

Analysis of Policy Anomalies on Distributed Network Security Setups

J. G. Alfaro^{1,2}, F. Cuppens¹, and N. Cuppens-Boulahia¹

¹ GET/ENST-Bretagne, 35576 Cesson Sévigné - France
{Frederic.Cuppens, Nora.Cuppens}@enst-bretagne.fr

² dEIC/UAB, Edifici Q, 08193, Bellaterra, Barcelona - Spain
Joaquin.Garcia-Alfaro@deic.uab.es

Abstract: The use of different network security components, such as *firewalls* and *network intrusion detection systems* (NIDSs), is the dominant method to survey and guarantee the security policy in current corporate networks. On the one hand, firewalls are traditional security components which provide means to filter traffic within corporate networks, as well as to police the incoming and outgoing interaction with the Internet. On the other hand, NIDSs are complementary security components used to enhance the visibility level of the network, pointing to malicious or anomalous traffic. To properly configure both firewalls and NIDSs, it is necessary to use several sets of filtering and alerting rules. Nevertheless, the existence of anomalies between those rules, particularly in distributed multi-component scenarios, is very likely to degrade the network security policy. The discovering and removal of these anomalies is a serious and complex problem to solve. In this paper, we present a set of algorithms for such a management.

1 Introduction

Generally, once a security administrator has specified a security policy, he or she aims to enforce it in the information system to be protected. This enforcement consists in distributing the security rules expressed in this policy over different security components of the information system – such as firewalls, intrusion detection systems (IDSs), intrusion prevention systems (IPSs), proxies, etc – both at application, system, and network level. This implies cohesion of the security functions supplied by these components. In other words, security rules deployed over the different components must be consistent, not redundant and, as far as possible, optimal.

An approach based on a formal security policy refinement mechanism (using for instance abstract machines grounded on set theory and first order logic)

ensures cohesion, completeness and optimization as built-in properties. Unfortunately, in most cases, such an approach has not a wide follow and the policy is more often than not empirically deployed based on security administrator expertise and flair. It is then advisable to analyze the security rules deployed to detect and correct some policy anomalies – often referred in the literature as *intra- and inter-configuration anomalies* [4].

These anomalies might be the origin of security holes and/or heaviness of intrusion prevention and detection processes. Firewalls [6] and network intrusion detection systems (NIDSs) [12] are the most commonly used security components and, in this paper, we focus particularly on their security rules. Firewalls are prevention devices ensuring the access control. They manage the traffic between the public network and the private network zones on one hand and between private zones in the local network in the other hand. The undesirable traffic is blocked or deviated by such a component. NIDSs are detection devices ensuring a monitoring role. They are components that supervise the traffic and generate alerts in the case of suspicious traffic. The attributes used to block or to generate alerts are almost the same. The challenge, when these two kinds of components coexist in the security architecture of an information system is then to avoid inter-configuration anomalies.

In [7, 8], we presented an audit process to manage intra-firewall policy anomalies, in order to detect and remove anomalies within the set of rules of a given firewall. This audit process is based on the existence of relationships between the condition attributes of the filtering rules, such as coincidence, disjunction, and inclusion, and proposes a transformation process which derives from an initial set of rules – with potential policy anomalies – to an equivalent one which is completely free of errors. Furthermore, the resulting rules are completely disjoint, i.e., the ordering of rules is no longer relevant.

In this paper, we extend our proposal of detecting and removing intra-firewall policy anomalies [7, 8], to a distributed setup where both firewalls and NIDSs are in charge of the network security policy. This way, assuming that the role of both prevention and detection of network attacks is assigned to several components, our objective is to avoid intra and inter-component anomalies between filtering and alerting rules. The proposed approach is based on the similarity between the parameters of a filtering rule and those of an alerting rule. This way, we can check whether there are errors in those configurations regarding the policy deployment over each component which matches the same traffic.

The advantages of our proposal are threefold. First, as opposite to the related work we show in Section 6, our approach not only considers the analysis of relationships between rules two by two but also a complete analysis of the whole set of rules. This way, those conflicts due to the union of rules that

are not detected by other proposals (such as [2, 3, 9]) are properly discovered by our intra- and inter-component algorithms. Second, after applying our intra-component algorithms the resulting rules of each component are totally disjoint, i.e., the ordering of rules is no longer relevant. Hence, one can perform a second rewriting of rules in a *close* or *open* manner, generating a configuration that only contains *deny* (or *alert*) rules if the component default policy is open, and *accept* (or *pass*) rules if the default policy is close (cf. Section 3.1). Third, we also present in this paper a network model to determine which components are crossed by a given packet knowing its source and destination, as well as other network properties. Thanks to this model, our approach better defines all the set of anomalies studied in the related work. Furthermore the lack of this model in other approaches, such as [2, 3], may lead to inappropriate decisions.

The rest of this paper is organized as follows. Section 2 starts by introducing a network model that is further used in Section 3 and Section 4 when presenting, respectively, our intra and inter-component anomaly's classifications and algorithms. Section 5 overviews the performance of our proposed algorithms. Section 6 shows an analysis of some related work. Finally Section 7 closes the paper with some conclusions and gives an outlook on future work.

2 Network Model

The purpose of our network model is to determine which components within the network are crossed by a given packet, knowing its source and destination. It is defined as follows. First, and concerning the traffic flowing from two different zones of the distributed policy scenario, we may determine the set of components that are crossed by this flow. Regarding the scenario shown in Figure 1, for example, the set of components crossed by the network traffic flowing from zone *external network* to zone *private₃* equals $[C_1, C_2, C_4]$, and the set of components crossed by the network traffic flowing from zone *private₃* to zone *private₂* equals $[C_4, C_2, C_3]$.

Let C be a set of components and let Z be a set of zones. We assume that each pair of zones in Z are mutually disjoint, i.e., if $z_i \in Z$ and $z_j \in Z$ then $z_i \cap z_j = \emptyset$. We then define the predicate $connected(c_1, c_2)$ as a symmetric and anti-reflexive function which becomes *true* whether there exists, at least, one interface connecting component c_1 to component c_2 . On the other hand, we define the predicate $adjacent(c, z)$ as a relation between components and zones which becomes *true* whether the zone z is interfaced to component c . Referring to Figure 1, we can verify that predicates $connected(C_1, C_2)$ and $connected(C_1, C_3)$, as well as $adjacent(C_1, DMZ)$, $adjacent(C_2, private_1)$, $adjacent(C_3, DMZ)$, and so on, become *true*.

We then define the set of paths, P , as follows. If $c \in C$ then $[c] \in P$ is an atomic path. Similarly, if $[p.c_1] \in P$ (be “.” a concatenation functor) and $c_2 \in C$, such that $c_2 \notin p$ and $connected(c_1, c_2)$, then $[p.c_1.c_2] \in P$. This way, we can notice that, concerning Figure 1, $[C_1, C_2, C_4] \in P$ and $[C_1, C_3] \in P$.

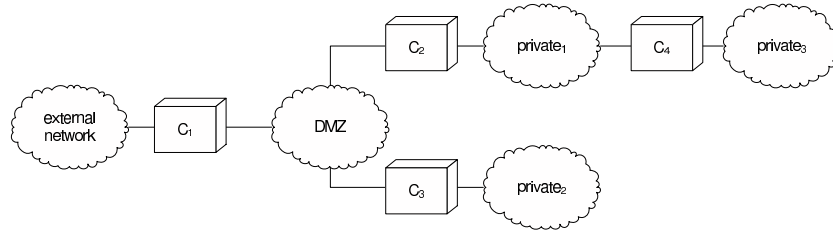


Fig. 1. Simple distributed policy setup.

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

Two additional functions are *route* and *minimal_route*. We first define function *route* from Z to Z in 2^P , such that $p \in route(z_1, z_2)$ iff the path p connects zone z_1 to zone z_2 . Formally, we define that $p \in route(z_1, z_2)$ iff $adjacent(first(p), z_1)$ and $adjacent(last(p), z_2)$. Similarly, we then define *minimal_route* from Z to Z in 2^P , such that $p \in minimal_route(z_1, z_2)$ iff the following conditions hold: (1) $p \in route(z_1, z_2)$; (2) There does not exist $p' \in route(z_1, z_2)$ such that $p' < p$. Regarding Figure 1, we can verify that the *minimal_route* from zone *private3* to zone *private2* equals $[C_4, C_2, C_3]$, i.e., $minimal_route(private_3, private_2) = \{[C_4, C_2, C_3]\}$.

Let us finally conclude this section by defining the predicate *affects*(Z, A_c) as a boolean expression which becomes *true* whether there is, at least, an element $z \in Z$ such that the configuration of z is vulnerable to the attack category $A_c \in V$, where V is a vulnerability set built from a vulnerability database, such as CVE/CAN [11] or OSVDB [13].

3 Intra-Component Analysis

In this section we extend our previous work on analysis of network access control rules for the configuration of firewalls [7, 8], concentrating on anomalies that may also arise in NIDS setups. In particular, a new case of anomaly is pointed out (cf. Intra-Component Irrelevance) and the associated code of our intra-component algorithms has been properly revised³.

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

$$R_i : \{condition_i\} \rightarrow decision_i \quad (1)$$

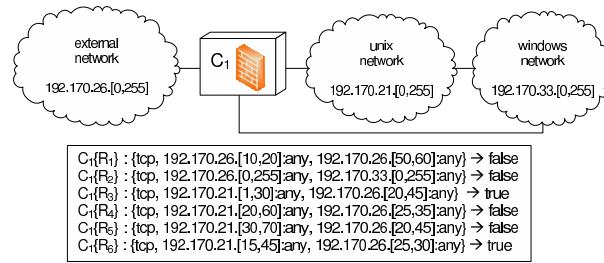
where i is the relative position of the rule within the set of rules, $\{condition_i\}$ is the conjunctive set of condition attributes such that $\{condition_i\}$ equals $C_1 \wedge C_2 \wedge \dots \wedge C_p$ – being p the number of condition attributes of the given rule – and *decision* is a boolean value in $\{true, false\}$.

We shall notice that the decision of a filtering rule will be positive (*true*) whether it applies to a specific value related to *deny* (or *filter*) the traffic it matches, and will be negative (*false*) whether it applies to a specific value related to *accept* (or *ignore*) the traffic it matches. Similarly, the decision of an alerting rule will be positive (*true*) whether it applies to a specific value related to *alert* (or *warn*) about the traffic it matches, and will be negative (*false*) whether it applies to a specific value related to *pass* (or *ignore*) the traffic it matches.

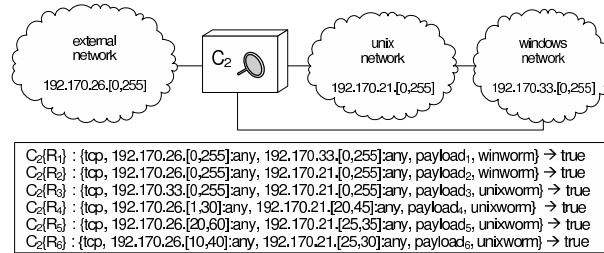
Let us continue by classifying the complete set of anomalies that can occur within a single component configuration. An example for each anomaly will be illustrated through the sample scenario shown in Figure 2.

Intra-Component Shadowing A configuration rule R_i is shadowed in a set of configuration rules R whether such a rule never applies because all the packets that R_i may match, are previously matched by another rule, or combination of rules, with higher priority. Regarding Figure 2, rule $C_1\{R_6\}$ is shadowed by the union of rules $C_1\{R_3\}$ and $C_1\{R_5\}$.

³ Because of the similarity between the revision and the previous work already covered in [7, 8], we move our intra-component audit algorithms to Appendix A. Their correctness and complexity can also be found in [7, 8].



(a) Example scenario of a filtering policy.



(b) Example scenario of an alerting policy.

Fig. 2. Example filtering and alerting policies.

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

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

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

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

(3) The component is a nids, i.e., the predicate $isNIDS(c)$ (cf. Section 2) becomes *true*, and, at least, one of the condition attributes in R_i is related with a classification of attack A_c which does not affect the destination zone of such a rule – i.e., the predicate $affects(z_d, A_c)$ becomes *false*. Regarding Figure 2, we can see that rule $C_2\{R_2\}$ is irrelevant since the nodes in the destination zone *unix network* are not affected by vulnerabilities classified as *winworm*.

3.1 Default policies

Each component implements a positive (i.e., close) or negative (i.e., open) default policy. In the positive policy, the default policy is to *alert* or to *deny* a packet when any configuration rule applies. Conversely, the negative policy will *accept* or *pass* a packet when no rule applies.

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

4 Inter-Component Analysis

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

The main hypotheses to deploy our algorithms hold the following: (1) An upstream network traffic flows away from the closest component to the origin of this traffic (i.e., the most-upstream component [3]) towards the closest component to the remote destination (i.e., the most-downstream component [3]); (2) Every component's policy in the network has been rewritten using the intra-component algorithms defined in Appendix A, i.e., it does not contain intra-component anomalies and the rules within such a policy are completely independent between them.

4.1 Inter-Component Anomalies Classification

In this section, we classify the complete set of anomalies that can occur within a multi-component policy. Our classification is based on the network model pre-

sented in Section 2. An example for each anomaly will be illustrated through the distributed multi-component policy setup shown in Figure 3.

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

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

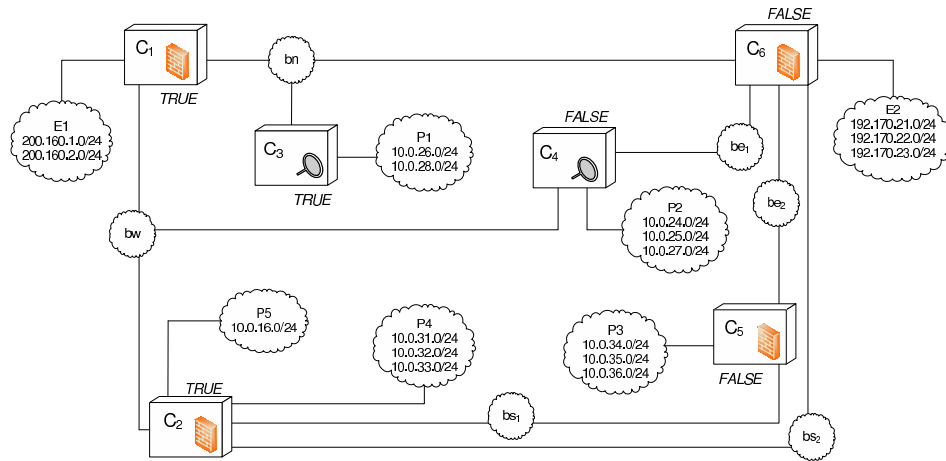
On the other hand, the implicit shadowing between the rule $C_1\{R_5\}$ and the default policy of component C_2 is a proper example of *implicit full shadowing* between two firewalls. Finally, the anomaly between the rule $C_1\{R_6\}$, $C_2\{R_1\}$, and the default policy of component C_2 shows an example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

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

Rules $C_5\{R_3\}$ and $C_6\{R_1\}$ show a proper example of *full redundancy* between two firewalls, whereas rules $C_4\{R_3\}$ and $C_6\{R_5\}$ show an example of *full redundancy* between a firewall and a NIDS. Similarly, rules $C_5\{R_4\}$ and $C_6\{R_2\}$ show a proper example of *partial redundancy* between two firewalls, whereas rules $C_4\{R_4\}$ and $C_6\{R_6\}$ show an example of *partial redundancy* between a firewall and a NIDS.

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

Inter-Component Misconnection A misconnection anomaly occurs between two components whether the following conditions hold: (1) The most-upstream component is a firewall; (2) the most-upstream firewall permits (explicitly, by



$C_1\{R_1\} : (\text{tcp } 200.160.1.[0,255]\text{any}, 10.0.16.[0,255]\text{any}) \rightarrow \text{false}$ $C_1\{R_2\} : (\text{tcp } 200.160.1.[0,255]\text{any}, 10.0.16.[0,255]\text{any}) \rightarrow \text{false}$ $C_1\{R_3\} : (\text{tcp } 192.170.21.[20,33]\text{any}, 200.160.1.[20,30]\text{any}) \rightarrow \text{false}$ $C_1\{R_4\} : (\text{tcp } 192.170.21.[60,80]\text{any}, 200.160.1.[20,30]\text{any}) \rightarrow \text{false}$ $C_1\{R_5\} : (\text{tcp } 10.0.33.[0,30]\text{any}, 200.160.21.[0,30]\text{any}) \rightarrow \text{false}$ $C_1\{R_6\} : (\text{tcp } 10.0.31.[10,20]\text{any}, 200.160.1.[0,255]\text{any}) \rightarrow \text{false}$ $C_1\{R_7\} : (\text{tcp } 10.0.33.[0,255]\text{any}, 200.160.1.[10,12]\text{any}) \rightarrow \text{false}$	$C_2\{R_1\} : (\text{tcp } 10.0.31.[15,17]\text{any}, 200.160.1.[0,255]\text{any}) \rightarrow \text{false}$ $C_2\{R_2\} : (\text{tcp } 10.0.32.[0,70]\text{any}, 10.0.35.[0,255]\text{any}) \rightarrow \text{false}$ $C_2\{R_3\} : (\text{tcp } 10.0.32.[0,70]\text{any}, 200.160.2.[0,255]\text{any}) \rightarrow \text{false}$ $C_2\{R_4\} : (\text{tcp } 10.0.33.[0,255]\text{any}, 200.160.1.[0,255]\text{any}) \rightarrow \text{false}$ $C_2\{R_5\} : (\text{tcp } 10.0.32.[0,70]\text{any}, 10.0.25.[0,255]\text{any}) \rightarrow \text{false}$ $C_2\{R_6\} : (\text{tcp } 200.160.1.[0,2,255]\text{any}, 10.0.16.[0,255]\text{any}) \rightarrow \text{false}$
$C_3\{R_1\} : (\text{tcp } 10.0.31.[15,17]\text{any}, 200.160.1.[0,255]\text{any}, p_{3-1}, C_{3-1}) \rightarrow \text{false}$ $C_3\{R_2\} : (\text{tcp } 10.0.32.[0,70]\text{any}, 10.0.35.[0,255]\text{any}, p_{3-2}, C_{3-2}) \rightarrow \text{false}$	$C_4\{R_1\} : (\text{tcp } 10.0.32.10\text{any}, 10.0.25.[0,255]\text{any}, p_{4-1}, C_{4-1}) \rightarrow \text{true}$ $C_4\{R_2\} : (\text{tcp } 10.0.32.[60,80]\text{any}, 10.0.25.[0,255]\text{any}, p_{4-2}, C_{4-2}) \rightarrow \text{true}$ $C_4\{R_3\} : (\text{tcp } 192.170.22.[15,30]\text{any}, 10.0.24.[0,255]\text{any}, p_{4-3}, C_{4-3}) \rightarrow \text{true}$ $C_4\{R_4\} : (\text{tcp } 192.170.23.[0,255]\text{any}, 10.0.24.[0,255]\text{any}, p_{4-4}, C_{4-4}) \rightarrow \text{true}$ $C_4\{R_5\} : (\text{tcp } 192.170.21.[18,20]\text{any}, 10.0.27.[0,255]\text{any}, p_{4-5}, C_{4-5}) \rightarrow \text{true}$
$C_5\{R_1\} : (\text{tcp } 192.170.22.[0,255]\text{any}, 10.0.34.[0,255]\text{any}) \rightarrow \text{true}$ $C_5\{R_2\} : (\text{tcp } 192.170.23.[18,20]\text{any}, 10.0.34.[0,255]\text{any}) \rightarrow \text{true}$ $C_5\{R_3\} : (\text{tcp } 192.170.21.[10,40]\text{any}, 200.160.1.[0,255]\text{any}) \rightarrow \text{true}$ $C_5\{R_4\} : (\text{tcp } 192.170.21.[65,70]\text{any}, 200.160.1.[0,255]\text{any}) \rightarrow \text{true}$ $C_5\{R_5\} : (\text{tcp } 192.170.22.[0,255]\text{any}, 10.0.24.[0,255]\text{any}) \rightarrow \text{true}$ $C_5\{R_6\} : (\text{tcp } 192.170.23.[18,20]\text{any}, 10.0.24.[0,255]\text{any}) \rightarrow \text{true}$ $C_5\{R_7\} : (\text{tcp } 192.170.21.[10,40]\text{any}, 10.0.26.[0,255]\text{any}) \rightarrow \text{true}$ $C_5\{R_8\} : (\text{tcp } 192.170.21.[65,70]\text{any}, 10.0.26.[0,255]\text{any}) \rightarrow \text{true}$	$C_5\{R_1\} : (\text{tcp } 10.0.32.10\text{any}, 10.0.35.[0,255]\text{any}) \rightarrow \text{true}$ $C_5\{R_2\} : (\text{tcp } 10.0.32.[60,80]\text{any}, 10.0.35.[0,255]\text{any}) \rightarrow \text{true}$ $C_5\{R_3\} : (\text{tcp } 192.170.22.[15,30]\text{any}, 10.0.34.[0,255]\text{any}) \rightarrow \text{true}$ $C_5\{R_4\} : (\text{tcp } 192.170.23.[0,255]\text{any}, 10.0.34.[0,255]\text{any}) \rightarrow \text{true}$ $C_5\{R_5\} : (\text{tcp } 192.170.21.[18,20]\text{any}, 10.0.36.[0,255]\text{any}) \rightarrow \text{true}$

Fig. 3. An example for a distributed network policy setup.

means of negative rules; or implicitly, through its default policy) all the traffic – or just a part of it – that is denied or alerted by a downstream component.

An explicit misconnection anomaly between two firewalls is shown through the rules $C_5\{R_1\}$ and $C_2\{R_2\}$ (*full misconnection*); and the rules $C_5\{R_2\}$ and $C_2\{R_2\}$ (*partial misconnection*). An implicit misconnection anomaly between two firewalls is also shown by the rule $C_1\{R_5\}$ and the default policy of firewall C_2 (*full misconnection*); and the rules $C_1\{R_6\}$ and $C_2\{R_1\}$, together with the default policy of C_2 (*partial misconnection*). Similarly, the pair of rules $C_4\{R_1\}$ - $C_2\{R_5\}$ and the pair of rules $C_4\{R_2\}$ - $C_2\{R_5\}$ show, respectively, an explicit example of full and partial misconnection anomaly between a firewall and a NIDS. Finally, the rule $C_4\{R_5\}$ together with the negative policy of the firewall C_2 shows an example of implicit misconnection anomaly between a firewall and a NIDS.

4.2 Inter-Component Analysis Algorithms

For reasons of clarity, we split the whole analysis process in four different algorithms. The input for the first algorithm (cf. Algorithm 5) is the set of components C , such that for all $c \in C$, we note $c[rules]$ as the set of configuration rules of component c , and $c[policy] \in \{true, false\}$ as the default policy of such a component c . In turn, each rule $r \in c[rules]$ consists of a boolean expression over the attributes $szone$ (source zone), $dzone$ (destination zone), $sport$ (source port), $dport$ (destination port), $protocol$, and $decision$ (true or false).

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

Once in Algorithm 6, we compute the minimal route of components that connects zone z_1 to z_2 , i.e., $[C_1, C_2, \dots, C_n] \in minimal_route(z_1, z_2)$. Then, we decompose the set of components inside each path in downstream path ($path_d$) and upstream path ($path_u$). To do so, we use the implicit functions $head$ and $tail$. The first component $c_d \in path_d$, and the last component $c_u \in path_u$ are passed, respectively, as argument to the last two algorithms (i.e., Algorithm 7 and Algorithm 8) in order to conclude the set of necessary checks that guarantee the audit process⁴.

Let us conclude by giving an outlook in Figure 4 to the set of warnings after the execution of Algorithm 5 over the scenario of Figure 3:

$C_1\{R_3\} - C_6\{R_3, R_4\}$: Full Shadowing	$C_4\{R_2\} - C_2\{R_5\}$: Partial Misconnection
$C_1\{R_4\} - C_6\{R_4\}$: Partial Shadowing	$C_4\{R_3\} - C_6\{R_5\}$: Full Redundancy
$C_1\{R_5\} - C_2\{pol.\}$: Full Shadowing	$C_4\{R_4\} - C_6\{R_6\}$: Partial Redundancy
$C_1\{R_6\} - C_2\{R_1, pol.\}$: Partial Shadowing	$C_4\{R_5\} - C_6\{pol.\}$: Full Misconnection
$C_2\{R_3\} - C_1\{pol.\}$: Full Misconnection	$C_5\{R_1\} - C_2\{R_2\}$: Full Misconnection
$C_2\{R_4\} - C_1\{R_7, pol.\}$: Partial Misconnection	$C_5\{R_2\} - C_2\{R_2\}$: Partial Misconnection
$C_3\{R_1\} - C_6\{R_7, R_8\}$: Full Shadowing	$C_5\{R_3\} - C_6\{R_1\}$: Full Redundancy
$C_3\{R_2\} - C_6\{R_8\}$: Partial Shadowing	$C_5\{R_4\} - C_6\{R_2\}$: Partial Redundancy
$C_4\{R_1\} - C_2\{R_5\}$: Full Misconnection	$C_5\{R_5\} - C_6\{pol.\}$: Full Misconnection

Fig. 4. Execution of Algorithm 5 over the scenario of Figure 3.

⁴ The operator “ \sim ” within algorithms 7 and 8 denotes that two rules r_i and r_j are correlated if every attribute in R_i has a non empty intersection with the corresponding attribute in R_j .

Algorithm 5: inter-component-audit(C)

```

1 foreach  $c \in C$  do
2   foreach  $r \in c[\text{rules}]$  do
3      $Z_s \leftarrow \{z \in Z \mid z \cap \text{source}(r) \neq \emptyset\}$ ;
4      $Z_d \leftarrow \{z \in Z \mid z \cap \text{dest}(r) \neq \emptyset\}$ ;
5     foreach  $z_1 \in Z_s$  do
6       foreach  $z_2 \in Z_d$  do
7         audit( $c, r, z_1, z_2$ );

```

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

```

1 foreach  $p \in \text{minimal\_route}(z_1, z_2)$  do
2    $\text{path}_d \leftarrow \text{tail}(p, c)$ ;
3    $\text{path}_u \leftarrow \text{header}(p, c)$ ;
4   if  $\text{path}_d \neq \emptyset$  and  $r[\text{decision}] = \text{"false"}$ 
5   and isFirewall( $c$ ) then
6      $c_d \leftarrow \text{first}(\text{path}_d)$ ;
7     downstream( $r, c, c_d$ );
8   if  $\text{path}_u \neq \emptyset$  then
9      $c_u \leftarrow \text{last}(\text{path}_u)$ ;
10    if isFirewall( $c_u$ ) then
11      upstream( $r, c, c_u$ );

```

Algorithm 7: downstream(r, c, c_d)

```

1 if  $c_d[\text{policy}] = \text{true}$  then
2    $R_{df} \leftarrow \{r_d \in c_d \mid r_d \vee r \wedge r_d[\text{decision}] = \text{false}\}$ ;
3   if  $R_{df} = \emptyset$  then warning
4     ( $\text{"Full Misconnection"}$ );
5   else if  $\neg \text{testRedundancy}(R_{df}, r)$  then
6     warning( $\text{"Partial Misconnection"}$ );

```

Algorithm 8: upstream(r, c, c_u)

```

1  $R_{uf} \leftarrow \{r_u \in c_u \mid r_u \vee r \wedge r_u[\text{decision}] = \text{false}\}$ ;
2  $R_{ut} \leftarrow \{r_u \in c_u \mid r_u \vee r \wedge r_u[\text{decision}] = \text{true}\}$ ;
3 if  $r[\text{decision}] = \text{"true"}$  then
4   if testRedundancy( $R_{uf}, r$ ) then
5     | warning( $\text{"Full Spurious"}$ );
6   else if  $R_{ua} \neq \emptyset$  then
7     | warning( $\text{"Partial Spurious"}$ );
8   else if testRedundancy( $R_{ut}, r$ ) then
9     | warning( $\text{"Full Redundancy"}$ );
10  else if  $R_{ut} \neq \emptyset$  then
11    | warning( $\text{"Partial Redundancy"}$ );
12  else if  $R_{uf} = \emptyset$  and  $R_{ut} = \emptyset$ 
13  and  $c_u[\text{policy}] = \text{false}$  then
14    | warning( $\text{"Full Misconnection"}$ );
15 else
16   if testRedundancy( $R_{ut}, r$ ) then
17     | warning( $\text{"Full Shadowing"}$ );
18   else if  $R_{ut} \neq \emptyset$  then
19     | warning( $\text{"Partial Shadowing"}$ );
20   else if  $R_{uf} = \emptyset$  and  $c_u[\text{policy}] = \text{true}$  then
21     | warning( $\text{"Full Shadowing"}$ );
22   else if  $\neg \text{testRedundancy}(R_{uf}, r)$ 
23   and  $c_u[\text{policy}] = \text{true}$  then
24     | warning( $\text{"Partial Shadowing"}$ );

```

5 Performance Evaluation

In this section, we present an evaluation of the performance of MIRAGE (which stands for MISconfigURAtion manaGER), a software prototype that implements the intra and inter-firewall algorithms presented in sections 3 and 4. MIRAGE has been developed using PHP, a scripting language that is especially suited for web services development and can be embedded into HTML for the construction of client-side GUI based applications [5]. MIRAGE can be locally or remotely executed by using a HTTP server and a web browser.

Inspired by the experiments done in [3], we evaluated our algorithms through a set of experiments over two different IPv4 real networks. The topology for the first network consisted of a single firewall based Netfilter [16], and a single NIDS based on Snort [15] – connected to three different zones with more than 50 hosts. The topology for the second network consisted of six different components – based on netfilter, ipfilter [14], and snort [15] – protecting six different zones with more than 200 hosts. The whole of these experiments were carried out on an Intel-Pentium M 1.4 GHz processor with 512 MB RAM, running Debian GNU/Linux 2.6.8, and using Apache/1.3 with PHP/4.3 configured.

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

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

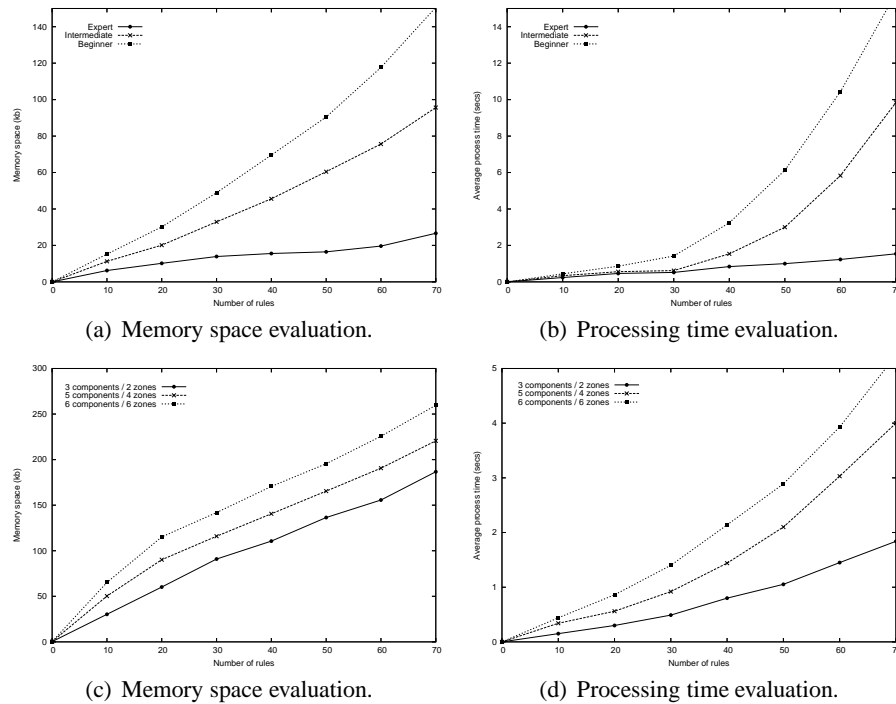


Fig. 5. Evaluation of our set of intra- and inter-component algorithms.

6 Related Work

Some related proposals to our work, such as [1, 9, 2, 10, 3, 4], provide means to directly manage the discovery of anomalies from the components' configuration. For instance, the authors in [1] consider that, in a configuration set, two rules are in conflict when the first rule in order matches some packets that match the second rule, and the second rule also matches some of the packets that match the first rule. This approach is very limited since it just detects a particular case of ambiguity within a single component configuration. Furthermore, it does not provide detection on multiple-component configurations.

In [9], two cases of anomalies are considered. First, a rule R_j is defined as backward redundant iff there exists another rule R_i with higher priority in order such that all the packets that match rule R_j also match rule R_i . Second, a rule R_i is defined as forward redundant iff there exists another rule R_j with the same decision and less priority in order such that the following conditions hold: (1) all the packets that match R_i also match R_j ; (2) for each rule R_k between R_i and R_j , and that matches all the packets that also match rule R_i , R_k has the same decision as R_i . Although this approach seems to head in the right direction, we consider it as incomplete, since it does not detect all the possible cases of intra-component anomalies (as we define in this paper). For instance, given the set of rules shown in Figure 6(a), since R_2 comes after R_1 , rule R_2 only applies over the interval $[51, 70]$ – i.e., R_2 is not necessary, since, if we remove this rule from the configuration, the filtering policy does not change. The detection proposal, as defined in [9], cannot detect the redundancy of rule R_2 within the configuration of such a given firewall. Furthermore, neither [9] nor [10] provide detection on multiple-component configurations.

$R_1 : s \in [10, 50] \rightarrow deny$	$R_1 : s \in [10, 50] \rightarrow accept$
$R_2 : s \in [40, 70] \rightarrow accept$	$R_2 : s \in [40, 90] \rightarrow accept$
$R_3 : s \in [50, 80] \rightarrow accept$	$R_3 : s \in [30, 80] \rightarrow deny$

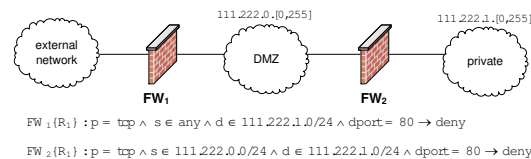
(a) Set of rules A (b) Set of rules B

Fig. 6. Example of some firewall configurations.

To our best knowledge, the approach presented in [2–4] propose the most efficient set of techniques and algorithms to detect policy anomalies in both single and multi-firewall configuration setups. In addition to the discovery process, their approach also attempts an optimal insertion of arbitrary rules into an existing configuration, through a tree based representation of the filtering criteria. Nonetheless, and even though the efficiency of their proposed discovering algorithms and techniques is very promising, we also consider this approach as incomplete. First, their intra- and inter-component discovery approach is not

complete since, given a single- or multiple-component security policy, their detection algorithms are based on the analysis of relationships between rules two by two. This way, errors due to the union of rules are not explicitly considered (as our approach does). The set of rules shown in Figure 6(b), for example, may lead their discovery algorithms to inappropriate decisions. The approach defined in [2] cannot detect that rule R_3 will be never applied due to the union of rules R_1 and R_2 . Just a correlation signal – that is obviously a weaker signal than a shadowing one – would be labeled. Though in [3] the authors pointed out to this problematic, claiming that they break down the initial set of rules into an equivalent set of rules free of overlaps between rules, no specific algorithms have been provided for solving it in [2–4].

Second, their inter-component discovery approach considers as anomalies some situations that, from our point of view, must be suited to avoid inconsistent decisions between components used in the same policy to control or survey to different zones. For instance, given the following scenario:



their algorithms will inappropriately report a redundancy anomaly between filtering rules $FW_1\{R_1\}$ and $FW_2\{R_1\}$. This is because rule $FW_1\{R_1\}$ matches every packet that also $FW_2\{R_1\}$ does. As a consequence, [2] considers rule $FW_2\{R_1\}$ as redundant since packets denied by this rule are already denied by rule $FW_1\{R_1\}$. However, this conclusion is not appropriate because rule $FW_1\{R_1\}$ applies to packets from the external zone to the private zone whereas rule $FW_2\{R_1\}$ applies to packets from the DMZ zone to the private zone. So, rule $FW_2\{R_1\}$ is useful and cannot be removed. Though in [2, 3] the authors claim that their analysis technique marks every rule that is used on a network path, no specific algorithms have been provided for doing so. The main advantage of our proposal over their approach is that it includes a model of the traffic which flows through each component. We consider this is necessary to draw the right conclusion in this case.

Finally, although in [4] the authors consider their work as sufficiently general to be used for verifying many other filtering based security policies such as intrusion detection and prevention systems, no specific mechanisms have been provided for doing so.

7 Conclusions

In this paper we presented an audit process to set a distributed security scenario composed of both *firewalls* and *network intrusion detection systems* (NIDSs) free of anomalies. Our audit process has been presented in two main blocks. We first presented, in Section 3, a set of algorithms for intra-component analysis, according to the discovering and removal of policy anomalies over single-component environments. We then presented, in Section 4, a set of algorithms for inter-component analysis, in order to detect and warn the security officer about the complete existence of anomalies over a multi-component environment.

Some advantages of our approach are the following. First, our intra-firewall transformation process verifies that the resulting rules are completely independent between them. Otherwise, each rule considered as useless during the process is reported to the security officer, in order to verify the correctness of the whole process. Second, we can perform a second rewriting of rules, generating a configuration that only contains positive rules if the component default policy is negative, and negative rules if the default policy is positive. Third, the network model presented in Section 2 allows us to determine which components are crossed by a given packet knowing its source and destination, as well as other network properties. Thanks to this model, our approach better defines all the set of anomalies studied in the related work, and it reports, moreover, two new anomalies (*irrelevance* and *misconnection*) not reported, as defined in this paper, in none of the other approaches. Furthermore, and as pointed out in Section 6, the lack of this model in [2–4] leads to inappropriate decisions.

The implementation of our approach in a software prototype demonstrates the practicability of our work. We shortly discussed this implementation, based on a scripting language [5], and presented an evaluation of its performance. Although these experimental results show that our algorithms have strong requirements, we believe that these requirements are reasonable for off-line analysis, since it is not part of the critical performance of the audited component.

As future work, we are currently studying the anomaly problems of security rules in the case where the security architecture includes firewalls, IDS/IPS, and IPSec devices. Though there is a real similarity between the parameters of those devices' rules, more investigation has to be done in order to extend our proposal. In parallel to this work, we are also considering to extend our approach to the analysis of stateful policies.

Acknowledgements

This work was supported by funding from the French ministry of research, under the *ACI DESIRS* project, the Spanish Government project *TIC2003-02041*, and the Catalan Government grants *2003FI126* and *2005BE77*.

References

1. Adishesu, H., Suri, S., and Parulkar, G. (2000). Detecting and Resolving Packet Filter Conflicts. In *19th Annual Joint Conference of the IEEE Computer and Communications Societies*, 1203–1212.
2. Al-Shaer, E. S., Hamed, H. H. (2004). Discovery of Policy Anomalies in Distributed Firewalls. In *IEEE INFOCOM'04*, March.
3. Al-Shaer, E. S., Hamed, H. H., and Masum, H. (2005). Conflict Classification and Analysis of Distributed Firewall Policies. In *IEEE Journal on Selected Areas in Communications*, 23(10).
4. Al-Shaer, E. S., Hamed, H. H. (2006). Taxonomy of Conflicts in Network Security Policies. In *IEEE Communications Magazine*, 44(3), March.
5. Castagnetto, J. et al. (1999). *Professional PHP Programming*. Wrox Press Inc, ISBN 1-86100-296-3.
6. Cheswick, W. R., Bellovin, S. M., Rubin A. D. (2003). *Firewalls and Internet security: repelling the wily hacker*. Addison-Wesley, second edition.
7. Cuppens, F., Cuppens-Bouahia, N., and Garcia-Alfaro, J. (2005). Detection and Removal of Firewall Misconfiguration. In *Proceedings of the 2005 IASTED International Conference on Communication, Network and Information Security*, 154–162.
8. Cuppens, F., Cuppens-Bouahia, N., and Garcia-Alfaro, J. (2005). Misconfiguration Management of Network Security Components. In *Proceedings of the 7th International Symposium on System and Information Security*, Sao Paulo, Brazil.
9. Gupta, P. (2000). *Algorithms for Routing Lookups and Packet Classification*. PhD Thesis, Department of Computer Science, Stanford University.
10. Gouda, M. G. and Liu, A. X. (2004). Firewall Design: Consistency, Completeness and Compactness. In *24th IEEE International Conference on Distributed Computing Systems (ICDCS-04)*, pages 320–327.
11. MITRE Corp. Common Vulnerabilities and Exposures. [Online]. Available from: <http://cve.mitre.org/>
12. Northcutt, S. (2002). *Network Intrusion Detection: An analyst's Hand Book*. New Riders Publishing, third edition.
13. Open Security Foundation. Open Source Vulnerability Database. [Online]. Available from: <http://osvdb.org/>
14. Reed, D. IP Filter. [Online]. Available from: <http://www.ja.net/CERT/Software/ipfilter/ip-filter.html>
15. Roesch, M. (1999), Snort: lightweight intrusion detection for networks. In *13th USENIX Systems Administration Conference*, Seattle, WA.
16. Welte, H., Kadlecik, J., Josefsson, M., McHardy, P., and et al. The netfilter project: firewalling, nat and packet mangling for linux 2.4x and 2.6.x. [Online]. Available from: <http://www.netfilter.org/>

A Intra-Component Algorithms

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

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

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

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

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

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

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

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

Algorithm 1: exclusion(B, A)	Algorithm 4: intra-component-audit(c, R)
<pre> 1 $C[condition] \leftarrow \emptyset$; 2 $C[shadowing] \leftarrow false$; 3 $C[redundancy] \leftarrow false$; 4 $C[irrelevance] \leftarrow false$; 5 $C[decision] \leftarrow B[decision]$; 6 forall the elements of $A[condition]$ and $B[condition]$ do 7 if $((A_1 \cap B_1) \neq \emptyset)$ and $((A_2 \cap B_2) \neq \emptyset)$ 8 and ... and $(A_p \cap B_p) \neq \emptyset$ then 9 $C[condition] \leftarrow C[condition] \cup$ 10 $\{(B_1 - A_1) \wedge B_2 \wedge \dots \wedge B_p,$ 11 $(A_1 \cap B_1) \wedge (B_2 - A_2) \wedge \dots \wedge B_p,$ 12 $(A_1 \cap B_1) \wedge (A_2 \cap B_2) \wedge (B_3 - A_3) \wedge \dots \wedge B_p,$ 13 \dots 14 $(A_1 \cap B_1) \wedge \dots \wedge (A_{p-1} \cap B_{p-1}) \wedge (B_p - A_p)\}$; 15 else 16 $C[condition] \leftarrow (C[condition] \cup B[condition])$; 17 return C; </pre>	<pre> 1 begin 2 $n \leftarrow count(R)$; 3 /*Phase 1*/ 4 for $i \leftarrow 1$ to $(n - 1)$ do 5 for $j \leftarrow (i + 1)$ to n do 6 if $R_i[decision] \neq R_j[decision]$ then 7 $R_j \leftarrow exclusion(R_j, R_i)$; 8 if $R_j[condition] = \emptyset$ then 9 $warning("Shadowing")$; 10 $R_j[shadowing] \leftarrow true$; 11 /*Phase 2*/ 12 for $i \leftarrow 1$ to $(n - 1)$ do 13 $R_a \leftarrow \{r_k \in R \mid n \geq k > i \text{ and}$ 14 $r_k[decision] = r_i[decision]\}$; 15 if $testRedundancy(R_a, R_i)$ then 16 $warning("Redundancy")$; 17 $R_i[condition] \leftarrow \emptyset$; 18 $R_i[redundancy] \leftarrow true$; 19 else 20 for $j \leftarrow (i + 1)$ to n do 21 if $R_i[decision] = R_j[decision]$ then 22 $R_j \leftarrow exclusion(R_j, R_i)$; 23 if $(\neg R_j[redundancy])$ and 24 $R_j[condition] = \emptyset$ then 25 $warning("Shadowing")$; 26 $R_j[shadowing] \leftarrow true$; 27 /*Phase 3*/ 28 for $i \leftarrow 1$ to n do 29 if $R_i[condition] \neq \emptyset$ then 30 if $testIrrelevance(c, R_i)$ then 31 $R_i[irrelevance] \leftarrow true$; 32 $r[condition] \leftarrow \emptyset$; 33 end </pre>
<pre> Algorithm 2: testIrrelevance(c, r) 1 $z_s \leftarrow source(r)$; 2 $z_d \leftarrow dest(r)$; 3 if $(z_s = z_d)$ and $(\neg r[decision])$ then 4 $warning("First case of irrelevance")$; 5 else if $z_s \neq z_d$ then 6 $p \leftarrow minimal_route(z_s, z_d)$; 7 if $c \notin p$ and $(\neg r[decision])$ then 8 $warning("Second case of irrelevance")$; 9 else if $(\neg empty(r[A_c]))$ and $(\neg affects(z_d, r[A_c]))$ then 10 $warning("Third case of irrelevance")$; 11 else return false; 12 return true; </pre>	
<pre> Algorithm 3: testRedundancy(R, r) 1 $i \leftarrow 1$; 2 $temp \leftarrow r$; 3 while $\neg test$ and $(i \leq count(R))$ do 4 $temp \leftarrow exclusion(temp, R_i)$; 5 if $temp[condition] = \emptyset$ then 6 return true; 7 $i \leftarrow (i + 1)$; 8 return false; </pre>	