp/110, and that the process may be stopped, according to the corresponding script (/etc/popd).

Alert2. CVE-2007-2173 is an injection attack towards the Courier-Imap service, which allows remote attackers to execute arbitrary commands on the server. Specially crafted login credentials have to be sent to the server to exploit this vulnerability. Detection can thus be achieved using a Snort signature detecting the corresponding specificities in the packet payload. For this attack, we provide a similar countermeasure than for POP3, *i.e.* we choose to close the IMAP service (/etc/imapd) and to block IMAP traffic on the firewall (port tcp/143).

Alert3. CVE-2005-0738 is a stack overflow attack in Microsoft Exchange Server 2003, which allows users to cause a denial-of-service by deleting or moving a folder with deeply nested subfolders. Detection is mostly to be achieved through the observation of the

depth of user directories, which should not exceed a given number of sub-folders. We will consider two sample countermeasures in such a case, depending on the strategy defined by the security officer. The first countermeasure consists in preventing the incriminated user write access to his mailbox, by changing permissions of the mail repository file. The second countermeasure consists in closing the Exchange service, similarly as in the two precedent examples. The first envisioned response is more fine-grained, since it is attacker-centric, although actual response is enforced on the server. This may not be sufficient in some cases, since it only addresses the attack issue, but not exactly the actual threat, which is “*if somebody has completed this attack, maybe somebody else will be able to do so*”. Depending on the strategy chosen by the security officer for such an attack, the second countermeasure may be preferred, so that the service will be temporarily isolated, which allows the security officer to apply the necessary long-term solutions (*e.g.* patching the application).

For all three alerts, we consider that *pc-charlie* is the attacker reported in the IDMEF *Source.Node* class and that *mel1* is the victim reported in the IDMEF *Target.Node* class. The activity is provided by the *Target.Service* class. Alert3 provides information-level data, namely a user and a file (mailbox), found in *Target.File* class.

7.3.4.2 Description of the experiment

Temporal considerations. Alerts are simultaneously sent through a common IDMEF logfile sent by the ACE at time t , according to figure 7.5.

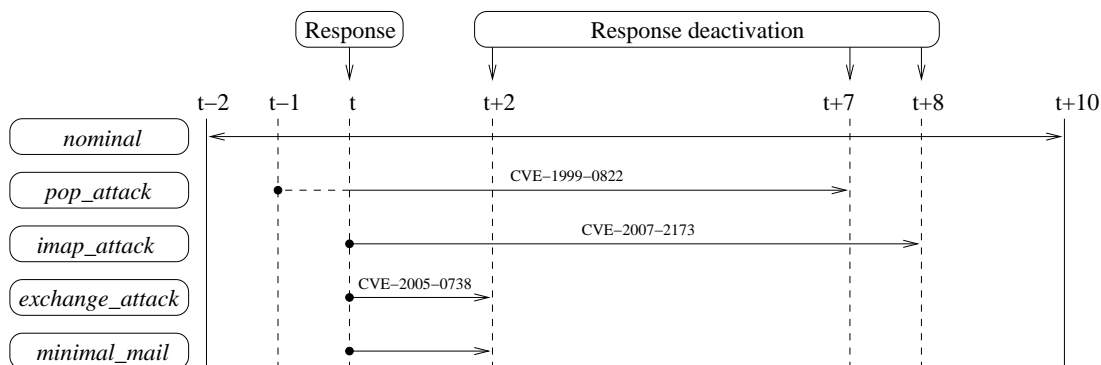


Figure 7.5: Experimentation: sequence of alerts and associated contexts

Alert lifetimes are computed according to the table presented in Section 4.3. We recap these durations in minutes in table 7.1, respectively to the vulnerability references.

Figure 7.5 shows contexts activated for our case study policy, and threat contexts deactivation, depending on alert lifetime computation. The *nominal* context is always active, according to its definition. The *minimal_mail* context is active between t and $t+2$, according to the policy definition, which indicates its activation when all three threat contexts are active. Context *working_hours* does not appear explicitly

Reference	Severity	Type	Context	Lifetime
CVE-1999-0822	High	Admin	pop_attack	8
CVE-2007-2173	High	Admin	imap_attack	8
CVE-2005-0738	Low	DoS	exchange_attack	2

Table 7.1: Considered alert lifetimes

in the experiment. In fact, we consider that the experiment is realized during working hours. Thus, the security rule associated with the minimal context is active when *minimal_mail* is active. Depending on considered subjects, actions and objects, contexts are here only likely to be activated, not necessarily actually activated. They are for instance conjointly activated in our case study for subject *pc-charlie*, action *tcp/110* and object *mel1*.

Note that a countermeasure lifetime does not necessarily correspond to the actual alert lifetime, since alert deathtime is calculated considering the time the alert was created by the ACE. Countermeasure lifetime thus depends on the delay between the createtime and the time alert is processed by the PIE. This is depicted by a portion of dotted line in the figure. This explains why context *pop_attack*, which lasts as long as context *imap_attack*, expires in fact a minute before.

Response strategy. The different entities involved in the case study are listed in table 7.2, distinguishing the three phases considered in the PIE process, namely *syntactic mappings*, *enrichment* and *strategy application*. We refer to port numbers related to Microsoft Exchange as “tcp/13x” for the sake of simplicity.

Three service-level triples are provided by the enrichment phase. On this purpose, relations are defined between network-level actions, like *tcp/110*, and the corresponding service-level actions, like */etc/popd*, through the *linkEntity* predicate.

Strategy application results in the adjunction of triples linked with the enlargement of service-level response to all mail stations (*pc-charlie* and *pc-alice*). This corresponds to the use of the ‘ao’ response class at the service level for CVE-2005-0738, which does not constrain subjects. This strategy isolates the threatened service (Exchange) from any potential attacker, *i.e.* any subject empowered in the role *mail_station*. This choice is also due to ensure enforcement consistency. Stopping a service process for a particular user results in stopping the service for all users. Modeling an ‘ao’ strategy is thus a way to render this limitation transparent by taking it into account in the model³.

Listing 7.12 lists the defined response classes to implement response strategy. This listing conforms to the aforementioned strategy information, *i.e.* network-level countermeasures consider a ‘sao’ strategy, and service-level degrades the strategy to ‘ao’ to consider all subjects. Information-level response is considered here only for *Alert3* and is said to be ‘sao’.

³To provide finer-grained service-level responses, one could envision the deployment of multi-threaded service processes, *i.e.* one instance of the considered process per user.

Phase	Subject	Action	Object	Comment
Syntactic mappings	pc-charlie	tcp/110	mel1	Found in Alert1 (CVE-1999-0822).
	pc-charlie	tcp/143	mel1	Found in Alert2 (CVE-2007-2173).
	pc-charlie	tcp/13x	mel1	Found in Alert3 (CVE-2005-0738).
	charlie	write	/var/spool/mail/charlie	Found in Alert3 (CVE-2005-0738).
Enrichment	pc-charlie	tcp/110	mel1	
	pc-charlie	tcp/143	mel1	
	pc-charlie	tcp/13x	mel1	
	charlie	write	/var/spool/mail/charlie	
	pc-charlie	/etc/popd	mel1	Process related to tcp/110 port.
	pc-charlie	/etc/imapd	mel1	Process related to tcp/143 port.
pc-charlie	exchange.exe	mel1	Process related to tcp/13x port.	
Strategy application	pc-charlie	tcp/110	mel1	
	pc-charlie	tcp/143	mel1	
	pc-charlie	tcp/13x	mel1	
	charlie	write	/var/spool/mail/charlie	
	pc-charlie	/etc/popd	mel1	
	pc-charlie	/etc/imapd	mel1	
	pc-charlie	exchange.exe	mel1	
	pc-alice	/etc/popd	mel1	Added by strategy.
	pc-alice	/etc/imapd	mel1	Added by strategy.
pc-alice	exchange.exe	mel1	Added by strategy.	

Table 7.2: Experiment: concrete entities at the different phases of the process

Listing 7.12: Experiment: response classes

```

response__class('CVE-1999-0822',s,a,o,network).
response__class('CVE-1999-0822',x,a,o,service).

response__class('CVE-2007-2173',s,a,o,network).
response__class('CVE-2007-2173',x,a,o,service).

response__class('CVE-2005-0738',s,a,o,network).
response__class('CVE-2005-0738',x,a,o,service).
response__class('CVE-2005-0738',s,a,o,information).

```

Context activation. Figure 7.5 is a restricted vision of context activation. Contexts are associated with $(subject, action, object)$ triples. This means in particular that the *minimal_context*, according to its definition, is not activated provided the three attack contexts are active, but provided that they are active for the same subject and object. Figure 7.5 thus applies to subject *pc-charlie* and object *mel1*. We give in listing 7.13 a complete list of activated *hold* facts dealing with threat. The *minimal_mail* context is active according to figure 7.5, for subject *pc-charlie* and object *mel1*, but also for subject *pc-alice* and object *mel1*, due to strategy, which adds the three last *hold* statements. This illustrates the side-effects a response may have on legitimate users (depending on scale of response and PEP capabilities) and the necessity of the *minimal_mail* context. In our case study, the minimal policy re-opens the pop path for both *pc-alice* and *pc-charlie*, even though *pc-charlie* appears to be the source of all attacks, because the minimal requirement guarantees access to all users regardless of the threat. Our system considers potential negative side-effects of automated response and cannot be absolutely sure that Charlie is the actual (or deliberate) attacker. Although IDMEF provides a *spoofed* parameter, the ACE may not always be able to report that an external attacker has spoofed Charlie’s station to attack all mail services. We do not want minimal requirements which close mail access to Charlie in such a case.

Note however that in another application, with different minimal requirements, one may choose to keep a service process on, but to block *pc-charlie* for all services at the firewall level. This is a matter of strategy, considering impacts, costs and potential side-effects. This evaluation is left to the consideration of the security administrator.

Listing 7.13: Experiment: threat-related hold facts

```
hold(pie,pc-charlie,tcp/110,mell,pop_attack).
hold(pie,pc-charlie,tcp/143,mell,imap_attack).
hold(pie,pc-charlie,tcp/13x,mell,exchange_attack).
hold(pie,charlie,write,/var/spool/mail/charlie,exchange_attack).
hold(pie,pc-charlie,/etc/popd,mell,pop_attack).
hold(pie,pc-charlie,/etc/imapd,mell,imap_attack).
hold(pie,pc-charlie,exchange.exe,mell,exchange_attack).
hold(pie,pc-alice,/etc/popd,mell,pop_attack).
hold(pie,pc-alice,/etc/imapd,mell,imap_attack).
hold(pie,pc-alice,exchange.exe,mell,exchange_attack).
```

7.3.4.3 Results

We now give the resulting concrete permissions and prohibitions which are instantiated by the PIE. Note that a complete policy re-evaluation is necessary each time a set of alerts is provided, and each time an alert lifetime expires. This explains the segmentation into multiple temporal intervals. Concrete permissions are represented by “Pe” in the table, whereas concrete prohibitions are identified by “Pr”. Permissions which are activated with the *minimal_mail* context are distinguished through the bold italic notation “*Pe*”. Table 7.3 shows the resulting network-level policy instances. Table 7.4 deals with the service level, and table 7.5 with the information level.

Subject	Action	Object	t-2:t-1	t-1:t	t:t+2	t+2:t+7	t+7:t+8	t+8:t+10
pc-alice	tcp/110	mell	Pe	Pe	Pe	Pe	Pe	Pe
pc-alice	tcp/143	mell	Pe	Pe	Pe	Pe	Pe	Pe
pc-alice	tcp/13x	mell	Pe	Pe	Pe	Pe	Pe	Pe
pc-charlie	tcp/110	mell	Pe	Pe	<i>Pe</i>	Pr	Pe	Pe
pc-charlie	tcp/143	mell	Pe	Pe	Pr	Pr	Pr	Pe
pc-charlie	tcp/13x	mell	Pe	Pe	Pr	Pe	Pe	Pe

Table 7.3: Experiment: resulting network-level policy instances

Subject	Action	Object	t-2:t-1	t-1:t	t:t+2	t+2:t+7	t+7:t+8	t+8:t+10
pc-alice	/etc/popd	mell	Pe	Pe	<i>Pe</i>	Pr	Pe	Pe
pc-alice	/etc/imapd	mell	Pe	Pe	Pr	Pr	Pr	Pe
pc-alice	exchange.exe	mell	Pe	Pe	Pr	Pe	Pe	Pe
pc-charlie	/etc/popd	mell	Pe	Pe	<i>Pe</i>	Pr	Pe	Pe
pc-charlie	/etc/imapd	mell	Pe	Pe	Pr	Pr	Pr	Pe
pc-charlie	exchange.exe	mell	Pe	Pe	Pr	Pe	Pe	Pe

Table 7.4: Experiment: resulting service-level policy instances

Subject	Action	Object	t-2:t-1	t-1:t	t:t+2	t+2:t+7	t+7:t+8	t+8:t+10
alice	read	/var/spool/mail/alice	Pe	Pe	Pe	Pe	Pe	Pe
alice	write	/var/spool/mail/alice	Pe	Pe	Pe	Pe	Pe	Pe
charlie	read	/var/spool/mail/charlie	Pe	Pe	Pe	Pe	Pe	Pe
charlie	write	/var/spool/mail/charlie	Pe	Pe	Pr	Pe	Pe	Pe

Table 7.5: Experiment: resulting information-level policy instances

We verify through these tables that there always exists a path to achieve the *read_mail* activity at the network / transport and service levels, and that reading access permissions to mailboxes are maintained, fulfilling thus the minimal requirements.

7.4 Policy Decision Point

7.4.1 Objectives

In the implementation, we do not provide any notion of local strategy. Local strategy would for instance include user advertisement about an incoming change in the current policy, and possibly a delay in the actual enforcement of countermeasures which would have too many negative side-effects if brutally applied. Thus, policy instances are directly translated and enforced by a Perl PDP program, which provides *ad-hoc* enforcement of concrete policy instances, without any refinement of application scenarios. Once the translation achieved, the PDP is in charge of sending adequate commands to PEPs for actual policy enforcement.

7.4.2 Considered countermeasures

We currently consider two main kinds of countermeasures, access control and reconfiguration, declined through the three considered layers. We manage firewall rules at the network / transport level, service process start / stop at the service level, and file right management at the information level. Table 7.6 lists the considered policy instances used for countermeasure expression.

subject	action	object	reaction
ip1	port2	ip2	network or host firewall rule: filter packets from ip1 to ip2 on port2.
<i>any</i>	process	ip2	start/stop process on host ip2.
user	fileright	file	change access rights for user to file.

Table 7.6: Policy instances relevant to the implementation use case

Firewall rules. They are expressed at their elementary level, *i.e.* the more fine-grained filtering mean envisioned is to filter from a given source address to a given destination address on a given destination port. In practice, instances which are brought up by policy compilation are concrete objects, *i.e.* *is_permitted(s, a, o)* and *is_prohibited(s, a, o)*. If a countermeasure is selected for a larger scale application, for instance, if the action in the *hold* fact is '_'', this means that it will result in the instantiation of all relevant elementary concrete rules, *i.e.* a rule for each relevant port number. This is due to the way Prolog operates, since '_'' means "*all matching entities*". We do not actually worry about this, since beyond the high number of elementary rules which may be generated to enforce an (*IP1, any, IP2*) rule, it is consistent and relevant to

the model. The only disadvantage relates to enforcement, since it is more costly to enforce n rules for n different ports than a single rule blocking all traffic from $ip1$ to $ip2$. However, this is not necessarily a weakness of our implementation, since we aim at avoiding as much as possible such brutal responses. Actions taken in response to threats should be as fine-grained as possible, which results in this use case in service-oriented responses at most, that is to say, dealing with a single port number. However, means may be envisioned on factorization purpose regarding concrete entities. In particular, future work should include the possibility to group entities either in the model, or at least thanks to a preprocessing phase prior to actual enforcement.

Processes. We consider “*any*” subjects, since enforcement does not take subject into account. Information of interest is reduced to the name of the process and the host it is running on. A large number are to be derived according to the multiple subjects. Once again, it is relevant to the model, since it reflects the fact that there is no subject (neither mail user, nor mail station), which is allowed to access the given process in case of *is_prohibited*.

File access rights. User and file are found in the IDMEF *Target.File* class. Information about the action is given by the *FileAccess* subclass of the *File* class. We consider here the *read* and *write* actions, which refer to reading and writing access rights.

7.4.3 Policy enforcement

The knowledge of the PEPs may either be given in the concrete policy instances (*e.g.* for processes), or assumed to be known by the PDP (*e.g.* for firewall rules). We restrict here to the fact that the use case is so simplified that we only manage filtering between the two presented networks with a single firewall. This avoids the prior determination of the PEP to enforce, which is considered part of the future work. In more realistic settings, a full description of the PEP capabilities is required, and possibly functionalities allowing dynamic determination of the PEPs.

For each considered PEP, *i.e.* firewall *fw1* for firewall rules, and server *mel1* for service reconfigurations and file access right modification, we consider that a simple Perl agent is running on each of these hosts, ready to receive orders and to proceed enforcement. Communications between the PDP (client) and the PEPs (servers) are simply based on the use of the Perl `Socket` library, using TCP as transport protocol over IP. Security of the communications is ensured through the use of `stunnel`⁴ to provide a secure channel between the PDP and the PEPs through the use of SSL (Secured Socket Layer).

Enforcing firewall rules. Regarding the use case layout, we consider here enforcement of network firewall rules between two sub-networks, namely the user stations zone (IP 192.168.1.0), and the dmz (IP 192.168.2.0), which includes the mail server. We consider that *fw1* is running a “Netfilter” firewall. Listing 7.14 shows the policy instance

⁴<http://www.stunnel.org>

to enforce, issued from the PIE. Listing 7.15 gives the corresponding translation to be sent to the corresponding PEP agent. The target is used by the PDP to establish the connection with the adequate PEP. The Action is then executed locally on the PEP by the PEP agent. Note that this results in an *exec* action which is an *iptables* rule dealing with the FORWARD chain, *i.e.* it filters packets which traverse the firewall. Note also that we use the `-I` command in order to simply put the new rule at the beginning of the chain, assuming a “*default DROP*” policy.

Listing 7.14: Firewall sample policy: PDP input

```
<PolicyInstance type="prohibition">
  <Subject>
    <Host>
      <name>pc-charlie</name>
      <address>192.168.1.13</address>
      <netmask>255.255.255.0</netmask>
    </Host>
  </Subject>
  <Action>
    <Service>
      <name>pop3</name>
      <port>tcp/110</port>
    </Service>
  </Action>
  <Object>
    <Host>
      <name>mell</name>
      <address>192.168.2.50</address>
      <netmask>255.255.255.0</netmask>
    </Host>
  </Object>
</PolicyInstance>
```

Listing 7.15: Firewall sample policy: PDP output

```
<PEP-Message>
  <PEP-Command>
    <Target>
      <address>192.168.2.50</address>
    </Target>
    <Action type="exec">
      iptables -I FORWARD -s 192.168.1.13 -d 192.168.2.50 -p tcp --dport 110 -j DROP
    </Action>
  </PEP-Command>
</PEP-Message>
```

XML usage all along the process ensures translation through XSL transformations, as explained in [24].

Enforcing service reconfigurations. Enforcing service reconfigurations provides an example of service-level response. Service reconfigurations considered in the implementation use case consist in starting and stopping service processes. Listings 7.16 and 7.17 illustrate a sample service policy instance activated by the PIE and its corresponding translation by the PDP. In the given example, we consider the *path* attribute to stop the considered service daemon.

Listing 7.16: Service process sample policy: PDP input

```

<PolicyInstance type="prohibition">
  <Subject>
    <Host>
      <name>pc-charlie</name>
      <address>192.168.1.13</address>
      <netmask>255.255.255.0</netmask>
    </Host>
  </Subject>
  <Action>
    <Process>
      <name>popd</name>
      <pid></pid>
      <path>/etc/popd</path>
    </Process>
  </Action>
  <Object>
    <Host>
      <name>mell</name>
      <address>192.168.2.50</address>
      <netmask>255.255.255.0</netmask>
    </Host>
  </Object>
</PolicyInstance>

```

Listing 7.17: Service process sample policy: PDP output

```

<PEP-Message>
  <PEP-Command>
    <Target>
      <address>192.168.2.50</address>
    </Target>
    <Action type="exec">/etc/popd stop</Action>
  </PEP-Command>
</PEP-Message>

```

Enforcing file access rights. Enforcing file access rights provides an example of information-level response. Listings 7.18 and 7.19 illustrate a sample file policy instance activated by the PIE and its corresponding translation by the PDP. Note here that subject *charlie* does not have any particular impact on response, since we simply disable reading permission to anyone, since Charlie is the only one allowed to access the file. Note that in other applications, files can be shared between multiple users, namely groups, and rights can be managed at the group level. The countermeasure is effectively deployed through the `chmod -w` system command.

Listing 7.18: File access sample policy: PDP input

```

<PolicyInstance type="prohibition">
  <Subject>
    <User>
      <name>charlie</name>
      <uid></uid>
    </User>
  </Subject>
  <Action>
    <Fileright>
      <type>write</type>
    </Fileright>
  </Action>
</PolicyInstance>

```

```
</Action>
<Object>
  <File>
    <name>charlie</name>
    <path>/var/spool/mail/charlie</path>
    <host>mell</host>
  </File>
</Object>
</PolicyInstance>
```

Listing 7.19: File access sample policy: PDP output

```
<PEP-Message>
  <PEP-Command>
    <Target>
      <address>192.168.2.50</address>
    </Target>
    <Action type="exec">chmod -w /var/spool/mail/charlie</Action>
  </PEP-Command>
</PEP-Message>
```

7.5 Conclusion

In this chapter, we presented a sample implementation of a threat response system through a simple email use case. We showed that it is possible to implement the PIE function proposed in this thesis, and to activate new policy instances according to incoming alerts. We divided the PIE into two main functionalities, namely the Threat and Response Characterization Engine (TRCE) and the Policy Core Engine (PCE). The former deals with incoming alert analysis to activate contexts providing adequate response to characterized threats. The latter is responsible for dynamic priority assessment, ensuring thus conflict resolution, and policy compilation.

We implemented the PIE as a Prolog engine interfaced with a Perl sequencer which is responsible for input / output and sequencing of operations, especially interrogation of the Prolog engine. The Prolog engine implements the Or-BAC model and the necessary first-order logic to implement the TRCE and PCE functionalities.

The PDP is currently restricted to an ad-hoc Perl program which translates policy instances into commands to send to PEP agents for further enforcement. It does not provide any notion of local strategy in a first time. This is left to future work. PEP agents receive policy instances as system commands ready to enforce, or as specific commands, which implementation is specific to the PEP.

We provided a simple experiment, through the injection of three sample alerts in the system. According to the defined strategy, and to predefined timers to deactivate threat context, we observed the corresponding changes in the instantiated policy. We showed that it is possible to define minimal requirements which cannot be overridden whatever the threat.

Given the obtained results, we state that the approach shows an encouraging way of research, but there are still several issues to this work. In the following chapter, we discuss a few axes which may be investigated at short or middle-term to improve and extend the approach.

Chapter 8

Discussion

Contents

8.1	Implementing and improving the current model	133
8.1.1	About strategy improvements	133
8.1.2	Adaptive response deactivation	136
8.1.3	About vertical dependencies in the three layer model	136
8.1.4	Dynamic management of minimal contexts	137
8.1.5	Extend the approach to other PEPs	137
8.2	Consistent cartography model	138
8.2.1	Information of interest	138
8.2.2	About the use of cartography for threat response	140
8.3	Considerations about the policy	145
8.3.1	About a cost-sensitive model	145
8.3.2	About deployed policy validation	146
8.3.3	About policy simulation	147

8.1 Implementing and improving the current model

The current implementation brings up a proof-of-concept of the general idea, which is to include response to threat in a dynamic policy. Better management of the presented structuration according to three layers and extension of the approach to other PEPs should improve the system. In this section, we successively provide elements about strategy improvement possibilities, adaptive response deactivation, vertical dependencies in the three layer model, dynamic management of minimal contexts, and finally extension of the approach to other PEPs.

8.1.1 About strategy improvements

Accounting history for better adaptability. Taking history of alerts and applied countermeasures into account may allow us to select alternatives to countermeasures

which prove to be inefficient. This requires functionalities for response inefficiency assessment, *i.e.* providing the knowledge of countermeasures which do not satisfy the *sufficiency* requirement. More generally, history is useful for decision support systems, considering for instance countermeasures which have solved in the past a similar issue, countermeasures which have been forced by the operator, or even countermeasures which have been deactivated by the operator because of non-efficiency.

Managing scale of response. Scale of response is currently managed through a very simple solution, enlarging response at a whole activity instead of a single action. A more efficient solution may include functionalities to group policy instances in order to reduce the deployment cost. In particular, one can consider groups of users, networks or sub-networks, ranges of IP addresses, etc. There are at least two potential solutions to this problem.

A first solution requires an additional processing phase considering the set of policy instances to enforce, at the PDP level, in order to analyze if it is possible to make such groups. A simple example deals with firewall rules. In the current implementation, in order to filter from a whole sub-network composed of n hosts, the PIE provides n policy instances, *i.e.* one for each host. In such a case, the PDP could be able to better adapt deployment by first grouping these rules into a single one dedicated to the whole sub-network.

A second solution is to envision factorization of elementary entities at the PIE level, integrating grouping functionalities at the model level. Grouping is thus realized at strategy application, using less fine-grained concrete instances (*e.g.* an IP address range or a group of users) to ensure policy instances ready to enforce. This may seem to be a better solution, since response strategy already considers scale of response, although it currently needs to report as many *hold* facts as there are elementary entities. Moreover, an additional processing phase at the PDP level would probably slow down the system.

Local strategy at the PDP level. The current implementation of the PDP is purely executive regarding the policy, *i.e.* it only translates policy instances and pushes configurations to apply to PEPs. One should wonder which specific elements of local strategy are interesting to consider, especially to better balance the compromise between security and convenience. Indeed, brutal actions may often have undesirable side-effects regarding end-users. A PDP implementing translation and deployment scenarios which take into account minimal requirements of convenience and continuity of service would be of particular interest.

Let us consider a basic scenario consisting in switching from HTTP to HTTPS for OWA, considering a threat to confidentiality and/or integrity regarding the HTTP service. This scenario is simply composed of four steps:

1. Start the HTTPS service (for instance, the apache-ssl process),
2. Open port TCP/443 (generally associated with HTTPS),
3. Close port TCP/80 (HTTP),

4. Stop the HTTP service (for instance, the apache process).

Such a scenario addresses policy enforcement since it actually switches from HTTP to HTTPS. However, switching is brutal, and more fine-grained scenarios could be envisioned to provide better convenience to users. In particular, one may want to provide service redirection, so that users trying to access HTTP on port TCP/80 are automatically redirected to HTTPS on port TCP/443. Such an scenario may be expressed according to the following steps:

1. Start the HTTPS service (for instance, the apache-ssl process),
2. Open port TCP/443 (generally associated with HTTPS),
3. Keep HTTP open, but change its configuration, so that traffic which arrives on port TCP/80 is automatically redirected on port TCP/443.

Each action is enforced locally on the webserver, represented by its IP address, namely 192.168.2.51. Listing 8.1 gives the corresponding message sent by the PDP to the PEP. It includes *exec* actions, which are directly executed by the PEP agent, and a *specific* action, which is a specific functionality of this particular PEP agent. For instance, traffic redirection is ensured by the `redirect_traffic(http,https)` function in the example, which implementation must be provided by the PEP agent.

Listing 8.1: Sample policy from PDP to PEP

```
<PEP-Message desc="switch from http to https with automatic redirection">
<PEP-Command desc="start https service">
  <Target>
    <address>192.168.2.51</address>
  </Target>
  <Action type="exec">/etc/init.d/apache-ssl start</Action>
</PEP-Command>
<PEP-Command desc="open 443/https tcp port">
  <Target>
    <address>192.168.2.51</address>
  </Target>
  <Action type="exec">
    iptables -I INPUT -i eth0 -p tcp --dport 443 -m state ESTABLISHED -j ACCEPT
  </Action>
  <Action type="exec">
    iptables -I OUTPUT -i eth0 -p tcp --sport 443 -m state ESTABLISHED -j ACCEPT
  </Action>
  <Action type="exec">
    iptables -I INPUT -i eth0 -p tcp --dport 443 -m state NEW -j ACCEPT
  </Action>
</PEP-Command>
<PEP-Command desc="keep http service started but redirect traffic">
  <Target>
    <address>192.168.2.51</address>
  </Target>
  <Action type="specific">redirect_traffic(http,https)</Action>
</PEP-Command>
</PEP-Message>
```

One may also want to preserve already established network sessions for at least a given amount of time for better continuity of service. This may be achieved by adding steps consisting in informing users that they are provided a given amount of time to properly end their session on port TCP/80 before switching to port TCP/443.

Many advanced scenarios may thus be envisioned, which do not change the PIE implementation, but deal with local strategy, *i.e.* with the way PDP locally deploys policy instances. Sequencing of corresponding actions should thus be defined to provide the PDP the ability to achieve the adequate translation and enforcement.

8.1.2 Adaptive response deactivation

Response deactivation is currently managed statically, according to predefined alert lifetimes, which rely on expert knowledge about the alerts. More realistic settings would provide functionalities to dynamically deactivate threat contexts with respect to actual current threat. In our system, since one cannot tell whether the threat has disappeared once a given timer expires, alerts are asserted again once their lifetime expires until the manifestations of the threat have not disappeared from the alert flow. Thus, corresponding threat contexts remain active. Adaptive response deactivation may be based on a thorough analysis of the alert flow, considering various metrics, among which the evolution of the volume of alerts.

8.1.3 About vertical dependencies in the three layer model

Although the three layer decomposition is mainly a way to better structure the model, we observe that vertical constraints may be expressed, to better control response impact and minimal requirements. In particular, there are obvious dependencies between the three layers, namely *network / transport*, *service* and *information*. This may be represented by transverse activities, namely *Host* and *Utilize*, as explained in figure 8.1.

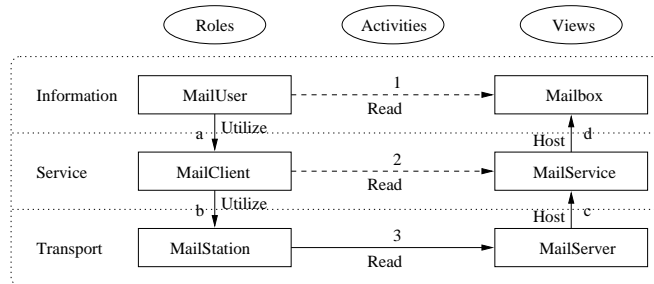


Figure 8.1: Vertical dependencies in the three layer model

Such activities may allow the definition of access control rules at the host level, ensuring better control over the minimal requirements. Listing 8.2 recaps the list of abstract permissions considered in the *Nominal* context in the email example illustrated in figure 8.1.

Listing 8.2: Permissions relevant to email access

<code>Permission(pie, 'MailUser', 'Read', 'Mailbox', 'Nominal')</code> .	(1)
<code>Permission(pie, 'MailClient', 'Read', 'MailService', 'Nominal')</code> .	(2)
<code>Permission(pie, 'MailStation', 'Read', 'MailServer', 'Nominal')</code> .	(3)
<code>Permission(pie, 'MailUser', 'Utilize', 'MailClient', 'Nominal')</code> .	(a)
<code>Permission(pie, 'MailClient', 'Utilize', 'MailStation', 'Nominal')</code> .	(b)
<code>Permission(pie, 'MailServer', 'Host', 'MailService', 'Nominal')</code> .	(c)
<code>Permission(pie, 'MailService', 'Host', 'Mailbox', 'Nominal')</code> .	(d)

Beyond classical rules (1), (2) and (3), four rules appear, namely (a), (b), (c), and (d), which model local authorizations required for mail access. At the client-side, a

mail user needs access to a mail client, which needs to be executed on a mail station. At server-side, the mail server must host a mail service, which must provide mailboxes.

It appears here that minimal requirements should take into account the fact that the lower layers are required by the higher ones. For instance, we could model that both network and service level are required for information level to be operational. Similarly, we could model that prohibiting the network or the service level would automatically lead to the loss of the information level. For the moment, we consider that such minimal requirements are defined statically in the policy, associated to minimal contexts. Modeling these dependencies could also facilitate policy rules definition, providing the possibility to derive implicit authorizations at lower levels when a single authorization is explicitly defined at a higher level. For instance, defining rule (1) only could be sufficient in the *Nominal* context, since dependencies allow to derive rules (2) and (3). We should also model explicitly that permission (1) requires permission (a), (b), (c) and (d), drawing the path of authorizations from a mail user to a mailbox.

8.1.4 Dynamic management of minimal contexts

Minimal requirements are currently formulated statically at policy definition time. This means that, considering multiple paths to resources, like in the email use case, the path to preserve in minimal mode is defined *a priori*. However, depending on the current state of the system, *i.e.* the context, some path to re-open may be preferable to others.

Many considerations may enter in the choice of the minimal requirements. In the presented implementation case study, we choose to keep open the pop path in presence of the *minimal_mail* context, because it presents less vulnerabilities than other services, especially Exchange. However, we observe that the severity of *Alert3* (exchange attack) is lower than severity of *Alert1* (pop attack). In this particular case, *i.e.* with these particular alerts, it may be more suitable to keep open the path to Exchange. Determining minimal requirements dynamically would thus be a valuable improvement to the system. This requires to take into account additional parameters, mainly dealing with risk analysis.

8.1.5 Extend the approach to other PEPs

For the moment, we limit the approach to firewall rules deployment, service reconfiguration and file access rights management. There are many other adjustment variables to enforce a security policy. A significant part of the future work will be to extend the approach to other PEPs, especially regarding authentication and encryption, quarantine and redirection.

Authentication can be constrained depending on threat, varying from simple password logins to the use of certificates or biometry. Similarly, encryption algorithms may be modulated according to threat. Wedde and Lischka [86] model a security level hierarchy of the usable algorithms for authentication and encryption (*e.g.* RSA2048 is more secure than RSA1024), in order to implement modular authorizations. On this purpose, they also define a trust level hierarchy to assign levels to communication channels. The spatial context of the considered subject and object determines this level,

e.g. hostile, internal, internal_trusted, top_secured, etc. Security requirements of channels between two hosts are then managed according to the trust level hierarchy. If authentication and encryption means are stronger or equal to required ones, access is authorized, else refused. Make use of such an approach to compose threat contexts with such spatial contexts to modulate authentication and encryption requirements could be an interesting improvement.

Another application of interest is quarantine and redirection mechanisms, for instance to isolate some threatened or threatening hosts or various resources in a secured environment with minimal service continuity requirements.

8.2 Consistent cartography model

Knowledge of the network cartography, including topology - logical as well as physical - and inventory of the hosts and hosted software, is of crucial interest for many purposes [15, 14], for network management purpose as well as for operational security. In particular, in this work, it may (1) improve intrusion detection diagnosis, supporting the assumption made regarding the reliability of the ACE, (2) facilitate policy definition enrichment and maintenance, (3) allow enrichment phase at the PIE level and possibly PDP, (4) enable context activation, (5) orient strategy application, especially through risk analysis, and (6) help policy enforcement points discovery. In a first part, we explain which information should be considered in the model, and provide a few elements about how data of interest can be collected. In a second part, we bring up a brief analysis of each of the six aforementioned applications of cartography in our work.

8.2.1 Information of interest

Network cartography consists in establishing the inventory of the hosts and analyzing component and functionality dependencies. The objective is to obtain a consistent and up-to-date model of the information system characteristics.

8.2.1.1 Inventory and topology

Inventory deals with the acquisition of host profiles. The objective is to observe and list the hosts composing the information system, dealing with hardware inventory, software inventory, but also host function inference. Host profiles include identifiers, especially IP addresses and DNS names, but also every application known to be running on the hosts, *e.g.* operating systems, but also services or various applications, like client software. Information about hardware is also considered, and modeling logical topology is also a matter of interest, especially to deal with physical and logical dependencies, as explained in the following.

Inventory of the hosts is achieved through two main approaches: active network mapping and passive network mapping. Active network mapping stimulates hosts to obtain properties. Tools like `nmap`¹ or `Nessus`² propose many stimuli which are applied

¹<http://insecure.org/nmap>

²<http://www.nessus.org>

to hosts, among which port scanning, to establish the list of open ports and to deduce running services. This approach presents the advantage of providing information when required, but is also likely to bring up many negative side-effects. First, it generates additional traffic on the network, and second, some of the tests may actually be harmful for tested hosts, sometimes leading to service or application crashes or even total loss of host connectivity. Passive network mapping collects characteristics by observing the network traffic, without any actual interaction with the information system. Host properties are deduced from many parameters in the communications, like ports or applicative banners, as explained in [75, 60]. Operating systems are recognized through passive OS fingerprinting. This approach prevents undesirable side-effects, and allows better reactivity, since changes in host profiles are rapidly observed in the communications. However, collected data clearly relies on the kind of communication hosts are involved in, that is some data which would be available with the active approach may be difficult to observe with the passive approach in some circumstances.

Host profiles allow the deduction of host functions (*e.g.* workstation, router, web proxy) from collected properties. In particular, a host which receives and emits a high number of web requests and replies in a short amount of time on port TCP/1080³ is likely to be a web proxy. In fact, one should note that such a host is to be seen running many web clients and servers, since software-related information given in the http banners refer to actual client or actual server hosts. Such applications must not be considered running on this host, but they confirm that the host is very likely to be a web proxy.

Finally, a logical vision of the network topology is obtained through the consideration of network and transport-related parameters, like IP addresses and TCP Time-To-Live (TTL).

Note that inventory may also include the discovery of users, as well as end-resources, like mail accounts or various files.

8.2.1.2 Component and functionality dependencies

Some servers (components) or services (functionalities) may be dependent from each other. For example, hosts on the network rely on infrastructure equipments, such as DNS (Domain Name System), DHCP (Dynamic Host Configuration Protocol), or directory services, like Active Directory. Applicative components also present dependencies, like web servers, which are generally built upon many functionalities, including databases and high-level language interpreters (like PHP). Loosing such functionalities or components presents side-effects on dependent entities.

To our knowledge, a few works exist on the subject, especially those of Toth [79, 78] and Laborde [55, 54]. In [79], Toth and Kruegel model dependencies between services, users and resources and allow a first approach towards evaluation of negative side-effects of response mechanisms. They model *direct* dependencies, which are dependencies of

³Traditional ports used for webproxying especially include TCP/1080, but also commonly TCP/3128, TCP/8000, TCP/8080 and possibly others. In fact, analysis cannot build upon port number consideration only. This explains why it is in particular necessary to observe additional parameters, like traffic type and volume.

entities on various services. For instance, the dependency between a user and the DNS service is a direct dependency. They also model *indirect* dependencies, which deal with network topology, impacting the communication path. For instance, a firewall or a router between two hosts introduce such dependencies. Toth and Kruegel observe that if indirect dependencies are not fulfilled, direct dependencies cannot be fulfilled too. This restates the problem of vertical dependencies in our three layer model. For instance, if a packet is filtered by a firewall (indirect dependency), communication at the network / transport level is lost, and direct dependencies are trivially impacted too. Building upon these observations, Toth and Kruegel define dependency trees, based on *OR-dependencies* (need for accessing at least one of the set of given services) and *AND-dependencies* (need for accessing all given services). They define then the *capability* value for considered entities, which is set to 1.0 when an independent entity provides service and 0.0 when an independent entity does not provide service. Then, for entities which depend on other entities, they provide a recursive algorithm that computes capabilities of each entity in the dependency tree. Based on the capability value, they define a *penalty cost* which represents the cost when the considered entity becomes unavailable. This penalty cost is then used to choose the best appropriate countermeasure among a set of potential alternatives. In [55], Laborde & al. also notice that although security application layers mainly focus on end-to-end entities, intermediate systems like routers and firewalls are also involved in the security deployment. To model the dependencies between end-to-end entities and intermediate entities, they propose a flow oriented modeling language, which builds upon a classification of the mechanisms to consider. For this purpose, they distinguish five functionalities, namely *active end-flow functionalities*, *passive end-flow functionalities*, *transform functionalities*, *filter functionalities* and *channel functionalities*, in order to model entities which are involved in communications between two end-points. This allows them in particular to verify a property of *accessibility*, which is comparable to our desire to ensure minimal requirements, especially dealing with the preservation of at least one path to resources.

Modeling dependencies would provide means to better assess threat and its consequences, as well as ensure better choice of response considering potential side-effects regarding these dependencies. Approaches like those presented by Toth and Laborde should be investigated regarding our approach, in order to propose such functionalities in our system.

8.2.2 About the use of cartography for threat response

Host profiles, *i.e.* hardware and software inventories and host functions, topology, and component and functionality dependencies provide valuable improvements to our threat response system, which we explain in the following.

8.2.2.1 Alert Correlation

In [75], our purpose with cartographic information is to enable alert correlation. With a detailed view of the vulnerabilities existing on an information system, it is possible to adjust the priority of alerts generated by intrusion detection systems, according to

the actual risk of successful intrusion [66].

The knowledge of services offered by the information systems allows us to (1) detect configurations that frequently trigger false alerts and (2) adjust alert priorities according to the existence of vulnerable services.

Correlation with vulnerabilities. Vulnerability assessment tools associate hosts and vulnerability references. This association indicates that the host is vulnerable to attacks exploiting the referenced vulnerability. Most intrusion detection systems also associate alerts with vulnerability references. A very simple and classic schema for prioritizing alerts is to increase the severity of alerts when signatures and hosts share common references.

Network mapping observation does not immediately offer the possibility to link vulnerability information and hosts. This linkage is obtained using additional information extracted from vulnerability databases such as the Open Source Vulnerability Database⁴ (OSVDB) or ICAT⁵. These databases link service names and versions with vulnerability references. Our current implementation uses OSVDB. We use the `object_correlation` table of the OSVDB database to obtain version information, and the `ext_ref` table to associate the product with vulnerability references as found in the intrusion detection signatures.

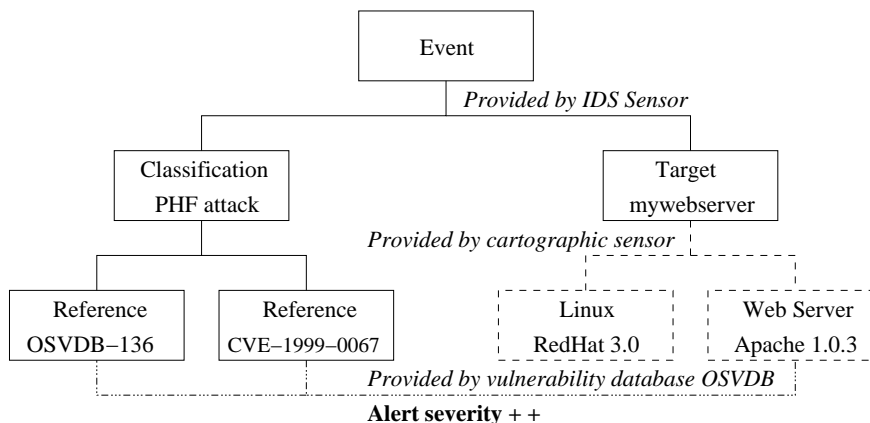


Figure 8.2: Correlation with vulnerabilities

An example of this correlation component is shown in Figure 8.2. My webserver is an apache 1.0.3 on Linux RedHat, identified by the passive mapping sensor. The intrusion detection system sends an alert with the web server as target and the PHF message as classification. Using the information from OSVDB, one can easily conclude that the web server is vulnerable to the attack.

To successfully build this correlation component, one needs to associate the product name and product version provided both by the network mapping database and by

⁴<http://www.osvdb.org/>

⁵<http://icat.nist.gov>

the vulnerability database. This is a challenge as the information that transits on the network is different from the one stored in the vulnerability database. For example, “Microsoft Internet Information Server” is known as “Microsoft IIS” in the vulnerability database. Also, many versions of Windows are known both by a product name (Windows NT, Windows XP) and by a version number that is not explicitly related to the application, but that is shown in the traces collected from the network.

Alert severity mitigation. Hosts characteristics provided by passive mapping systems can also be used to mitigate alerts severity. This is achieved by comparing the requirements of a target for an attack to be successful with the actual host characteristics. For instance, an attack exploiting a flaw in a Microsoft IIS product launched against a host which is known to use a Linux operating system will not work. Thus, the severity of the alert can be mitigated.

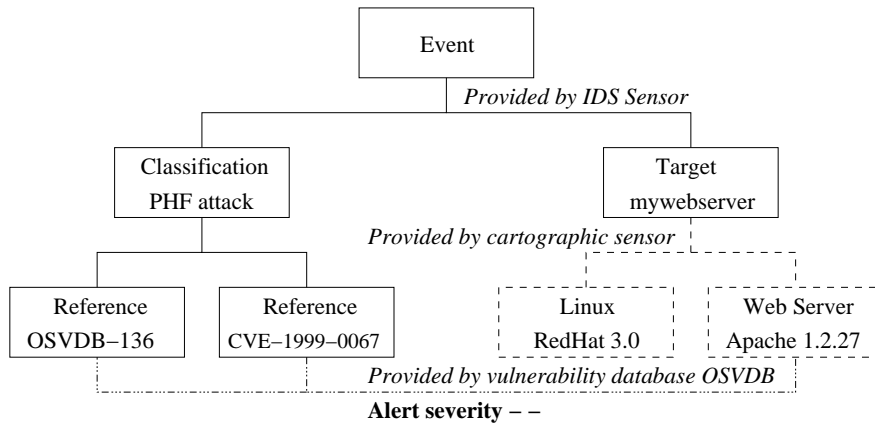


Figure 8.3: Alert severity mitigation

An example of this correlation component is shown in Figure 8.3. My webserver is an apache 1.3.27 on Linux RedHat, identified by the passive mapping sensor. The intrusion detection system sends an alert with the web server as target and the PHF message as classification. Using the information from OSVDB, one can easily conclude that the web server is *not* vulnerable to the attack.

False positive recognition. As suggested by Julisch [50] or Manganaris [59], the excessive amount of alerts triggered by intrusion detection systems is mainly provoked by false positives (*a.k.a.* false alerts). Most of these false positives come from activities misinterpreted as attacks by IDSes. These activities are induced by network components whose normal behavior includes misuse-like side effects on the monitored environment. Since misinterpreted activities are recurrent, the resulting alerts have a strong impact on the overall number of alerts. By comparing the characteristics of the hosts involved in the alerts with the configurations which are prone to false positives, it is possible to recognize alerts which are false positives.

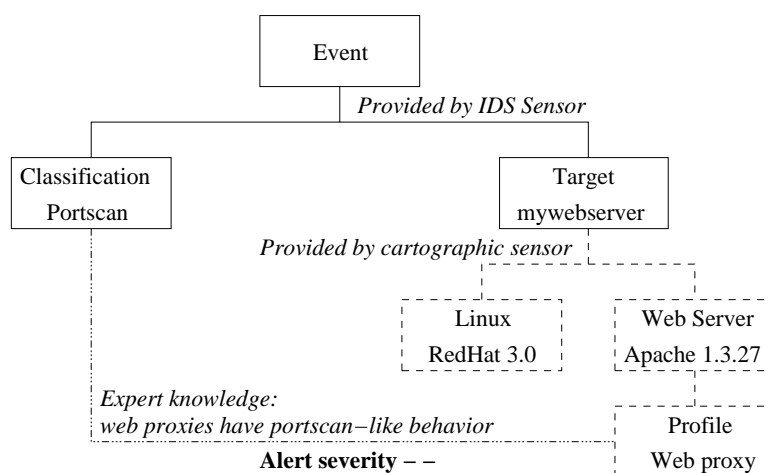


Figure 8.4: False positive recognition

An example of this correlation component is shown in Figure 8.4. The passive cartography sensor identifies my web server, and given the parameters in the HTTP request, is able to identify that this web server is acting as a proxy. The proxy profile is therefore attached to the web server. This proxy profile is known to generate portscan false alerts, hence the severity of the event is reduced.

8.2.2.2 Policy definition enrichment and maintainance

The policy definition includes inventory, especially hosts, software, users, mail accounts, and also functions of the hosts, especially regarding services. The definition of subjects, actions and objects is assisted by a network mapping system, to (1) simplify the process for the administrator (or operator), but also (2) to maintain an up-to-date definition of the policy. In particular, some changes may occur which have to be reflected rapidly in the policy definition. For instance, IP addresses or DNS names may change. In such cases, differences between the vision of hosts described in the policy and the actual vision of the information system could lead to irrelevant policy enforcement or impossibility to compile or enforce policy, which would result in errors and more generally inefficiency of the threat response system.

8.2.2.3 Enrichment phase of the PIE and possibly PDP

Cartography can also be used at the PIE level to enrich alerts which are not fully qualified. This goes from simple correspondences between IP addresses with DNS names to the knowledge of very specific information like process identifiers (pid). However, it remains unclear whether the knowledge of such detailed information have to be considered at the PIE level. The fact is that pids are very local information, and that there is obviously no particular interest to define the policy so fine-grained that each running process is considered on its own, that is as a single instance. In fact,

we consider in our approach global definitions for instantiable objects like processes, so that multiple instances may match a same entity in the policy definition. At least, the knowledge of the pid may be of interest for policy enforcement at the PDP level. This would lead to the need of a local model at the PDP level, which would consider enrichment of information potentially interesting for actual policy enforcement.

8.2.2.4 Context activation

Cartography is strongly linked with contexts, especially spatial contexts. Knowledge of the topology, both logical and physical, provides means to evaluate spatial contexts, such as *in_vlan0* (logical) or *in_office* (physical).

8.2.2.5 Strategy application

We feel through Section 6.3.3 the interest of a risk analysis considering vulnerability references reported into alerts and possibly actual vulnerabilities of the hosts. At least, one may assume that the ACE actually reports alerts characterizing hosts which are effectively vulnerable to considered attacks. However, even in such a configuration, vulnerability assessment is still necessary to extend the approach to other potentially vulnerable hosts, so that response is not only dedicated to a given attack or intrusion (*e.g.* centered on victim only), but also to an actual threat for the information system (*e.g.* extended to other potential victims). Vulnerability assessment requires the knowledge of host profiles, including name and versions of hosted applications, which is brought up by a consistent cartography model. Vulnerability assessment should be considered to carry out the required risk analysis in order to provide better response strategy.

8.2.2.6 PEP discovery

Policy enforcement points (PEPs) are currently statically declared in our system. This forces the security operator to make an exhaustive list of PEPs, including the parameters which are necessary for translation of policy instances into configurations at the PDP level.

Allowing automated discovery and selection of PEPs regarding a given policy instance is one of the field of research cartography may lead to. For instance, let us consider two hosts H_a and H_b located on two different sub-networks between which a firewall allows filtering. A policy instance enforces port filtering between two hosts H_a and H_b . The topology is used to look for a host which function is *firewall* located between H_a and H_b . The inventory allows us to deduce which firewall implementation to consider for translation and actual deployment of the required reconfiguration.

8.2.2.7 Additional discussion about cartography

We state that modeling the information system characteristics is of major interest for many purposes, as previously explained. According to the results we obtained in our passive network mapping experiment [75], we argue that it is an interesting

field of research. Future work should include both consideration of cartography in the aforementioned applications, especially regarding dependencies between entities, and also improvements of our passive network mapping system. In particular, the approach should be extended to other applicative protocols than HTTP. Regarding applicative protocols, a significant improvement would be to implement protocol recognition, to ensure that traffic is properly associated with the underlying protocol independently of port information available in network packets.

Another challenge in this area is the implementation of a reliable protocol analyzer for Microsoft protocols. While service platforms rely heavily on well known web protocols for which protocol analysis tools are widely available, internal traffic relies heavily on NetBIOS protocols, supporting especially file sharing protocols such as CIFS and direct connections between Microsoft Exchange and mail clients, as mentioned in the email case study. This traffic is currently not analyzed, which indicates that we are missing a lot of information related to our client machines.

Finally, although we obtained encouraging results in our approach, we are to admit that passive network mapping suffers from limitations concerning temporality. In fact, properties are updated only thanks to interactions between hosts, that is to say it is challenging to ensure a consistent and up-to-date view of host properties using only the passive acquisition approach. Therefore, we wonder whether adding a few active functionalities with minor side effects on the monitored information system could provide a significant gain.

8.3 Considerations about the policy

In this section, we first discuss the need for actually developing a cost-sensitive model. We then provide in a second section a few considerations about policy validation. Finally, a last section is dedicated to the interest of integrating policy simulation functionalities in the system.

8.3.1 About a cost-sensitive model

Our current model does not explicitly take into account the notion of cost. Developing a model which considers especially the *application cost*, the *compilation cost* and the *deployment cost* of the policy would be of great interest.

Application cost. Considering a given policy expression, one should wonder about the consequences of such a policy on users, services, business requirements, etc. Application cost evaluates the actual impact of the policy application on the system. In particular, application cost ensures that response leads to a better state than non-response (cost of response versus cost of non-response). Indeed, a particularly undesirable side-effect of automated response would be to result in a situation which is even worse than if no response had been triggered.

Application cost is strongly linked with the notion of context. In particular, one should consider brutal countermeasure effects, *i.e.* countermeasures which do not provide any service continuity. For instance, if a countermeasure increases authentication

constraints, a question which arises is “do we need to re-authenticate users which had already been logged on with the current authentication mechanism, or do we need to shutdown their sessions, requiring for the new authentication mechanism?” Answering this question requires to consider the cost of shutting down current sessions (especially business cost). Similarly, dealing with file access control, one may consider the consequences of brutally closing a file which is currently being written.

Beyond metrics which have to be found to characterize application cost, one should also provide means to ensure a better compromise regarding response. The PDP, through the deployment of efficient local strategy, can guarantee minimal continuity of service, and deploy rollback mechanisms to recover from illegitimate file modifications or deletions. Considering the possibility to refuse response if it leads to a worse state than non-response, the PIE should be able to envision alternatives to response which seem too brutal.

Compilation cost. Compilation cost deals with complexity of a given policy update. It considers that a response to current threat may require a large number of modifications, resulting in the instantiation of multiple policy instances at policy compilation.

Compilation cost includes conflict resolution cost among the set of policy rules. Depending on the number of considered rules and on the Conflict Resolution Strategy (CRS), compilation may be more or less time-costly, which is to be considered regarding threat response reactivity.

Finally, compilation cost is influenced by the heterogeneity of the equipments. Indeed, many types of PEPs exist, with many different implementations, which renders more or less costly the countermeasure translation process. Note that we include here translation cost in the compilation cost, considering that compilation may be assimilated to the process which actually produces new configurations to enforce⁶.

Deployment cost. Deployment cost considers metrics related to the time taken for a given countermeasure to be effective. Deployment cost mainly deals with the number of equipments to enforce. This partly explains why we avoid at much as possible client-centric response, preferring server-centric response, which addresses threat response close to service, and minimizes the number of enforcement operations.

8.3.2 About deployed policy validation

In our approach, policy instances are pushed from the PIE to PEPs, assuming first that the PDP is able to translate, and then that the existing PEPs allow enforcement. There is no mechanism to ensure that the proposed response actually addresses the issue it has been deployed for. We only consider PEPs as sensors. They report information to the ACE which may be likely to assess the new state of the system, *i.e.* to characterize if the response successfully applies. We assumed in our approach the *correctness* requirement, *the countermeasures do what they claim to do*, and we aim at providing the *sufficiency*

⁶This is not exactly the case in our proposal, where compilation is sometimes only assimilated to instantiation of policy instances at the PIE level.

requirement through response strategy, *if the countermeasures do what they claim to do, the threats to the assets are countered.*

In practice, such assumptions may be considered too strong. The possibility for the PDP to request alternative countermeasures in case of non-applicability or inefficiency of the selected countermeasure should be investigated. This requires at least two additional elements: (1) an algorithm which validates policy at the concrete level regarding the abstract definition of the policy, and (2) a loop allowing the PDP to report information and to request for alternatives to the PIE.

8.3.3 About policy simulation

Another interesting field of research deals with policy simulation. Policy simulation consists in the virtual analysis of the effects of a given countermeasure on a model of the information system. Of course, this first requires the existence of such a model, which needs in addition to be up-to-date, to avoid misinterpretations. We come back here to the necessity of a consistent cartography model envisioned in section 8.2.

Although simulation may appear to be time-consuming, since it requires the enforcement of the whole current policy on the model to assess the effects of the new policy, this may prove to be a gain in certain cases regarding response efficiency. Except cases where the selected countermeasure is already suitable, it would prevent from inefficient or even negative countermeasures enforcement.

Simulation is thus a way to implement alternative countermeasure selection, avoiding (1) the need for PDP-PIE loopbacks in case of non-applicability, and (2) the need for deployed policy validation.

An architecture taking into account simulation would probably be built upon figure 8.5. A simulation component is added, including a virtual PDP and a model of the information system, to first simulate the impact of countermeasures on the model before effectively deploying them or selecting alternatives.

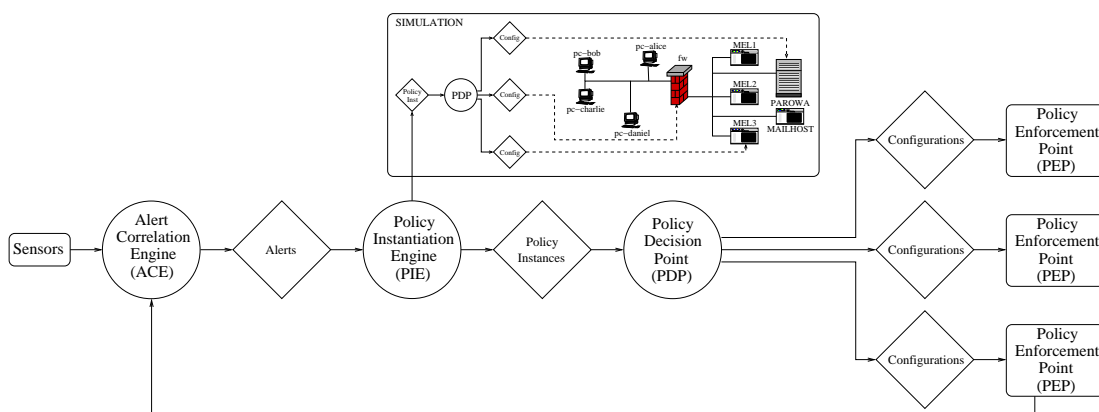


Figure 8.5: Threat response system with simulation component

Chapter 9

Conclusion and Perspectives

Throughout this thesis, our objective was to propose the connection of security monitoring to security policies in order to provide automated threat response. We build upon the fact that intrusion detection technology has matured enough to provide reliable diagnoses, and thus to envision automated response. We also rely on the fact that recent advances in the field of security policy formalization, especially Or-BAC, provide the necessary expressiveness to integrate threat response, especially the possibility to adapt policy instantiation to current context.

We first proposed an architecture for a threat response system, drawing a parallel with the AAA architecture, and Col. John Boyd OODA's military strategy. We recommended the adjunction of a Policy Instantiation Engine (PIE) to the traditional Policy Decision Point (PDP) and Policy Enforcement Points (PEPs). The PIE considers a generic policy, defined at the abstract level, which is able to activate concrete policy instances depending on alerts received by an Alert Correlation Engine (ACE). The PIE is thus the main contribution of our architecture proposal, since it is responsible for characterizing threats and triggering adequate countermeasures, as new policy instances. We also redefined the function of the PDP, which is relegated to the role of a local decisional entity. It mainly deals with policy instances translation according to the capabilities of the PEPs. However, it may also be in charge of local strategy functionalities, which are not in the scope of the PIE.

We then showed how we can make use of the Or-BAC model to implement this architecture. We especially described how to model Or-BAC entities to fit threat response requirements through an email use case. Thanks to its generic policy definition at the abstract level, Or-BAC is used to derive many different policy instantiations according to current context. We provided means to model and activate threat contexts, and proposed threat context deactivation through the use of timers which are predefined thanks to expert knowledge. Although this is a simple approach, we argue that it allows a continuous assessment of the best compromise between multiple variables, among which security, but also performance, convenience and business constraints. In addition, one should note that context lifetimes are extended as long as the threat is characterized by the PIE, *i.e.* as long as similar alerts are reported by the ACE. We also provided context composition and defined how to deal with composed context priorities. Conflict resolution has been addressed through dynamic priority assessment.

Two conflict resolution strategies have been proposed in order to possibly determine a partial order between potentially conflicting security rules, which relieves the security operator from the tedious task of manually ordering all security rules.

We discussed response strategy, considering three major adjustment variables, namely direction, target layer and scale of response. We thus first envisioned either attacker-centric response or victim-centric response. Secondly, we declined response through network / transport, service, and information levels. Thirdly, we considered either very specific concrete-level or enlarged abstract-level actions. We explained how we consider information available into alerts, but also proposed possibilities of enrichment considering the model of the world. Finally, regarding strategy, we stated that detailed analysis of the vulnerability references may provide valuable data to orient strategy.

We provided a simple implementation of a threat response system, especially a proof-of-concept of the PIE functionality, although it is currently restricted to a limited use case. However, we argue that the approach may be extended, through the definition of new entities composing the information system, new security rules, and also new vulnerabilities and strategy definitions. Policy instances are translated thanks to an *ad-hoc* PDP implementation, and further enforced thanks to a simple agent deployed on the considered PEPs.

Perspectives mainly deal with model improvements and strategy assessment. In particular, one should focus on modeling dependencies between layers (network / transport, service, and information) and services (end-services, like server functionality dependencies, but also infrastructure dependencies). Regarding strategy, a suitable orientation would be to build upon a cost-model, allowing better assessment of the most adequate response, especially regarding the risk, the response impact on the system, and deployment constraints. Extending the approach to other kinds of PEPs is another area of research. In particular, it could be interesting to focus on a dynamic assessment of cryptographic mechanisms, both regarding authentication and encryption of the communications. A cost-model would help in such a task, the trade-off between security and other operational constraints leading to the enforcement of different encryption means, through the adaptation of algorithms and key lengths to current context. Redirection also seems to be an application of interest for our system, in particular to isolate assumed attackers or potentially threatening subjects in constrained environments.

To conclude, this work was clearly the opportunity to investigate a large number of concepts and technologies, dealing with three main fields of research, namely *intrusion detection*, *intrusion and threat response*, and *security policies*. Our wish was to put the light on a particularly sensitive issue which had not been addressed before: linking security policies with the means used to control their fulfillment, namely security monitoring, and especially intrusion detection. Obviously, the finality is to provide dynamic and automated response to threat. Although we only give here a preliminary answer to the considered problem, we prove through a very simple implementation that it is an encouraging way of research. Moreover, many questions arose which provide substantial future work to envision at short and middle-term. However, future work must be done within a crucial constraint: response must not lead to a state which would be even worse than in the absence of response. This is obviously the fundamental condition to satisfy to succeed in the will of threat response automation.

Bibliography

- [1] Conceptual Model and Architecture of MAFTIA. MAFTIA deliverable D21, January 2003.
- [2] Common Criteria for Information Technology Security Evaluation - Part 1: Introduction and general model, September 2006.
- [3] Ehab S. Al-Shaer and Hazem H. Hamed. Discovery of policy anomalies in distributed firewalls. In *INFOCOM*, 2004.
- [4] Algirdas Avizienis and L. Chen. On the implementation of N-version programming for software fault-tolerance during execution. In *Proceedings of the IEEE COMPSAC 77*, pages 149–155, November 1977.
- [5] Debra Anderson, Thane Frivold, and Alfonso Valdes. Next-Generation Intrusion Detection Expert System (NIDES) - A Summary. Technical Report SRI-CSL-95-07, SRI, Menlo Park, May 1995.
- [6] James P. Anderson. Computer Security Threat Monitoring and Surveillance. Technical report, Fort Washington - Technical Report Contract 79F26400, 1980.
- [7] Ross J. Anderson. *Security Engineering: a Guide to Building Dependable Distributed Systems*. Wiley Computer Publishing, 2001.
- [8] Fabien Autrel. *Fusion, corrélation pondérée et réaction dans un environnement de détection d'intrusions coopérative*. PhD thesis, Ecole Nationale Supérieure de l'Aéronautique et de l'Espace, March 2005.
- [9] Algirdas Avizienis and John P. J. Kelly. Fault Tolerance by Design Diversity: Concepts and Experiments. *Computer*, 17:67–80, August 1984.
- [10] Algirdas Avizienis, Jean-Claude Laprie, and Brian Randell. Fundamental Concepts of Dependability. Research Report, April 2001.
- [11] Daniel Barbará, Julia Couto, Sushil Jajodia, Leonard Popyack, and Ningning Wu. ADAM: Detecting Intrusions by Data Mining. In *Proceedings of the 2001 IEEE Workshop on Information Assurance and Security*, United States Military Academy, West Point, NY, June 2001.

-
- [12] John R. Boyd. The Essence of Winning or Losing. Unpublished lecture notes, 1996.
- [13] Richard Brackney. Cyber-Intrusion Response. In *Proceedings of the 17th IEEE Symposium on Reliable Distributed Systems*, page 413, West Lafayette, IN, October 1998.
- [14] Yuri Breitbart, Minos Garofalakis, Ben Jai, Cliff Martin, Rajeev Rastogi, and Avi Silberschatz. Topology discovery in heterogeneous IP networks: the NetInventory system. *IEEE/ACM Transactions on Networking*, 12(4):401–441, June 2004.
- [15] Yuri Breitbart, Minos N. Garofalakis, Cliff Martin, Rajeev Rastogi, S. Seshadri, and Abraham Silberschatz. Topology Discovery in Heterogeneous IP Networks. In *INFOCOM (1)*, pages 265–274, 2000.
- [16] Patrick Brézillon and Ghita Kouadri Mostéfaoui. Context-Based Security Policies: A New Modeling Approach. In *PerCom Workshops*, pages 154–158. IEEE Computer Society, 2004.
- [17] Mark Burgess. Cfengine: a system configuration engine. Technical report, University of Oslo, 1993.
- [18] Curtis A. Carver, John M.D. Hill, and Udo W. Pooch. Limiting Uncertainty in Intrusion Response. In *Proceedings of the 2001 IEEE workshop on Systems, Man, and Cybernetics Information Assurance and Security*, pages 142–147, United States Military Academy, West Point, NY, June 2001.
- [19] Curtis A. Carver and Udo W. Pooch. An Intrusion Response Taxonomy and its Role in Automatic Intrusion Response. In *IEEE Systems, Man, and Cybernetics Information Assurance and Security Workshop*, June 2000.
- [20] K. Chan, J. Seligson, D. Durham, S. Gai, K. McCloghrie, S. Herzog, F. Reichmeyer, R. Yavatkar, and A. Smith. COPS Usage for Policy Provisioning (COPS-PR). RFC 3084, March 2001.
- [21] Microsoft Corporation. Introduction to Network Access Protection. Microsoft Windows Server System White Paper, December 2006.
- [22] Alexis Cort. Algorithm-based approaches to intrusion detection and response. Part of the Information Security Reading Room, SANS Institute, March 2004.
- [23] Frédéric Cuppens, Nora Cuppens-Boulahia, and Alexandre Miège. Inheritance hierarchies in the Or-BAC Model and application in a network environment. In Andrei Sabelfeld, editor, *FCS'04*, volume 31, pages 41–59, June 2004.
- [24] Frédéric Cuppens, Nora Cuppens-Boulahia, Thierry Sans, and Alexandre Miège. A Formal Approach to Specify and Deploy a Network Security Policy. In *Second Workshop on Formal Aspects of Security and Trust (FAST)*, Toulouse, France, 2004.

-
- [25] Frédéric Cuppens, Sylvain Gombault, and Thierry Sans. Selecting Appropriate Counter-Measures in an Intrusion Detection Framework. In *17th IEEE Computer Security Foundations Workshop (CSFW)*, Pacific Grove, CA, USA, June 2004.
- [26] Frédéric Cuppens and Alexandre Miège. Alert Correlation in a Cooperative Intrusion Detection Framework. In *Proceedings of the IEEE Symposium on Security and Privacy*, 2002.
- [27] Frédéric Cuppens and Alexandre Miège. Modelling Contexts in the Or-BAC Model. In *ACSAC '03: Proceedings of the 19th Annual Computer Security Applications Conference*, page 416. IEEE Computer Society, 2003.
- [28] Frédéric Cuppens and Rodolphe Ortalo. LAMBDA: A Language to Model a Database for Detection of Attacks. In Hervé Debar, Ludovic Mé, and Shyhtsun Felix Wu, editors, *Recent Advances in Intrusion Detection*, volume 1907 of *Lecture Notes in Computer Science*, pages 197–216. Springer, 2000.
- [29] Oliver Dain and Robert K. Cunningham. Fusing a Heterogeneous Alert Stream into Scenarios. In *Proceedings of the 2001 ACM Workshop on Data Mining for Security Applications*, pages 1–13, November 2001.
- [30] C. de Laat, G. Gross, L. Gommans, J. Vollbrecht, and D. Spence. Generic AAA Architecture. RFC 2903, August 2000.
- [31] Hervé Debar, David Curry, and Ben Feinstein. The Intrusion Detection Message Exchange Format. RFC 4765, November 2006.
- [32] Hervé Debar, Benjamin Morin, Frédéric Cuppens, Fabien Autrel, Ludovic Mé, Bernard Vivinis, Salem Benferhat, Mireille Ducassé, and Rodolphe Ortalo. Détection d'intrusions : corrélation d'alertes. *Technique et science informatiques*, 23:359–390, 2004.
- [33] Hervé Debar, Yohann Thomas, Frédéric Cuppens, and Nora Cuppens-Boulahia. Enabling automated threat response through the use of a dynamic security policy. *Journal in Computer Virology*, 2007.
- [34] Hervé Debar, Yohann Thomas, Frédéric Cuppens, and Nora Cuppens-Boulahia. *Response : bridging the link between intrusion detection alerts and security policies*. Book chapter, to appear.
- [35] Hervé Debar, Yohann Thomas, Nora Cuppens-Boulahia, and Frédéric Cuppens. Using Contextual Security Policies for Threat Response. In Roland Bueschkes and Pavel Laskov, editors, *Proceedings of the 3rd Conference on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA 06)*, Berlin, Germany, July 2006. Springer.
- [36] Hervé Debar and Andreas Wespi. Aggregation and Correlation of Intrusion-Detection Alerts. In Wenke Lee, Ludovic Mé, and Andreas Wespi, editors, *Proceedings of the 4th International Symposium on Recent Advances in Intrusion Detection (RAID)*, volume 2212, pages 85–103. Springer, October 2001.

-
- [37] Dorothy E. Denning, D. L. Edwards, R. Jagannathan, T. F. Lunt, and P. G. Neumann. A Prototype IDES — A Real-Time Intrusion Detection Expert System. Technical report, Computer Science Laboratory, SRI International, 1987.
- [38] Christophe Dousson. *Suivi d'évolutions et reconnaissance de chroniques*. PhD thesis, 1994.
- [39] D. Durham, J. Boyle, R. Cohen, S. Herzog, R. Rajan, and A. Sastry. The COPS (Common Open Policy Services) Protocol. RFC 2748, January 2000.
- [40] R. Enns. NETCONF Configuration Protocol. RFC 4741, December 2006.
- [41] S. Farrell, J. Vollbrecht, P. Calhoun, L. Gommans, G. Gross, B. de Bruijn, C. de Laat, M. Holdrege, and D. Spence. AAA Authorization Requirements. RFC 2906, August 2000.
- [42] Eric A. Fisch. *A Taxonomy and implementation of automated responses to intrusive behavior*. PhD thesis, Texas A&M University, 1996.
- [43] Sally Floyd. Inappropriate TCP Resets Considered Harmful. RFC 3360, August 2002. <http://www.ietf.org/rfc/rfc3360.txt>.
- [44] Frédéric Cuppens and Nora Cuppens-Boulahia and Meriam Ben Ghorbel. High Level Conflict Management Strategies in Advanced Access Control Models. *Electronic Notes in Theoretical Computer Science (ENTCS)*, 186:3–26, July 2007.
- [45] Tim Grant. Unifying Planning and Control using an OODA-based Architecture. In *Proceedings of the Annual Conference of the South African Institute of Computer Scientists and Information Technologists (SAICSIT)*, pages 159–170, 2005.
- [46] Michael A. Harrison, Walter L. Ruzzo, and Jeffrey D. Ullman. Protection in Operating Systems. *Communication of the ACM*, 19(8):461–471, August 1976.
- [47] Dan Hawrylkiw. Network Intrusion and use of automated responses. Intrusion Detection FAQ, SANS Institute.
- [48] Sushil Jojodia, Michiharu Kudo, and V.S. Subrahmanian. *Provisional Authorizations*, pages 133–159. Kluwer Academic Publishers, 2001.
- [49] Erland Jonsson. Towards an Integrated Conceptual Model of Security and Dependability. In *ARES '06: Proceedings of the First International Conference on Availability, Reliability and Security (ARES'06)*, pages 646–653. IEEE Computer Society, 2006.
- [50] Klaus Julisch. Mining Alarm Clusters to Improve Alarm Handling Efficiency. In *Proceedings of the 17th Annual Computer Security Applications Conference (ACSAC)*, December 2001.

- [51] Anas Abou El Kalam, Salem Benferhat, Alexandre Miège, Rania El Baida, Frédéric Cuppens, Claire Saurel, Philippe Balbiani, Yves Deswarte, and Gilles Trouessin. Organization Based Access Control. In *Proceedings of IEEE 4th International Workshop on Policies for Distributed Systems and Networks (POLICY 2003)*, Lake Como, Italy, June 2003.
- [52] John P. J. Kelly, Thomas I. McVittie, and Wayne I. Yamamoto. Implementing Design Diversity to Achieve Fault Tolerance. *IEEE Software*, 08(4):61–71, 1991.
- [53] Michiharu Kudo and Satoshi Hada. XML Document Security based on Provisional Authorization. In *CCS '00: Proceedings of the 7th ACM conference on Computer and communications security*, pages 87–96. ACM Press, 2000.
- [54] Romain Laborde. *Un cadre formel pour le raffinement de l'information de gestion de sécurité réseau : Expression et Analyse*. PhD thesis, Université Toulouse III - Paul Sabatier, 2005.
- [55] Romain Laborde, François Barrère, and Abdelmalek Benzekri. A formal framework (Expression + Analysis) for network security mechanisms configuration. In *Proceedings of the 2005 Fourth IEEE International Symposium on Network Computing and Applications (NCA'05)*, 2005.
- [56] Dionysius Lardner. Babbage's Calculating Engine. *Edinburgh Review*, (59):263–327, July 1834.
- [57] Wenke Lee, Wei Fan, Matthew Miller, Salvatore J. Stolfo, and Erez Zadok. Toward Cost-Sensitive Modeling for Intrusion Detection and Response. *Journal of Computer Security*, 10(1/2):5–22, 2002.
- [58] Stefanos Manganaris, Marvin Christensen, Dan Zerkle, and Keith Hermiz. A Data Mining Analysis of RTID Alarms. In *Recent Advances in Intrusion Detection (RAID99)*, 1999.
- [59] Stefanos Manganaris, Marvin Christensen, Dan Zerkle, and Keith Hermiz. A Data Mining Analysis of RTID Alarms. In *Computer Networks*, number 4, pages 571–577, 2000.
- [60] Frédéric Massicotte, Mathieu Couture, and Yvan Labiche. Context-Based Intrusion Detection Using Snort, Nessus and Bugtraq Databases. In *PST*, 2005.
- [61] Cédric Michel and Ludovic Mé. ADeLe: An Attack Description Language for Knowledge-Based Intrusion Detection. In Michel Dupuy and Pierre Paradinas, editors, *SEC*, volume 193 of *IFIP Conference Proceedings*, pages 353–368. Kluwer, 2001.
- [62] Alexandre Miège. *Definition of a formal framework for specifying security policies. The Or-BAC model and extensions*. PhD thesis, ENST, 2005.
- [63] B. Moore. Policy Core Information Model (PCIM) Extensions. RFC 3460, January 2003.

- [64] B. Moore, E. Ellesson, J. Strassner, and A. Westerinen. Policy Core Information Model – Version 1 Specification. RFC 3060, February 2001.
- [65] Benjamin Morin and Hervé Debar. Correlation of Intrusion Symptoms: An Application of Chronicles. In Giovanni Vigna, Erland Jonsson, and Christopher Kruegel, editors, *Proceedings of the 6th International Symposium on Recent Advances in Intrusion Detection (RAID)*, volume 2820. Springer, September 2003.
- [66] Benjamin Morin, Ludovic Mé, Hervé Debar, and Mireille Ducassé. M2D2: A Formal Data Model for IDS Alert Correlation. In *Proceedings of the Fifth International Symposium on Recent Advances in Intrusion Detection (RAID)*, October 2002.
- [67] Peng Ning, Yun Cui, and Douglas S. Reeves. Constructing Attack Scenarios Through Correlation of Intrusion Alerts. In *Proceedings of the 9th Conference on Computer and Communication Security*, 2002.
- [68] Stephen Northcutt. *Network Intrusion Detection: An Analyst's Handbook*. New Riders, 1999.
- [69] Phillip A. Porras and Peter G. Neumann. EMERALD: Event Monitoring Enabling Responses to Anomalous Live Disturbances. In *Proc. 20th NIST-NCSC National Information Systems Security Conference*, pages 353–365, 1997.
- [70] Aaron Rankin. On the State and Progression of Automated Intrusion Response Models.
- [71] Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein, and Charles E. Youman. Role-Based Access Control Models. *IEEE Computer*, 29(2):38–47, 1996.
- [72] Robert W. Shirey. Internet Security Glossary. RFC 2828, May 2000. <http://www.ietf.org/rfc/rfc2828.txt>.
- [73] Natalia Stakhanova, Samik Basu, and Johnny Wong. A taxonomy of intrusion response systems. *International Journal of Information and Computer Security*, 1(1/2):169–184, 2007.
- [74] J. Strassner, B. Moore, R. Moats, and E. Ellesson. Policy Core Lightweight Directory Access Protocol (LDAP) Schema. RFC 3703, February 2004.
- [75] Yohann Thomas, Hervé Debar, and Benjamin Morin. Improving Security Management through Passive Network Observation. In *ARES '06: Proceedings of the First International Conference on Availability, Reliability and Security (ARES'06)*, pages 382–389. IEEE Computer Society, 2006.
- [76] Elvis Tombini, Hervé Debar, Ludovic Mé, and Mireille Ducassé. A Serial Combination of Anomaly and Misuse IDSes Applied to HTTP Traffic. In *Proceedings of the 20th Annual Computer Security Applications Conference (ACSAC 2004)*, Tucson, Arizona, USA, December 2004. IEEE Computer Society.

- [77] Eric Totel, Frédéric Majorczyk, and Ludovic Mé. COTS Diversity based Intrusion Detection and Application to Web servers. In Alfonso Valdes and Diego Zamboni, editors, *Proceedings of the 8th International Symposium on Recent Advances in Intrusion Detection (RAID)*. Springer, September 2005.
- [78] Thomas Toth. *Improving Intrusion Detection Systems*. PhD thesis, Technischen Universität Wien, 2003.
- [79] Thomas Toth and Christopher Kruegel. Evaluating the Impact of Automated Intrusion Response Mechanisms. In *Proceedings of the 18th Annual Computer Security Applications Conference (ACSAC)*, Las Vegas, NV, USA, December 2002. IEEE Computer Society Press.
- [80] Jeffrey D. Ullman. *Principles of Database and Knowledge Base Systems*. Computer Science Press, 1989.
- [81] Alfonso Valdes and Keith Skinner. Probabilistic Alert Correlation. In Wenke Lee, Ludovic Mé, and Andreas Wespi, editors, *Proceedings of the 4th International Symposium on Recent Advances in Intrusion Detection (RAID)*, volume 2212. Springer, October 2001.
- [82] Paulo Esteves Veríssimo, Nuno Ferreira Neves, and Miguel Pupo Correia. Intrusion-Tolerant Architectures: Concepts and Design. In *Architecting Dependable Systems*, volume 2677. Springer Verlag, 2003.
- [83] Jouni Viinikka, Hervé Debar, Ludovic Mé, and Renaud Séguier. Time Series Modeling for IDS Alert Management. In *Proceedings of the ACM Symposium on InformAtion, Computer and Communications Security(AsiaCCS'06)*, pages 102–113, Taipei, Taiwan, March 2006.
- [84] J. Vollbrecht, P. Calhoun, S. Farrell, L. Gommans, G. Gross, B. de Bruijn, C. de Laat, M. Holdrege, and D. Spence. AAA Authorization Application Examples. RFC 2905, August 2000.
- [85] J. Vollbrecht, P. Calhoun, S. Farrell, L. Gommans, G. Gross, B. de Bruijn, C. de Laat, M. Holdrege, and D. Spence. AAA Authorization Framework. RFC 2904, August 2000.
- [86] Horst F. Wedde and Mario Lischka. Role-based access control in ambient and remote space. In *SACMAT '04: Proceedings of the ninth ACM symposium on Access control models and technologies*, pages 21–30, New York, NY, USA, 2004. ACM Press.
- [87] Maj. Gregory B. White, Eric A. Fisch, and Udo W. Pooch. Cooperating Security Managers: A Peer-Based Intrusion Detection System. *IEEE Network*, 10(1):20–23, January/February 1996.
- [88] Mark Wood and Michael Erlinger. Intrusion Detection Message Exchange Requirements. Internet Draft, October 2002. Work in progress, expires April 22, 2003.

Appendix A

Glossary

Action. Any measure actually taken over policy enforcement points to implement a security policy. In a policy-based response system, an action is any measure taken in response to a threat.

Activity. Data collected by sensors to characterize events of interest for the security operator. Activity may be collected through network traffic observation, host log files or application log files.

Access Control. Protection of system resources against unauthorized access. Access control is a way to address security policy enforcement issues, mainly dealing with confidentiality and integrity.

Adaptive policy. Dynamic policy adapting to current context.

Administrator (*a.k.a. security administrator*). The human in charge of the definition of the security policy of a given organization. In particular, the administrator is responsible for IDS deployment and configuration. Sometimes, the administrator is also the operator of the IDS.

Aggregation. Technique aiming at grouping alerts sharing a common characteristic. Aggregation may be based on multiple parameters, allowing in particular to correlate alerts coming from various heterogeneous intrusion detection systems.

Alarm. Sometimes used in the intrusion detection literature to qualify high-level alerts. We refer to alarms as alerts, since granularity of the alerts is a very subjective concept.

Alert. Message reported by analyzers to managers, indicating that a given event has been observed, especially a malicious event or part of an attack scenario.

Alert Correlation Engine. Component in charge of alert correlation, that is false positives recognition, semantics improvement and severity assessment, to provide better alerts (higher-level alerts). Alerts reported by an ACE should be relevant, considering threat, and making the link with vulnerabilities of the potential victims.

Analyzer. Intrusion detection component in charge of analyzing data collected by sensors and reporting alerts characteristic of unauthorized or undesired activity, which may lead to intrusions.

Assets. Entities of value of the information system, *e.g.* various resources, which are to be protected.

Attack. Malicious intentional fault attempted on a software or hardware component of an information system, aiming at exploiting a vulnerability and leading to an intrusion in case of success.

Attack response. Reaction to an attack.

Attack scenario. Sequence of elementary attacks leading to an intrusion in case of success.

Availability. Security property aiming at ensuring that expected services and resources are delivered correctly to users and processes.

Closed policy. “Default deny” policy, *i.e.* what is not explicitly permitted or prohibited is implicitly prohibited.

Clustering. Technique aiming at grouping alerts among a *cluster*, that is a set of alerts corresponding to the same occurrence of an attack. Alerts issued from clustering are elementary alerts.

Confidentiality. Security property aiming at ensuring the non-disclosure of information to users or processes which are not authorized to access it.

Context. Name given to conditions characterizing the state of a system at a given time. In an information system, context may include information about current accesses to resources, time, location, inventory of the users and resources, configuration and function of the hosts, vulnerabilities, threats, etc.

Correlation. Technique aiming at analyzing elementary alerts in order to characterize intrusion plans, that is high-level non-elementary alerts. Correlation is generally associated with a set of techniques allowing false positives reduction, vulnerability assessment and alert severity mitigation.

Countermeasure. Action, or set of actions, taken in response to any manifestation of threat and changing the state of the system.

Data mining. Technique referring to data flow analysis, especially used in the field of alert correlation, to improve intrusion detection diagnosis.

DAC. Discretionary Access Control is an access control model which consists in the definition of *privileges* of *subjects* on *objects*, through the use of *access control matrix*.

Denial-of-service (DoS). Result of an intrusion leading to the ceasing of a service, *i.e.* loss of availability property.

Dependability. System property which deals with *means* (fault prevention, fault tolerance, fault removal and fault forecasting) to preserve *attributes* (mainly availability, reliability, safety, confidentiality, integrity and maintainability) from *threats* (faults, errors and failures).

Design diversity. Design method which consists in multiple and independently designed implementations (hardware and software) of common system requirements.

Dynamic policy. Policy which presents an implementation different from its definition, considering dynamic parameters, such as current context.

Elementary alert. Alert characterizing an elementary attack.

Elementary attack. Non-decomposable attack, *i.e.* atomic entity of an attack scenario.

Error. Part of the system state resulting from a fault and which may lead to a failure.

Event. Characterization of the information system's activity reported by sensors to analyzers in intrusion detection systems.

Failure. Occurrence appearing when the delivered service deviates from the behavior it has been designed for (correct service).

False negative. No alert in case of unsafe event or set of events.

False positive. Alert in case of safe event or set of events.

Fault. The cause of an error (and possibly of a failure). Faults may either be malicious (attacks), or non-malicious (normal use side-effects).

Fault forecasting. Mean used in dependability for fault characteristics estimations (present number, future incidence, and likely consequences).

Fault tolerance. Mean used in dependability to deliver correct service in the presence of faults.

Fault prevention. Mean used in dependability to prevent the occurrence or introduction of faults.

Fault removal. Mean used in dependability to reduce the number or severity of faults.

Fusion. Correlation technique aiming at creating a higher-level alert considering alerts grouped thanks to aggregation. Produced alerts may have greater granularity than elementary alerts.

IDMEF. Intrusion Detection Message Exchange Format (RFC4765), format used for communications between intrusion detection components.

IDWG. Intrusion Detection Working Group of the Internet Engineering Task Force (IETF), group interested in standardization of Intrusion Detection related works, like IDMEF.

Integrity. Security property aiming at ensuring that information is not manipulated by users or processes which are not authorized to do so.

Intrusion. Successful attack or attack scenario, leading to the violation of the security policy (malicious operational fault).

Intrusion detection. *A posteriori* detection of activity leading to the violation of the security policy, *i.e.* off-line observation of intrusions.

Intrusion prevention. *A priori* detection of a potential violation of the security policy, *i.e.* on-line observation of unexpected or potentially unexpected activity, likely to lead to an intrusion.

Intrusion response. Reaction to an intrusion, and thus to (the) attack(s) leading to the intrusion.

Manager. Intrusion detection component from which the operator manages the intrusion detection system. In particular, the manager includes the possibility to configure sensors and analyzers, alert notification and reporting, and sometimes data consolidation functionalities.

Meta-alert. Sometimes used to qualify high-level alerts, characterized through a correlation process, taking into account multiple events or lower-level alerts which have common characteristics (*e.g.* belonging to a same attack scenario).

Merging. Technique aiming at creating a higher-level alert considering alerts grouped in a cluster. Alert issued from merging are elementary alerts. Compared to clustering, merging allows the reduce the volume of redundant alerts.

Notification. Functionality of the ID manager which purpose is to warn the operator about alert occurrences. This may be achieved through a visualization interface displaying alerts, via e-mail or SMS, etc.

Open policy. “Default permit” policy, *i.e.* what is not explicitly permitted or prohibited is implicitly permitted.

Operator (*a.k.a.* security operator). The human which deals with notifications from the IDS manager, responsible for response initiation.

Or-BAC. Organization-Based Access Control (Or-BAC) is a formal access control model based centered on the concept of organization. It allows the definition of dynamic access control policies, which implementation is based on contexts. Or-BAC distinguishes concrete and abstract level, empowering subjects into roles, considering actions into activities, and using objects into views.

Policy definition. The formal aspect of a security policy, *i.e.* policy rules expressed in an adequate formal policy language.

Policy Decision Point (PDP). Component in charge of deciding how to enforce policy instances. A PDP translate policy instances into configurations to push to PEPs according to their capabilities.

Policy Enforcement Points (PEPs). Components in charge of the policy implementation, *i.e.* current expression of the policy requirements on the information system through their current configuration (*e.g.* a firewall).

Policy instantiation. The actual expression of a security policy, *i.e.* the way it is deployed at a given time on the policy enforcement points (PEP).

Policy Instantiation Engine (PIE). Component in charge of policy compilation according to its definition and potential additional information (especially context). A PIE produces policy instances to push to PDPs.

Probe. Intrusion detection component composed of (a) sensor(s) and an analyzer.

RBAC. Role-Based Access Control is a formal access control model based on the notion of *role*, which allows the abstraction of subjects. Access control is then defined through permissions for *roles* to make *actions* on *objects*.

Reaction. Action, or set of actions, taken in response to any manifestation of threat. Reaction includes all kinds of means to cope with threat, that is notification as well as actions changing the state of the system.

Resource. Any information or service available to subjects on an information system.

Response. Mechanism triggered in to cope with attacks (attack response), intrusions (intrusion response) and more generally threats (threat response).

Risk. An expectation of security policy violation, characterized as the probability that a particular threat will exploit a particular vulnerability leading to an intrusion. The notion of risk is strongly linked with the expected loss brought up by the potential harmful effects resulting from the intrusion.

Security policy. Set of requirements, formal or not, expressing what is authorized or not in term of activity in the information system. Security policies generally deal with three variables: confidentiality, integrity and availability. A security policy may be divided into multiple sub-policies, each describing specific requirements, such as authentication policy, access control policy, encryption policy, etc.

Sensor. Intrusion detection component in charge of data collection. Sensors observe the activity of the information system and report events to the analyzer.

Subject. Any user or process likely to make use of resources in an information system.

Threat. Any manifestation which represents a possibility of violation of the security policy. Faults, errors and failures may be considered as threats, as well as vulnerabilities, attacks and intrusions, as an extension of the notion of fault.

Threat action. Attack.

Threat agent. Subject (user or process) realizing an attack.

Threat consequences. A security violation that results from a threat action.

Threat response. Reaction to a threat, and thus to any occurrence allowing threat characterization (faults, errors and failures, and thus attacks and intrusions).

True negative. No alert in case of safe event or set of events.

True positive. Alert in case of unsafe event or set of events.

Vulnerability. Flaw or weakness in a system (hardware or software) design, implementation or operation and management, which could be exploited to violate the security policy.

Vulnerability assessment. Correlation of alerts with actual vulnerabilities of a considered victim to assess real sensitivity to a considered attack.