A safe Firewall Policy Change Management in a Distributed Environment

Amina Saâdaoui, Nihel Ben Youssef Ben Souayeh and Adel Bouhoula
Digital Security Research Unit, (Sup'Com), University of Carthage, Tunisia
{amina.saadaoui, nihel.benyoussef, adel.bouhoula}@supcom.tn

***Abstract*—Firewall is one of the most commonly used techniques to protect a network, it limits or provides access to specific network segments based on a set of filtering rules that should be configured with respect to the global security policy. Nevertheless, the security policy changes frequently due to business or application needs, and these change often impact firewall configurations by generating new conflicts between different rules. Therefore, discovering and removing automatically the configuration errors is a serious and an unavoidable task. This problem has been addressed through a variety of approaches from firewalls rules analysis to firewall configuration verification, but existent solutions have, essentially, three drawbacks : First, most of them did not analyze all relations between all rules in such a way, some classes of configuration errors could be uncharted. Second, the distinction between syntactic intentional anomalies and effective misconfigurations is not generally highlighted. effective misconfigurations. Third, although anomalies resolution is a tedious and error prone task, it is generally done manually by the network administrator. In this paper, we address this problem using a data structure (FDD: Firewall Decision Diagram). We propose a new approach to detect and correct automatically misconfigurations in a distributed environment. We demonstrate the applicability and scalability of our method by the use of a Satisfiability Solver. The first results we obtained are very promising.**

***Keywords*—Firewall, Security Policy, Misconfiguration, Anomalies, FDD, Distributed environment, Inference system, SAT Solver.**

I. Introduction

The complexity of networks is constantly increasing, as it is the size and complexity of firewall configurations. These Firewalls examine the traffic of network against an ordered list of filtered rules, generally, defined by network administrator according to the global security policy (SP). Over time, the exponential growth in network traffic, services and applications has led to a growth in rule-sets size and a growth in firewall complexity. Moreover, in multi-firewall environment, with hundreds of firewalls, it is difficult to verify detect and correct manually misconfigurations that can arise between different rules. Therefore, after each SP change, automated misconfigurations management to keep firewalls configured properly, without impacting network availability and IT productivity is essential for any enterprise.

In this paper, we consider the following problem: In a multi-firewall environment network, where each firewall can accumulate hundreds of changes over the years, how can we analyze detect and fix firewalls misconfigurations? As an example, consider the network topology shown in Figure 1. We have three firewalls that delimit three subdomains (configurations of firewalls 1, 2 and 3 are shown in Fig.2). The SP that should be implemented is described as follows: **SP_v1 : Deny access from net1 to net2 except http access from machine M1 to subnet21; Allow all traffic from net3 to net2**. We can note here that for traffic from net_1 to net_2, passing through $FW2$ and $FW3$ respectively, the SP is properly implemented. However, if the SP is modified to : **SP_v2 : Allow access from net1 to net2 except http access from machine M1 to subnet21; Deny all traffic from net3 to net2**, then this change requires a complete evaluation of firewalls configurations and we can note that rules of $FW1$, $FW2$ and $FW3$ are no longer implemented with respect to the SP. To deal with this problem, many solutions have been proposed but they have, essentially, the following drawbacks:

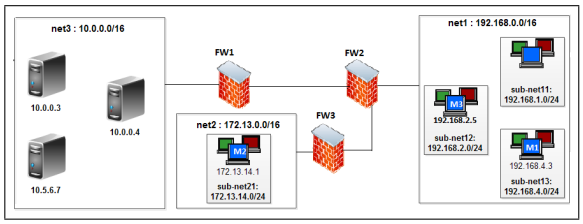

Fig. 1. Network Topology

- In a multi-firewall environment, mostly we consider anomalies between only two firewalls in a given network path which cannot give a precise idea on real conflicts that can arise between different rules of different firewalls and obviously will not help to fix them. In a distributed environment these anomalies could exist between rules of different firewalls in a given path. For example, if we have traffic from net_3 to net_2, the path followed by this traffic contains $Fw1$, $Fw2$ and $Fw3$ so we should analyze relations between rules of these firewalls.
- Some Studies deal only with pairwise filtering rules. In such way, some other classes of configuration anomalies could be uncharted. For instance, if we consider the network path composed by firewalls $Fw1$ and $Fw2$, rule r_1 from $Fw1$ is overlapped with three other rules (r_2 and r_3 from Fw_1 and r_3 from $Fw2$). Therefore any correction technique should take under consideration all relations between all rules otherwise the correction process will not help to obtain the $required$ action.
- Some Studies did not distinct between intentional syntactic anomalies and real configuration errors. For example, we can note that the filtering rules of firewall $Fw3$ are conform to the global security policy SP

Table 1. Firewall Configuration-FW1

Order	Action	Srce_@	Dest_Port	Dest_@	Protocol
r_{11}	accept	10.0.0.3	80	172.13.14.1	*
r_{21}	accept	10.0.0.0/16	80	172.13.14.0/24	TCP
r_{31}	deny	10.0.0.3	*	172.13.14.0/24	TCP
r_{41}	accept	192.168.4.0/24	80	172.13.14.0/24	TCP

Table 2. Firewall Configuration-FW2

Order	Action	Srce_@	Dest_Port	Dest_@	Protocol
r_{12}	deny	192.168.2.0/24	80	172.13.14.0/24	TCP
r_{22}	deny	192.168.1.0/24	80	172.13.14.0/24	TCP
r_{32}	accept	10.0.0.3	80	172.13.0.0/16	TCP
r_{42}	accept	192.168.2.0/24	80	172.13.14.0/24	TCP

Table 3. Firewall Configuration-FW3

Order	Action	Srce_@	Dest_Port	Dest_@	Protocol
r_{13}	accept	192.168.2.0/24	80	172.13.0.0/16	TCP
r_{23}	deny	192.168.1.0/24	80	172.13.14.0/24	TCP
r_{33}	deny	192.168.1.0/24	80	172.13.14.0/24	*
r_{43}	accept	192.168.1.0/24	80	172.13.0.0/16	TCP
r_{53}	accept	192.168.4.0/24	80	172.13.0.0/16	TCP
r_{63}	accept	192.168.4.0/24	80	172.13.0.0/16	TCP

Fig. 2. Firewalls configurations

as filtering rules of firewalls are, generally, processed from the top down, and the first match wins. Here, when traffic from $sub-net11$ tries to access $sub-net21$, it will be blocked through r_{33}. Although no misconfiguration is identified, most related studies [4], [9], [10] present the conflict between rules r_{33} and r_{43} as a purely syntactic anomaly, since these two rules handle common packets with different actions.

- The task of deciding whether an anomaly is a misconfiguration and of fixing it is, generally, left up to the administrator who should update the configuration by ensuring not altering the firewall behavior and not violating SP requirements. This task is more complex than it appears at first glance especially in distributed environment when a large number of firewalls and filtering rules is deployed.

In this paper, we propose a new approach to discover and fix misconfigurations in a multi-firewall environment. Our approach takes advantages of the sequential application of firewall rules modeling their relations in a firewall decision diagram (FDD). This data structure is presented by Gouda and Liu in [14], [19]. Given the structure of FDD and firewall rules, we can discover precisely anomalies and by considering SP we can decide whether an anomaly is intentional or if it is a real configuration error. In this line of research, our earlier work in [23], deals with formal analysis of single firewall configuration. We proposed a new approach to correct misconfigurations in a single firewall by modifying some rules fields, modifying their order, removing some rules. The contributions of this work can be summarized as follows:

- We use a formal method using inference systems and a SAT solver to deal with this problem.
- We extract and decide if an anomaly is a real misconfiguration or an intended anomaly in distributed environment by using the FDD.
- When the usual corrections of configuration errors are the insertion of new filtering rules, which is a practice not often optimal nor safe,We try to meet the challenge of first altering the distributed firewall rules in place (deleting, modifying fields, or even swap rules) to optimize configuration while preserving the behavior of firewalls intact (do not generate other errors).

This paper is organized as follows: Section II presents a summary of related work. Section III overviews the formal representation of firewall configurations and security policies and details FDD structure. In section IV, we articulate our approach to detect and correct firewall misconfigurations. In section V-A, we present the implementation and evaluation of our method. Finally, we present our conclusions and discuss our plans for future work.

II. Related work

A. Intra and Inter-Firewalls anomalies detection

Al shaer et al [4], [3] introduce a framework for discovering anomalies in centralized and distributed firewalls. They analyzed relations between rules using a state diagram that allows to identify anomalies and couple of rules involved in these anomalies or couple of firewalls (in case of inter-firewalls anomalies detection), this differs from our method that considers all rules and not only pairwise ones.
Hu et al [16] propose a new anomaly management framework (FAME) that facilitates the systematic detection and resolution of firewall policy anomalies by considering the analysis of relations between all rules in the firewall configuration. The proposed idea to resolve anomalies is based on calculating a risk level that permits, in some cases, users to manually select the appropriate strategies for resolving the conflict. In such a way, the administrator can make wrong choices. So, the administrator decides manually if an anomaly is a misconfiguration. Unlike that, our method incorporates the security policy which allows deciding, automatically, whether an anomaly is intentional or a real configuration error.
Authors in [9] and [22] introduced a method of analyzing packets from the filtering rule list by using the concept of Relational Algebra and a 2D box model to show a simulation of packets by rectangular boxes and identify anomalies and relations between rules. Abbes et al. present in [1] a method for firewall anomalies discovering. They represent filtering rules in a tree data structure called FAT that allows identifying anomalies between rules two by two. In opposition, in our work we also represent relations between rules in a data structure, but additionally we identify anomalies by considering all rules.
Authors in [1], [8], [16] proposed methods to manage a single firewall rules. This differs from our method that takes into account all firewalls in a given path in the network because even if each firewall in the network is well configured, anomalies could arise between rules of different Firewalls. Prior work on Inter-firewalls rules analysis [3], [13], [7], [15] focused on the analysis of relations between pairwise rules of two firewalls in a given network path. However, in reality it is common that a network path contains more than two firewalls and anomalies could happen between more than two rules in these firewalls. The precise indication of all firewalls and all rules involved in a misconfiguration will help to fix them easily without creating new misconfigurations.
In [21] authors present a firewall analysis engine named Fang, based on a combination of a graph algorithm and a rule-base simulator. FIREMAN [27] is a static analysis toolkit to check anomalies in firewalls. The tool can handle large set of firewall rules since it uses an efficient BDD. This tool can only show there are anomalies between one filtering rule and preceding rules, and cannot identify all rules involved in the anomaly. We

note that most of existing studies [2], [11], [1], [3] focused on the anomaly discovery issue. However, they did not propose methods to resolve these anomalies.
In [25] authors proposed a new approach for correcting anomalies within filtering rules. But the correction is not totally automatic it is assisted by the administrator to yield a required precision in reflecting the adopted security policy.

B. Firewall Configuration verification

Alex X. Liu [18] proposes a firewall verification method. The method takes as input a firewall configuration and a given property, then outputs whether the firewall configuration satisfies the property. Matsumoto and Bouhoula [20] propose a SAT based approach for verifying firewall configurations with respect to the security policy requirements. This method checks the correctness of the firewall configuration whether it contains anomalies or not. Ben Youssef and Bouhoula [26] propose an automatic method for checking whether a firewall is well configured according to a global security policy. This formal method allows verifying automatically that a firewall configuration is conform to a security policy and helping the administrator to correct the configuration errors and to reconfigure the firewall configuration by providing useful information. FINSAT [6], [5] incorporates ACL (Access Control List) conflict analysis procedure for detecting various types of ACL rule conflicts in the model using Boolean satisfiability (SAT) analysis. The conflicts are reported as "error(s)" in case of SAT result with satisfiable instances. Then, the Network administrator need to reconfigure by himself the ACL rules depending on the results.
The objectives of our work are different. We aim first to discover all misconfigurations by considering the requirement of the security policy then to fix these misconfigurations automatically with respect to SP. So, our work involves two aspects: Rule analysis aspect and firewall verification aspect.

III. Preliminaries

A. Firewall configuration

A single firewall configuration is a finite sequence of filtering rules of the form $FC = (r_i \Rightarrow A_i)_{0<i<N+1}$. These rules are tried in order, up to the first matching one. A filtering rule consists of a precondition r_i which is a region of the packet's space, usually, consisting of source address, destination address, protocol and destination port. Each right member A_i of a rule of FC is an action defining the behavior of the firewall on filtered packets: $A_i \in \{accept, deny\}$.

B. Security Policy

A security policy SP is presented as a finite unordered set of directives defining whether packets are accepted or denied. For example, a directive could be as follows:

- A network net1, except the machine A, has the right to access to the FTP service provided by a server S located in the network net2.

We consider also two sets, SP_accept and SP_deny where SP_accept consists of packets accepted to pass through the set of directives SP and SP_deny is the subset of denied packets. In this work we suppose that SP is consistent, i,e. $SP\_accept \cap SP\_deny = \emptyset$.

C. Firewall Decision Diagram (fdd) of a single firewall

The firewall decision diagram (fdd) as defined in [14], [19] is an acyclic and directed graph that has the following properties: There is exactly one node in fdd that has no incoming edges. This node is called the root of fdd. The nodes in fdd that have no outgoing edges are called terminal nodes. fdd is the union of direct paths dp_i. The algorithm used to construct an fdd is detailed in [14], [19]. So we have:

$$fdd = \bigcup_{i(i:1\rightarrow m)} dp_i.$$
$$dp_i = dp_i.srce \wedge dp_i.protocol \wedge dp_i.dest \wedge dp_i.port \wedge dp_i.rules \wedge dp_i.action.$$

- $dp_i.src$ is the range of source address represents by the direct path dp_i.
- $dp_i.dst$ is the range of destination address represents by the direct path dp_i.
- $dp_i.port$ is the range of port number represents by the direct path dp_i.
- $dp_i.protocol$ is the range of protocols represented by the direct path dp_i.
- $dp_i.rules$ is the set of rules from the firewall configuration that match the domain of packets represented by this direct path, $dp_i.rules = \{r_{ki}\}_{(k:1\rightarrow l)}$, where r_{1i} is the first rule in the firewall configuration applied on the domain of dp_i. The action of this direct path is the action applied by r_{1i}.
- $dp_i.action$= the action of this direct path dp_i.

D. FDD of a path in a distributed environment

A network path $path_i[src, dst]$ is composed by an ordered set of firewalls through which the traffic flow from the source src to the destination dst. $path_i = \{fc_j, n <= j <= m\}$. Let $Paths$ be the set of all possible paths in our network. $Paths = \{path_i, 1 <= i <= k\}$.
A FDD of a path $path_i$ is constructed using the collection of rules of different firewalls fc_j belonging to this path. Therefore, The FDD of the set $Paths$ of our network could be represented as follows: $FDD(Paths) = FDD = \bigcup_{\{0<i<N+1\}} fdd_i$, where each fdd_i is the FDD of the path $path_i$, so FDD is the union of fdd_i of each path in the network. We construct fdd_i by using the same algorithm depicted in paragraph III-C for the collection of rules of each $path_i$. The properties already defined for a direct path in a single firewall remains the same, only for sets $dp_i.rules$ and $dp_i.action$. In fact, we have to specify for each rule the firewall that belongs to it. Therefore, we define direct path $dp_j \in fdd_i$ as follows:
$dp_j = dp_j.srce \wedge dp_j.dest \wedge dp_j.port \wedge dp_j.protocol \wedge dp_j.rules \wedge dp_j.action$ where $dp_j.rules = \{r_{h_j}^k\}$ here k is the index of the each firewall through which the traffic flow in the path $path_i$.
The action of each direct path depends on the actions of each first rule handled by this direct path from each firewall in this path, so we have:

- $dp_j.action = accept$ if $\forall r_1{}_j^k \in dp_j.rules$, $action(r_1{}_j^k) = accept$.
- $dp_j.action = deny$ if $\exists! r_1{}_j^k \in dp_j.rules$, $action(r_1{}_j^k) = deny$.

Figures 3 and 4 show, respectively, fdds of two paths: $P1 = Path[net3, net2] = \{Fw1, Fw2, Fw3\}$ and $P2 =$

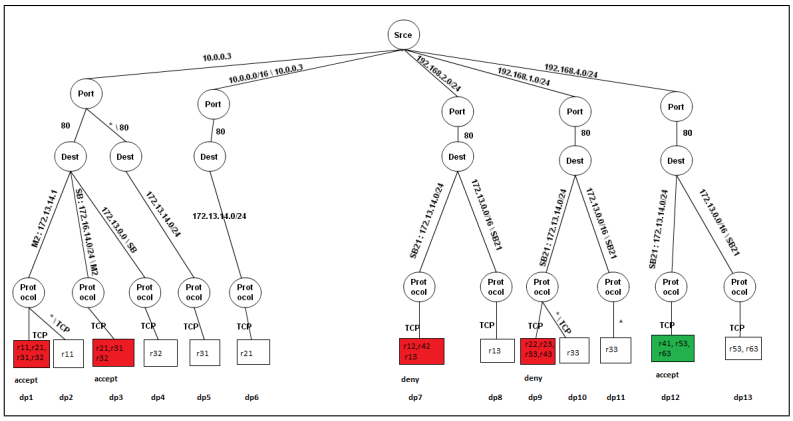

Fig. 3. FDD-P1 (FW1,FW2,FW3)

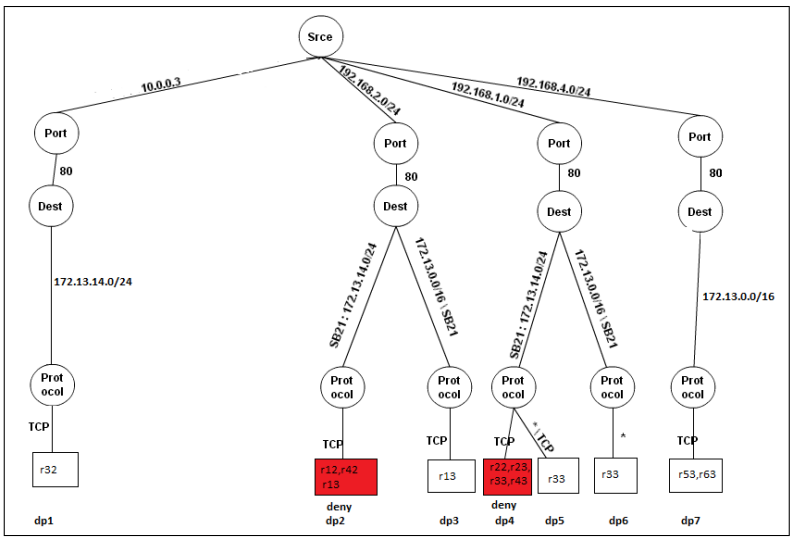

Fig. 4. FDD-P2 (FW2,FW3)

$Path[net1, net2] = \{Fw2, Fw3\}$ for the network shown in Fig. 1.

We consider the following functions:

- $dom(dp_i)$ is a function that maps each dp_i into the subset of packets $p \in P$ and represents the set of packets handled by dp_i. $dom(dp_i) = Packets\{dp_i.srce \wedge dp_i.protocol \wedge dp_i.dest \wedge dp_i.port\}$.
- $idx(r)$ this function returns the index of the rule r.

IV. OUR APPROACH FOR DISCOVERING AND RESOLVING MISCONFIGURATIONS

A. MISCONFIGURATIONS DISCOVERING

In our previous work [24], we presented an Inference system that allows to discover misconfigurations in distributed environment. In fact, in multi-firewalls an anomaly could happen between different firewalls in a network path if they apply different actions on the same traffic. Therefore, by using the data structure FDD already defined for each path, we can determine if a direct path in a given fdd_n contains an anomaly and if this anomaly is a real misconfiguration. So we define anomaly as follows: A direct path $dp_i \in FDD$ presents an anomaly $iff \exists {r_m}_i^k \in dp_i.rules$ where $act({r_m}_i^k) \neq act({r_m}_i^h)$ where $h \neq k$. So we have two types of misconfigurations: Total and partial misconfigurations.

- TMC : A direct path $dp_i \in fdd_n$ is totally misconfigured iff it presents an anomaly and **all** the packets mapped by this path apply a different action as applied in SP on these packets.
- PMC : A direct path $dp_i \in fdd_n$ is partially misconfigured iff it presents an anomaly and **some** packets mapped by this path apply a different action as applied in SP on these packets.

After discovering misconfigurations, we will try to fix them using one of five correction techniques. In the next five subsections, we discuss our approach for each correction technique, respectively. To facilitate the correction process, we define a new set FR_x, *Faulty rules*. For each direct path dp_x, we define the set of faulty rules correspondent to this direct path, we call it FR_x. The set of rules FR_x of each direct path depends on its action, so we have:

- $FR_x = \bigcup_i \{r_i, r_i = {r_1}_x^f \forall f\}$ if $dp_x.action = accept$ and $dp_x \in TMC$.
- $FR_x = \bigcup_i \{r_i, r_i = {r_1}_x^f \wedge action(r_i) = deny \forall f\}$ if $dp_x.action = deny$ and $dp_x \in TMC$.

These two sets define the rules that we **should** fix to correct the action of the direct path. Therefore, for all fixed misconfigurations $FR_x = \emptyset$. We define a new set FR which represents the set of all faulty rules of all totally misconfigured direct paths. $FR = \bigcup_x FR_x \ where \ dp_x \in TMC$.

B. MISCONFIGURATIONS CORRECTION TECHNIQUES

1) REMOVE-RULE INFERENCE SYSTEM: Once all misconfigurations have been discovered, we can start their correction process. First, we will try to correct Total misconfigurations by removing misconfigured rules using the inference system shown in Fig.5. Therefore, we should be sure that removing a given rule will not create other misconfigurations. To address this challenge, we use the FDD of all paths in our network. We can remove a rule only if this rule exists in a decision path as a first rule, then this path is totally misconfigured and the action of second rules in these paths are different from the actions of first rules. So if we remove this rule we will correct all these misconfigurations. The action of misconfigured direct path determines if we should correct one rule or all rules in this direct path. For example, if the misconfigured direct path applies the action accept to the set of packets mapped by this path, then we should fix at least one rule to modify the decision to deny. However, if the applied action is deny, then we should modify the action of all firewalls in the path to accept. Because from a source we should always apply the action accept by each firewall in the concerned path to reach a destination.

The rules of the system shown in Fig.5 apply to triple (FR, FR^*, FR_x) whose first component FR is the set of faulty rules of each dp_x in FDD as defined in the previous section, whose second component FR^* represents an updated version of FR after removing all rules after verifying if the third inference rule $remove$ is applied or not. In fact, the second inference rule $Parse$ allows to parse all faulty rules FR_x of each dp_x, then if the precondition of the third inference rule $remove$ is applied we update the three components FR, FDD and TMC by using the function $\setminus_{remove}$ which allows to modify the three components as follows :

$$Init \ \frac{}{FR, FR^*, \varnothing}$$

$$Parse \ \frac{\{FR_x\} \cup FR, FR^*, \varnothing}{FR, FR^*, FR_x}$$

$$Remove \ \frac{FR, FR^*, \{r_i\} \cup FR_x}{FR \setminus_{remove} \{r_i\}, FR^* \setminus_{remove} \{r_i\}, FR'_x} \quad \text{if (remove-condition)}$$

$$Follow \ \frac{FR, FR^*, \{r_i\} \cup FR_x}{FR, FR^*, FR_x} \quad \text{if remove is not applied}$$

$$Stop \ \frac{\varnothing, FR^*, \varnothing}{FR^*}$$

Fig. 5. Inference system for removing rules

- FR : $\forall x$ we remove r_i from FR_x if $dp_x.action = deny$, else if $dp_x.action = accept$, $FR_x = \emptyset$ because in this case, we should correct at least one rule to obtain the action deny.
- TMC : if $FR_x = \emptyset$ then we remove dp_x from TMC, the direct path has the action required by SP.
- FDD : we remove r_i from each $dp_y.rules$ $\forall y$ and if $FR_y = \emptyset$ we modify the direct path action.

The condition to apply this inference rule is shown in Fig.6. In general, we should verify before removing each rule if the misconfiguration will be fixed and we should be sure that we will not generate new misconfigurations. The inference rule $Follow$ is applied when other inference rules could not be applied. The **Stop** rule is applied when we parse the faulty rules correspondent to totally misconfigured direct paths from the set TMC.

$$if \ \forall dp_y \in FDD, where \{r_i\} \in dp_y.rules \begin{cases} r_1 y^f \neq_f r_i & \vee \\ r_1 y^f =_f r_i \wedge dp_y \in TMC \wedge (\exists r2y^f, act(r2y_f) \neq act(r_i)) & \vee \\ dp_y.act = accept \wedge r_1 y^f =_f r_i \wedge dp_y \notin TMC \wedge (\exists r2y^f, act(r2y_f) = act(r_i)) \end{cases}$$
$$where \begin{cases} FR'_x = \varnothing & if (dp_x.act = accept) \\ FR'_x = FR_x \setminus \{r_i\} & if (dp_x.act = deny) \end{cases}$$

Fig. 6. remove-condition

2) MODIFY-ACTION INFERENCE SYSTEM: After changing the action of a rule we should not generate new misconfigurations. So, we should verify first if **all** the direct paths in all $fdd_n \in FDD$ that have this rule as a first rule are totally misconfigured. If it is the case, we can change the action of the rule under consideration and using this one modification we will correct all misconfigured direct paths that have this rule as a first one.

The rules of the system shown in Fig.7 apply to triple (FR, FR^*, FR_x) whose first component FR is the set of faulty rules of each dp_x in FDD, whose second component FR^* represents an updated version of FR after modifying actions of rules after verifying if the third inference rule $modify$ is applied or not. In fact, the second inference rule $Parse$ allows to parse all faulty rules FR_x of each dp_x, then if the precondition of the third inference rule $modify$ is applied we update the three components FR, FDD and TMC by using the function $\setminus_{modify}$ which allows to modify the three components as follows :

$$Init \ \frac{}{FR, FR^*, \varnothing}$$

$$Parse \ \frac{\{FR_x\} \cup FR, FR^*, \varnothing}{FR, FR^*, FR_x}$$

$$Modify \ \frac{FR, FR^*, \{r_i\} \cup FR_x}{FR \setminus_{modify} \{r_i\}, FR^* \setminus_{modify} \{r_i\}, FR'_x} \quad \text{if (modify-condition)}$$

$$Follow \ \frac{FR, FR^*, \{r_i\} \cup FR_x}{FR, FR^*, FR_x} \quad \text{if modify is not applied}$$

$$Stop \ \frac{\varnothing, FR^*, \varnothing}{FR^*}$$

Fig. 7. Inference system for modifyin rules-actions

- FR : $\forall x$ we remove r_i from FR_x if $dp_x.action = deny$, else if $dp_x.action = accept$, $FR_x = \emptyset$.
- TMC : if $FR_x = \emptyset$ then we remove dp_x from TMC.
- FDD : we modify the action of r_i from each $dp_y.rules$ $\forall y$ and if $FR_y = \emptyset$ we modify the direct path action.

$$if \ \forall dp_y \in FDD, where \{r_i\} \in dp_y.rules \begin{cases} r_1 y^f \neq_f r_i & \vee \\ r_1 y^f =_f r_i \wedge dp_y \in TMC & \end{cases}$$
$$where \begin{cases} FR'_x = \varnothing & if (dp_x.act = accept) \\ FR'_x = FR_x \setminus \{r_i\} & if (dp_x.act = deny) \end{cases}$$

Fig. 8. modify-condition

The condition to apply this inference rule is shown in Fig.8. In general, we check if the rule under consideration verifies two properties in all paths then we can modify the action of the rule. These two properties are: r_i is not the first rule to be applied in direct paths that mapped this rule or it is the first rule and the direct path is totally misconfigured.

3) SWAP-RULES INFERENCE SYSTEM: Before swapping two rules, we need to test and to verify if this modification will generate new misconfigurations between one of the swapped rules and other rules. Also we should verify if this action will correct misconfigurations that exist in paths presented by these two rules. To address this challenge, we use the FDD as the core data structure. An FDD gives us a precise idea if the swap of the rules will correct the misconfigurations or not. In Fig.9 we propose an inference system that presents necessary and sufficient steps to correct total misconfigurations by swapping two rules.

The rules of the system shown in Fig.9 apply to triple (FR, FR^*, FR_x) whose first component FR is the set of faulty rules of each dp_x in FDD, whose second component FR^* represents an updated version of FR after modifying actions of rules after verifying if the third inference rule $swap$ is applied or not. In fact, the second inference rule $Parse$ allows to parse all faulty rules FR_x of each dp_x, then if the precondition of the third inference rule $swap$ is applied we update the three components FR, FDD and TMC by using the function $\setminus_{swap}((rules, rc, ri) :)$ which allows to modify the three components as follows :

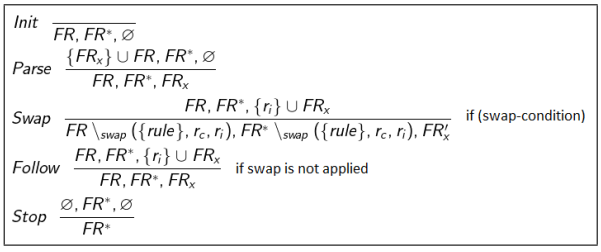

Fig. 9. Swap rules inference system

- FR : $\forall x$ we remove r_i from FR_x if $dp_x.action = deny$, else if $dp_x.action = accept$, $FR_x = \emptyset$.
- TMC : if $FR_x = \emptyset$ then we remove dp_x from TMC.
- FDD : we swap two rules r_c and r_i and if $FR_x = \emptyset$ we modify the direct path action.

The condition to apply this inference rule is shown in Fig.10. We should verify if we can swap rule r_i and rules from the set the set $CL_y(r_i)$ which is the candidate-rules list, rules from this list can be used to correct misconfigurations. In fact, for each dp_y, $CL_y(r_i)$ is composed by rules belonging to the same direct paths as r_i and having r_i as a first rule in these paths also they should have different action to this rule.

4) FIELD MODIFICATION INFERENCE SYSTEM: The rules of the inference system shown in Fig.11 apply to three components (PMC, TMC, FDD). The first component is the set of partial misconfigurations discovered. The second component TMC is the set of total misconfigurations not fixed using methods depicted in previous subsections. The third component FDD is the set of fdd_n of all paths of the network. $Parse\_PMC$ is used to parse the set of partial misconfigurations (i.e., dp_i that have partially an action not exactly the same as defined by SP). This inference rule will divide dp_i into two sets, the first one is the set of paths that have the $correct$ action as defined by SP ($dp_i \setminus SP_{dp_i.act}$), this set will replace the direct path dp_i in FDD, and the second one is the path dp'_i represents the subset of dp_i that should be fixed and will be added to the set of total misconfigurations, by this inference rule we transform the partial problem in dp_i into a $total$ problem (misconfiguration) in dp'_i which will be added to the set TMC. Therefore, the inference rule $Correct$ deals with each direct path from TMC and according to the required action by SP to this direct path we will add new rules in one or some firewalls in this path. The first case, when the required action is $deny$ (i.e., $dp_i.act = accept$), then we add only one rule in the first firewall of this direct path, the rule ${r_1j}^{1'}$ should have the action deny and will be added to the set $dp_i.rules$. The second case, when the required action is $accept$, in this case we have to modify the action of each firewall that have the action deny to accept in this direct path. Therefore, we insert new rules ${r_1j}^{k'}$ in each firewall that applies the action $deny$ on packets handled by the direct path dp_j.

V. Implementation and Evaluation

A. Case study

We have chosen to apply our approach on the case study shown in section I. The global security policy that should be implemented is described as follows: Allow access from net1 to net2 except http access from machine M1 to subnet21; Deny all traffic from net3 to net2.

As defined in section III, $Path$ is the set of all possible paths from a source to a destination by considering SP. In this case, we have: $P1 = Path[net3, net2] = \{Fw1, Fw2, Fw3\}$ and $P2 = Path[net1, net2] = \{Fw2, Fw3\}$. Figures 3 and 4 show, respectively, the FDD of two paths $P1$ and $P2$.

1) Discovering distributed firewalls misconfigurations: We proceed to the discovering of misconfigurations using the inference system previously described in Section IV. We parse all paths of all FDDs, for each path we verify if we have anomaly or not and if this anomaly is an effective misconfiguration:

- In path $P1 = Path[net1, net2] = \{Fw1, Fw2, Fw3\}$: In this path we have four total misconfigurations (colored in red in Fig.3), in direct paths dp_1, dp_3, dp_7 and dp_9. Also we have a partial misconfiguration in direct path dp_{12} (Colored in green), in fact, the traffic from machine M1 192.168.4.3 will be accepted by direct path dp_{12} even if we precisely indicated in SP that this traffic should be rejected, this misconfiguration is partial because other traffic from $net1$ will be allowed which is conform to SP. The SP is **partially** violated in this case.
- In path $P2 = Path[net3, net2] = \{Fw2, Fw3\}$: In this path we have two total misconfigurations (colored in red in Fig.4), in direct paths dp_2 and dp_4.

2) Distributed firewalls misconfigurations resolution: After discovering process has been established, we will proceed in this section to the resolution of these misconfigurations automatically and in contrast with SP.

TMC in dp_1 in $P1$: According to the process of correction explained in section IV, the set of faulty rules FR of this direct path contains rules r_{11}, r_{32}, the correct action is deny, therefore we can fix at least one of these rules to fix this total misconfiguration. So, we start by verifying if we can remove the rule r_{11}, it is not the case because r_{21} have the same action as r_{11}, so removing r_{11} will not fix the problem in this direct path and it is the same case for rule r_{32}. Then, we verify if we can modify the action of these rules, we note that rules r_{11} and r_{32} exist in other direct paths and

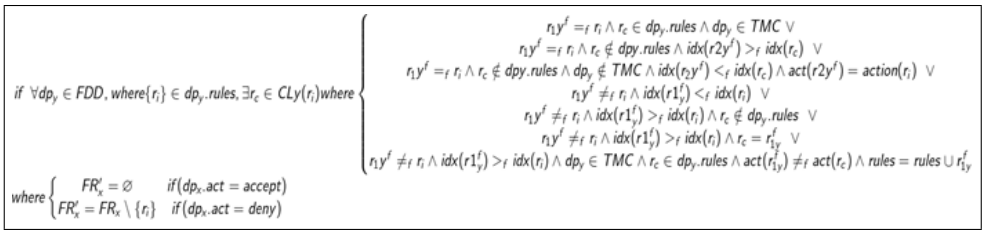

Fig. 10. Swap condition

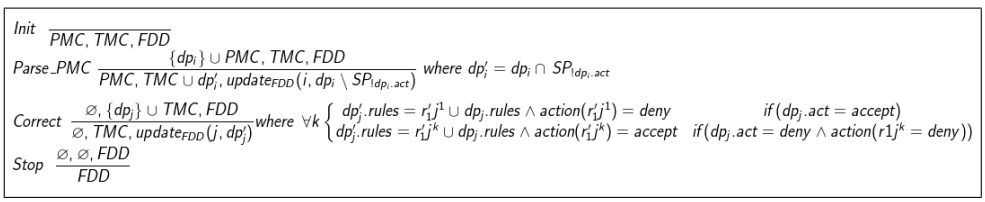

Fig. 11. Field modification Inference system

these paths does not present any anomaly. Therefore, we try to apply the swap inference system, the set of candidate rules $CL = \{r_{31}\}$, according to the FDD swapping r_{11} and r_{31} will not only correct this misconfiguration but also the second misconfiguration in dp_3 in path $P1$ and will not generate new misconfigurations. Therefore, for these two misconfigurations we will use the swap-technique.

TMC in dp_7 **in** $P1$ **and** dp_2 **in** $P2$: r_{12} exists only in dp_7 from $P1$ and dp_2 from $P2$ and these two direct paths are totally misconfigured. The set of faulty rules of these direct paths contains rule r_{12} only, also the second rule in these direct paths is the rule r_{42} which have a different action from r_{12}, so by removing this rule (i.e., r_{12}) we will correct these two misconfigurations and we will not generate new misconfigurations.

TMC dp_9 **in** $P1$ **and** dp_4 **in** $P2$: The set of faulty rules of these two direct paths contains rules r_{22} and r_{23}. We should correct these two rules because they have the action deny. According to the process of correction explained in section IV, we should start by verifying if we can remove these rules, it is the case for rule r_{22} but not for rule r_{23} because r_{33} have the action deny, so removing r_{23} will not fix the problem. So, we remove r_{22}. Then, for rule r_{23} we verify if we can modify the action of this rule, we note that r_{23} exists only in these misconfigured direct paths. So by changing the action of this rule (i.e., r_{23}) we will correct these misconfiguration and we will not generate new misconfigurations.

PMC in dp_{12} **from path** $P1$: This misconfiguration is *partial* so we use the method "field modification" to fix this problem. The intersection between DP_{12} ad SP_{deny} can be represented as follow : $BSP = DP_{12} \cap SP_{deny} =$ branch represented by these values : $[@src e, port, @dest, protocol] =$ $[192.168.4.3, 80,$
$172.13.14.0/24, TCP]$ Therefore, DP_{12} could be represented as follow: $DP_{12} = (DP_{12} \backslash BSP) \cup (DP_{12} \cap BSP)$. Then using our inference system shown in Fig.11 we use first the inference rule $Parse$ to divide this direct path into two sub-FDDs where the first $(DP_{12} \backslash BSP)$ represents paths which are conform to SP and the second one $DP_{12} \cap BSP$ is the totally misconfigured path. Then to correct $DP_{12} \cap BSP$ we use $Correct$, this inference rule will add new rules with new action at each direct path that contains the total misconfiguration.

B. Tool Evaluation

1) Complexity: For n rules in FC, there can be a maximum of $2n-1$ outgoing edges for a node. Therefore, the maximum number of paths in a constructed FDD is $(2n-1)^d$, where d is the number of fields in each rule. After the construction of FDD, the process of misconfigurations discovering and removing, is done on direct paths elements $dp_i.rules$. Therefore, for our inference systems, the complexity (without counting the elementary functions) is equivalent to the complexity of operations in an ordered list. Thus, in our case, the complexity of each inference system is equal to $O(n^d)$. Given that d is typically small (generally we have 4 or 5 fields) our inference systems have a reasonable response time in practice.

2) Implementation and Experimental results: In order to better assess the effectiveness of our approach, we implemented the techniques and inference systems described earlier in a software tool, using a Boolean satisfiability (SAT) based approach. This approach reduces the verification problem into Boolean formula and checks its satisfiability. In our case, in order to verify if an anomaly is a partial or total misconfiguration, we verify if the domain of the direct path reduced into Boolean formula is included or not in the domain of SP reduced into two sub-domains SP_deny and SP_accept as explained in the section III. We have chosen also the Java developing language. On the other hand, the verification of the satisfiability of Boolean expressions is performed using Limboole [17]. This tool allows to check satisfiability respectively tautology on arbitrary structural formulas and not just satisfiability for formulas in conjunctive normal form (CNF), and can handle large set of non-quantified Boolean clauses in reasonably good time. To evaluate a practical value of our inference systems, we have implemented them based on the FDD approach and we tested our developed tool using the rule collections of the open-source rules available at emerging threats (ETOpen) rule sets [12]. By default, all generated and distributed rules in [12] are deny (or DROP) rules because it is considered to be the safest way for distribution and it is up to each user to adjust them as needed for their network's needs. Our tool demonstrate the applicability and scalability of

proposed inference systems, we have also conducted a set of experiments to measure their performance, our tool has proved a stable performance showing acceptable processing time (the average processing time is some seconds) to the treatment of complex combination of thousands filtering rules.
We have also conducted a set of experiments to measure the performance of our inference systems. The experiments were run on an Intel Dual core 1.6 GHz with 2 Gbyte of RAM. It supposes that we have IPv4 addresses with net-masks and port numbers of 16 bit unsigned integer with range support. Figure 12 summarizes our results. We consider time treatment factor that we review by varying the number of rules. In overall terms, we consider the average processing time, in seconds, of the main procedures of FDD construction, misconfigurations detection and correction. In the end, our tool has proved a stable performance showing acceptable processing time to the treatment of complex combination of filtering rules.

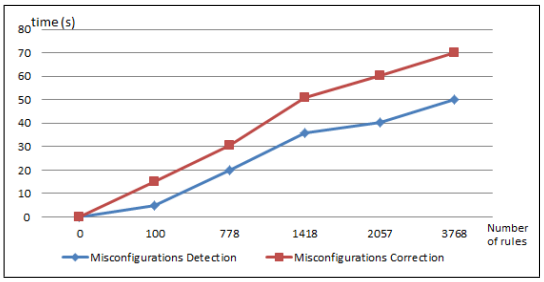

Fig. 12. Processing time

VI. Conclusion

The prevalent use of firewalls in network security emphasizes the importance of efficient and optimal configuration. This paper describes two problems. The first, is firewall misconfiguration discovering. In fact, we propose a method to discover and distinct real configuration errors. The second is misconfigurations resolution by using a formal method and a data structure (FDD). Specifically, we presented a classification of misconfigurations (total or partial) and propose a set of inference systems that allow optimal and safe correction of these conflicts, without generating new misconfigurations, through the analysis of the rule relations basing on FDD structure. The efficacy and scalability of our approach has been demonstrated and the first results we obtained are very promising. While the current approach primarily focuses on the detection and correction of firewalls configuration errors. As a future work, we are working on extending our approach in order to handle other network security components misconfigurations like IDS.

References

[1] Tarek Abbes, Adel Bouhoula, and Michael Rusinowitch. Detection of firewall configuration errors with updatable tree. *International Journal of Information Security*, pages 1–17, 2015.

[2] Hari Adiseshu, Subhash Suri, and Guru M. Parulkar. Detecting and resolving packet filter conflicts. In *Proceedings IEEE INFOCOM 2000, The Conference on Computer Communications, Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies, Reaching the Promised Land of Communications, Tel Aviv, Israel, March 26-30, 2000*, pages 1203–1212, 2000.

[3] Ehab S. Al-shaer and Hazem H. Hamed. Discovery of policy anomalies in distributed firewalls. In *IN IEEE INFOCOM04*, 2004.

[4] Ehab S. Al-Shaer and Hazem H. Hamed. Modeling and management of firewall policies. *IEEE Transactions on Network and Service Management*, 1(1):2–10, 2004.

[5] P. Bera, S.K. Ghosh, and P. Dasgupta. Policy based security analysis in enterprise networks: A formal approach. *Network and Service Management, IEEE Transactions on*, 7(4):231–243, 2010.

[6] Padmalochan Bera, Santosh K. Ghosh, and Pallab Dasgupta. Integrated security analysis framework for an enterprise network - a formal approach. *IET Information Security*, 4(4):283–300, 2010.

[7] A. Bouhoula and Z. Trabelsi. Handling anomalies in distributed firewalls. In *Innovations in Information Technology, 2006*, pages 1–5, Nov 2006.

[8] Adel Bouhoula, Zouheir Trabelsi, Ezedin Barka, and Mohammed Anis Benelbahri. Firewall filtering rules analysis for anomalies detection. *IJSN*, 3(3):161–172, 2008.

[9] Thawatchai Chomsiri and Chotipat Pornavalai. Firewall rules analysis. In *Security and Management*, pages 213–219, 2006.

[10] Frédéric Cuppens, Nora Cuppens-Boulahia, and Joaquin Garcia Alfaro. Detection and removal of firewall misconfiguration. In *CNIS IASTED, Phoenix, AZ, USA novembre*, 2005.

[11] David Eppstein and S. Muthukrishnan. Internet packet filter management and rectangle geometry. *CoRR*, cs.CG/0010018, 2000.

[12] Etopen ruleset, 2016.

[13] Joaquín García-Alfaro, Frédéric Cuppens, and Nora Cuppens-Boulahia. Analysis of policy anomalies on distributed network security setups. In *Computer Security - ESORICS 2006, 11th European Symposium on Research in Computer Security, Hamburg, Germany*, 2006.

[14] Mohamed G. Gouda and Alex X. Liu. Structured firewall design. *Computer Networks Journal (Elsevier)*, 51(4):1106–1120, March 2007.

[15] S. Hall, L. Ngoup, R. Villemaire, and O. Cherkaoui. Distributed firewall anomaly detection through ltl model checking. In *Integrated Network Management (IM 2013), 2013 IFIP/IEEE International Symposium on*, pages 194–201, May 2013.

[16] Hongxin Hu, Gail-Joon Ahn, and Ketan Kulkarni. Detecting and resolving firewall policy anomalies. *IEEE Transactions on Dependable and Secure Computing*, 9(3):318–331, 2012.

[17] Limboole sat solver, 2016.

[18] Alex X. Liu. Formal verification of firewall policies. In *ICC*, pages 1494–1498, 2008.

[19] Alex X. Liu and Mohamed G. Gouda. Diverse firewall design. *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 19(8), 2008.

[20] Soutaro Matsumoto and Adel Bouhoula. Automatic verification of firewall configuration with respect to security policy requirements. In *CISIS*, pages 123–130, 2008.

[21] Alain Mayer, Avishai Wool, and Elisha Ziskind. Fang: A firewall analysis engine. In *Proceedings of 2000 IEEE Symposium on Security and Privacy*, pages 177–187, 2000.

[22] Naveen Mukkapati and Ch.V.Bhargavi. Detecting policy anomalies in firewalls by relational algebra and raining 2d-box model. *IJCSNS International Journal of Computer Science and Network Security*, 13(5):94–99, 2013.

[23] Amina Saadaoui, Nihel Ben Youssef Ben Souayeh, and Adel Bouhoula. Automated and optimized fdd-based method to fix firewall misconfigurations. In *14th IEEE International Symposium on Network Computing and Applications, NCA 2015, Cambridge, MA, USA, September 28-30, 2015*, pages 63–67, 2015.

[24] Amina Saadaoui, Nihel Ben Youssef Ben Souayeh, and Adel Bouhoula. An accurate fdd-based approach for discovering distributed firewalls misconfigurations. In *2016 IEEE Trustcom/BigDataSE/ISPA, Tianjin, China, August 23-26, 2016*, pages 552–559, 2016.

[25] Anis Yazidi and Adel Bouhoula. On assisted packet filter conflicts resolution: An iterative relaxed approach. In *41st IEEE Conference on Local Computer Networks, LCN 2016, Dubai, United Arab Emirates, November 7-10, 2016*, pages 35–42, 2016.

[26] Nihel Ben Youssef, Adel Bouhoula, and Florent Jacquemard. Automatic verification of conformance of firewall configurations to security policies. In *ISCC*, pages 526–531, 2009.

[27] Lihua Yuan, Jianning Mai, Zhendong Su, Hao Chen, Chen-Nee Chuah, and Prasant Mohapatra. Fireman: A toolkit for firewall modeling and analysis. In *Proceedings of the 2006 IEEE Symposium on Security and Privacy*, Washington, DC, USA, 2006. IEEE Computer Society.