Automatic Firewalls' Configuration using Argumentation Reasoning

Erisa Karafili[1] $^{[0000-0002-8250-4389]}$ and Fulvio Valenza[2]

[1] Electronics and Computer Science, University of Southampton, UK
e.karafili@soton.ac.uk
[2] Department of Control and Computer Engineering, Politecnico di Torino, Italy
fulvio.valenza@polito.it

Abstract. Firewalls are widely used as the first frontier to protect the network from intrusions, vulnerability exploitations, and cyber-attacks. Usually, the configuration of this critical component of network security is done manually by network administrators that introduce human errors. In this paper, we present an automatic tool that is based on a formal framework, called *ArgoFiCo*. Our tool automatically configures the distributed firewalls of the network by generating conflict-free firewalls' configuration. *ArgoFiCo* is based on abduction and argumentation reasoning and it permits the identification and resolution of anomalies in firewalls. Our tool provides an answer to the human error problem as it automatically populates the firewalls of a network, given the network topology and the high-level requirements of the network behaviour.

1 Introduction

Firewalls are widely used as the first frontier to protect the network from intrusions, vulnerability exploitations, and cyber-attacks. The firewalls are mainly configured by network administrators and these configurations suffer from human errors. The impact of the human error in network security is significant, as nearly 60% of the security breaches that occurred in 2019 were due to humans errors made by systems and network administrators [24].

The network administrators have the difficult task to ensure the networks from security breaches, and this task requires very specific skills and competencies. Their typical approach is *trial and error* by creating ad-hoc rules to correct the reported misconfigurations. This solution is not sustainable in the long run, especially when there are different network administrators. Thus, there is a need for an automatic tool that evaluates the enforced policies [2] and guarantees the networks from misconfigurations.

We introduce a tool that automatically configures conflict-free firewalls. In particular, this tool is based on a formal framework, called *ArgoFiCo*, that uses abductive and argumentation reasoning. Our tool, given as input the network topology and high-level network behaviour requirements, provides as result the firewalls' configurations.

The used formal framework, *ArgoFiCo*, uses *preference-based argumentation* reasoning [12,10], which permits us to deal with conflicting policies and order them by introducing preferences. In particular, our tool uses Gorgias [10] that is a preference-based argumentation reasoning tool with abduction. The anomalies identified are automatically solved by removing the unnecessary rules or proposing a novel anomaly free re-ordering based on the introduced preferences. The re-ordering is provided by following various resolution strategies. Our framework simplifies the network administrators' work, by automatically configuring the firewalls of the network. Furthermore, as the process is automatic, it reduces the chances of human errors. We use a similar methodology as the one provided in [23] for analyzing the network behaviour rules and for automatically identifying the anomalies between them.

Our tool provides to the user together with the results also an *explanation*. This explanation gives new insights to the administrator that can now check if the information provided is correct, or if further adjustments are needed. Our tool provides flexibility to the administrator that can decide between two ordering strategies (that we call configurations). The tool permits the administrator to configure the firewalls of the network to allow or deny the flow of information in all possible paths between source and destination, called *max configuration*. The second type of configuration is more restrictive, as it configures the firewalls of the network to allow or deny the traffic only for the path derived from the routing between the source and destination, called *min configuration*.

We introduce the related work for the firewall configuration in Section 2. In Section 3, we present the used formal framework. We show the main components of our tool in Section 4 and provide the main results in Section 5. We conclude and describe some future research directions in Section 6.

2 Related Work

In this section, we briefly report the most important related work on *policy analysis* and *policy refinement*.

2.1 Policy Analysis

The main goal of policy analysis is to detect *anomalies* in the policies of network security controls like firewalls, VPN, IDS. Specifically, this anomaly analysis[3] looks for incorrect policy specifications that administrators may introduce in the network. In our view, an anomaly can be potential errors, conflicts, and sub-optimizations affecting either a single policy or a set of security policies [22].

One of the most important works on policy analysis with filtering configurations was proposed by *Al-Shaer et al.* in [1]. In this work, the authors classify and analyze the local anomalies, arising in a single firewall (*intra-firewall*

[3] Sometimes anomaly analysis is also referred to as either conflict analysis or policy validation.

anomaly) and the global anomalies that arise in distributed firewalls (*inter-firewall* anomaly). Another interesting work that performs a policy analysis is introduced in [3], where the authors use an argumentation framework for specifying security requirements. In this work, the authors focus on analyzing the requirements and identifying their orderings. The work does not deal with the configuration of different firewalls inside the same network and with inter-firewall anomalies.

Another work [20] deals only with redundant policies. It focuses on detecting and removing redundant filtering rules with a data-structure named FDD (Firewall Decision Diagram). A formal model for detecting anomalies among different security functions (inter-function anomalies) is introduced in [23,6]. The proposed model detects several kinds of errors and anomalies that originate from correlations between configuration rules of different network functions. This approach is able to cope with different network functions, such as firewalls, NAT/NAPT devices, traffic monitors, and encryption devices.

In this paper, we introduce a solution that uses a similar technique as [3] (i.e., argumentation reasoning) but our solution is able to work with different firewalls and to capture and resolve also inter-firewall anomalies. Furthermore, we use the same definitions of anomalies as in [1] but we apply different resolution techniques, as we use argumentation reasoning, while [1] uses FOL formulas. Our solution is also able to deal with redundant rules and the conflicts/anomalies identification is similar to the one introduced in [23].

2.2 Policy Refinement

The *policy refinement* is the process that automatically "determines the resources needed to satisfy policy requirements and translates high-level policies into operational policies that may be enforced by the system" [21]. Policy refinement is a critical task that may lead to either incorrect or sub-optimal implementations that affect network performance and can jeopardise the overall system security.

The most significant frameworks for automatic configuration of packets filtering firewalls based on policy refinement are FIRMATO [4], FACE [25], MIRAGE [9], and VEREFOO [7]. Moreover, a refinement model that allows the translation of high-level security requirements into low-level configuration settings for the virtual network security functions was introduced in [5].

All the above introduced frameworks perform refinement using FOL (i.e, First-Order Logic), while the solution we propose in this paper uses argumentation reasoning for the policy refinement.

3 Automatic Firewalls' Configuration

Our tool automatically configures the firewalls of the network, given the network topology and the behaviour requirements of the network. The input is provided

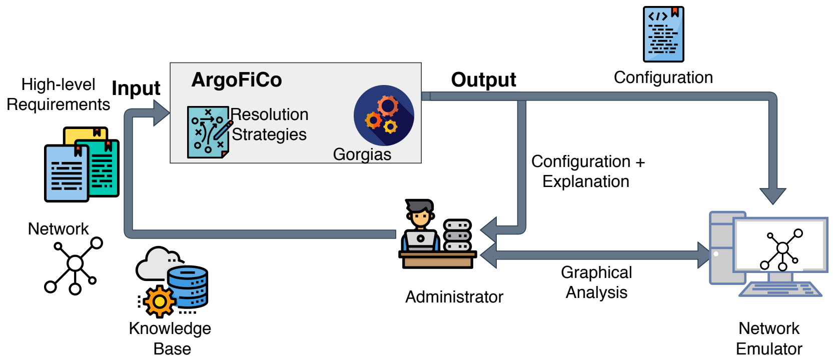

Fig. 1. An overview of our framework for Automatic Firewalls' Configuration via Argumentation Reasoning (ArgoFiCo)

by the users that are the network administrators. The outputs are the constructed firewalls' configurations that are anomaly free, i.e., no inter- or intra-firewall anomalies. An overview of the main workflow of our tool is provided in Figure 1. The introduced tool is based on a formal framework, called *ArgoFiCo* that uses abduction and argumentation reasoning. The theoretical formalism used by *ArgoFiCo* is introduced in Section 3.1 and a brief introduction of *ArgoFiCo*'s preliminary results on the firewalls' configuration problem are given in [17].

Let us briefly describe how our tool operates. It starts by analyzing the high-level requirements of the network (provided in the form of behavioural rules). This analysis identifies possible existing conflicts and anomalies. The analysis, identification, and resolution of the anomalies is performed automatically using *ArgoFiCo*, as the use of argumentation permits us to identify the conflicting rules and resolve the conflicts. The next step is the resolution of anomalies, which is done by removing or reordering the rules and by taking into account also the network topology and the resolution strategies. Once the rules are ordered, then the next step is the population of all the firewalls. The population is done by avoiding the anomalies and conflicts between rules of different firewalls. The final steps of our tool are the translation of the rules into low-level firewall rules and the generation of the firewalls' configurations.

3.1 Preference-Based Argumentation Reasoning

The used formalism by *ArgoFiCo* is preference-based argumentation [12,10], which permits us to work with conflicting rules, given its *non-monotonic* nature. *ArgoFiCo* uses the Gorgias [10] tool that is a preference-based argumentation reasoning tool, which uses abduction [11]. We decided to base *ArgoFiCo* on argumentation reasoning, given the extended applicability of this reasoning, e.g., policy analysis [3], secure data sharing [13,14,16], swarm of drones [15,8], and cyber investigations [18,19].

An *argumentation theory* is a pair $(\mathcal{T}, \mathcal{P})$ of argument rules $\mathcal{T}$ and preference rules $\mathcal{P}$. The argument rules $\mathcal{T}$ are a set of labelled formulas of the form: $rule_i : L \leftarrow L_1, \ldots, L_n$ where $L, L_1, \ldots, L_n$ are positive or negative ground literals, and $rule_i$ is the label denoting the rule name. In the above argument rule, L denotes the *conclusion* of the argument rule and $L_1, \ldots, L_n$ denote its *premises*. The premise of an argument rule is the set of conditions required for the conclusion to be true. Instead, for $\mathcal{P}$ the head L is of the form $rule_1 > rule_2$, where $rule_1$, $rule_2$ are labels of rules defined in $\mathcal{T}$, and $>$ refers to an irreflexive, transitive, and antisymmetric higher priority relation between the rules. The priority rule $rule_1 > rule_2$ means that $rule_1$ has *higher priority* over $rule_2$, or better $rule_1$ is preferred over $rule_2$. The priority rule is true always or under certain contexts or conditions. The premise of the rule describes the context where the priority rule is true and when the premise is empty it means that the priority rule is always true.

Our approach uses abduction [11]. The ability to work with incomplete information makes *abduction* suitable to capture anomalies in case not all the rules are provided. By introducing a set of *abducible* predicates Ab, abduction allows us to make assumptions $\triangle$ ($\triangle \subseteq Ab$), for reaching a conclusion L, as long as the assumptions $\triangle$ satisfy the *integrity constraints*. Integrity constraints are conditions that every admissible set of arguments must satisfy.

In our framework the preference-based reasoning rules are the high-level requirements of the network and are denoted as follows:

$$req(\text{allow/deny, source, destination, type of traffic}).$$

The premises of the above rule are the *source*, *destination*, *type* of traffic, and information related to the location of the source and destination, and the conclusion is the decision of *allowing* or *denying* the traffic. The preferences between rules are represented with the order between the rules. In particular, the priority rules permit us to order the rules and to have a total order for the set of rules. The preferences are introduced automatically by *ArgoFiCo*, by applying the conflicts resolution strategies.

3.2 Inputs and Outputs

In this section, we describe more in detail the inputs and outputs of our tool. Currently, the inputs are provided by the network administrator/s and in the future, we plan on fully automating this process. The provided inputs are the network behaviour policies, which are given in the form of high-level requirements of the network, the network topology, which includes the components of the network together with their containing relations, and also the resolution strategies. The user can decide the strategies or in case no strategy is provided by the user, a "default" mode is selected where the resolution strategies are decided by *ArgoFiCo* depending on the *configuration* (*max* or *min*). The outputs are the firewalls' low-level configurations. These configurations are generated automatically and are composed of all the ordered rules that should apply in each firewall.

High-level requirements The high-level network requirements describe the behaviour rules of the network. These requirements specify which communications are allowed and which are denied. Given the high complexity of the network, we permit the user to specify these requirements using statements that are close to natural language and are user-friendly. In particular, the user can provide requirements like: "*All internal hosts of the subnet can reach Server Mail*", "*The ciphered outgoing communications are denied*". The use of high-level requirements permits the user to provide the network requirements by avoiding human errors.

Network topology and knowledge base Another important input for our tool is the network topology, where the users provide information about the network. In particular, they provide information about the topology of the network, with its components (hosts and firewalls), their positions, containing relations, and the routing tables. The *knowledge base* contains the map from high-level terms to low-level network layer information, where the information is given in the form of IP-tables and the names used to describe the network components are translated into IPs. The network topology and knowledge base are needed by *ArgoFiCo* to correctly derive from the high-level requirements the low-level configuration for every firewall.

Low-level Configurations The low-level configurations are the outputs of our tool. These configurations are generated automatically and are composed of all the ordered rules that need to apply in each firewall.

3.3 Resolution Strategies

The resolution strategies describe how the anomalies should be solved in the policy specifications. Some of the implemented strategies are presented below[4]:

- *Allow Takes Precedence* (ATP) in case of contradicting actions that are simultaneously activated, we enforce the Allow rule over the Deny one;
- *Deny Takes Precedence* (DTP) in case of contradicting actions that are simultaneously activated, we enforce the Deny rule over the Allow one (this is a restrictive strategy);
- *Most Specific Takes Precedence* (MSTP) in case two conflicting rules are applied, we give priority to the most specific rule;
- *Least Specific Takes Precedence* (LSTP) in case two conflicting rules are applied, we give priority to the less specific rule.

ArgoFiCo applies different strategies for different types of anomalies.

[4] The presented strategies are some of the standard strategies. Further resolution strategies can be integrated into *ArgoFiCo*.

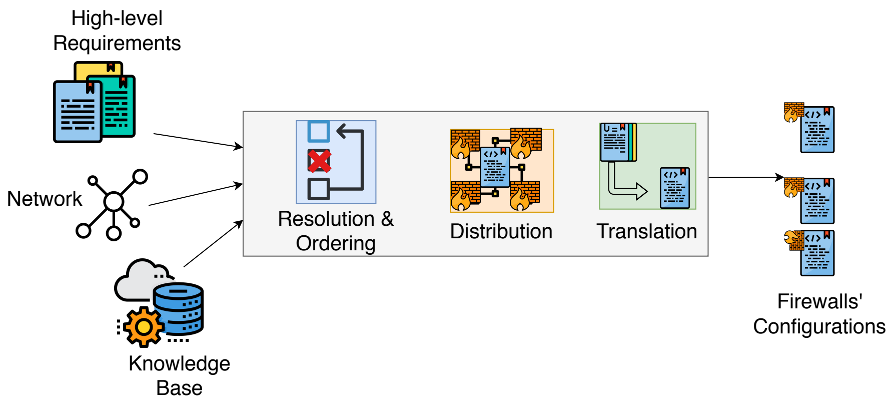

Fig. 2. The main components of *ArgoFiCo*

4 *ArgoFiCo*'s Components

Let us now describe in detail the main components of *ArgoFiCo*. We present in Figure 2 a general overview of the various components of our tool and how they interact with each other. The inputs are provided by the user and are shown on the left side. Our tool, shown in the center, is composed of three main components: the resolution and ordering module, the distribution module, and the translation module. The inputs are passed to the resolution and ordering module, which automatically checks the rules for anomalies, and orders them. The ordered high-level firewall rules are then passed to the distribution module, which distributes the rules to the various firewalls of the network. Finally, the rules for each firewall are translated into low-level firewall rules using the translation module, which generates for each firewall its configuration file, shown on the right side of Figure 2. Let us now describe in detail these three modules.

4.1 Resolution and Ordering Module

The high-level requirements, provided by the user, are passed to the resolution and ordering module. The high-level requirements have the following form:

$$req(allow/deny,\ source,\ destination,\ type\ of\ traffic).$$

These requirements are analyzed in order to identify possible anomalies. This module is able to resolve all the anomalies between rules of the same firewall. In particular, *ArgoFiCo* avoids all the anomalies introduced in [1] e.g., the *shadowing* anomaly, i.e., a rule is shadowed when a previous rule matches all its packets, and the shadowed rule is never activated, the *correlation* anomaly, i.e., two rules with conflicting actions match some packets of each other, the *generalization* anomaly, i.e., two rules with conflicting actions, where all the packets matched from one of the rules are a subset of the packets of the second rule, the *redundancy* anomaly, i.e., two rules with the same actions have matching or partially matching packets, the *irrelevance* anomaly, i.e., a rule that does not match any traffic that might flow in that network. Algorithm 1 shows the pseudo-code of the process that identifies the potential anomalies.

The potential anomalies are avoided using the resolution strategies, which in some cases remove rules and in other cases order them by introducing priorities. The use of argumentation reasoning permits us to work with conflicting rules by introducing priorities. These priorities are used to order the rules, where the ordering of the rules is performed by clustering the rules that have sources and destinations that are in relation[5] between each other. The rules between different clusters are not related, therefore, the order between them is irrelevant. Algorithm 1 shows the resolution strategies applied to avoid the anomalies, i.e., removal of rules or introduction of priorities.

```
// $p_1$ and $p_2$ are policy rules part of the high-level requirements given as input
if $!topology(p_1.src) \vee !topology(p_1.dst)$ then
    $remove\ p_1$
end
if $!topology(p_2.src) \vee !topology(p_2.dst)$ then
    $remove\ p_2$
end
if $exactmatch(p_1, p_2)$ then
    if $p_1.action == p_2.action$ then $remove\ p_2$;
    else if $p_1.action =$ "$allow$" then $remove\ p_1$;
    else $remove\ p_2$;
end
else if $inclusion(p_1, p_2)$ then
    if
      $p_1.action == p_2.action \wedge \nexists\ p_3\ s.t.,\ inclusion(p_3, p_2) \wedge inclusion(p_1, p_3) \wedge p_2.order >$
      $p_3.order \wedge p_3.order > p_1.order$ then $remove\ p_2$;
    else
        $p_2.order > p_1.order$
    end
end
else if $intersection(p_1, p_2) \wedge p_1.action! = p_2.action$ then
    if $(p_1.action =$ "$allow$"$)$ then $p_2.order > p_1.order$;
    else $p_1.order > p_2.order$;
end
```

Algorithm 1: Anomalies identification and resolution algorithm

Example 1. We now show with the use of an example how the resolution and ordering module works. Our example is a cyclic network topology, shown in Figure 3 that is composed of three different subnetworks. Each subnetwork contains different hosts and is connected to the others through the use of three firewalls. The high-level requirements provided by the administrator are presented below:

1. $req_1(deny, subnet1, subnet2, all)$
2. $req_2(allow, bob, david, tcp)$

[5] We use the same definitions of *relations* between firewall rules as in [1].

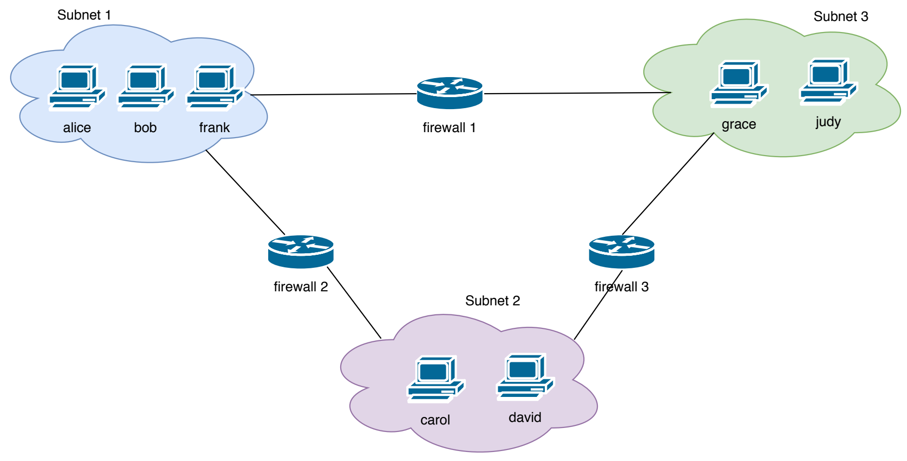

Fig. 3. Network topology for Example 1 and 2

3. $req_3(allow, bob, subnet2, tcp)$
4. $req_4(deny, subnet1, subnet3, all)$
5. $req_5(deny, eve, subnet4, all)$
6. $req_6(allow, subnet1, subnet3, all)$

The source and destination names are already provided in the topology and the protocols are *tcp*, *udp*, and *all*.

The above requirements are analyzed by the ordering and resolution module, which provides the following potential anomalies together with their resolution:

– Conflict 1 - redundancy: rule req_2 is included[6] in rule req_3 and their actions are the same. Since req_2 is more specific and it does not exist any rule req_x with a different action, included in req_3 and that includes req_2, then req_2 is removed [see line 13-14 of Algorithm 1];
– Conflict 2 - shadowing: rule req_3 is included in rule req_1 but they have different actions. Since rule req_3 is more specific, it has a higher priority with respect to rule req_1, $(req_3 > req_1)$ [see line 13-18 of Algorithm 1];
– Conflict 3 - irrelevance: rule req_5 has as source and destination hosts that are not in the topology. Thus, this rule is irrelevant and is removed [see line 2-7 of Algorithm 1];
– Conflict 4 - shadowing: req_6 is exactly matching req_4 but they have different actions. Thus, req_6 is removed [see line 8-12 of Algorithm 1].

The result of the resolution and ordering module is as below:

1. $req_3(allow, bob, subnet2, tcp)$
2. $req_1(deny, subnet1, subnet2, all)$

[6] In Algorithm 1, $inclusion(p_1, p_2)$ means that all the packets matched by rule p_2 are included in the packets matched by rule p_1.

3. $req_4(deny, subnet1, subnet3, all)$

where req_2, req_5, and req_6 are removed. □

4.2 Distribution Module

The distribution module is the second process of our tool. It gets as input the ordered rules, from where the intra-firewall anomalies are removed. The distribution module, given the input, analyzes the rules and gives as result the high-level configuration for each firewall. We call it a high-level configuration as the provided rules are in high-level form and not low-level network format. This module is able to identify and avoid all the potential anomalies between rules of different firewalls (inter-firewall anomalies). In particular, our tool avoids all the inter-firewall anomalies introduced in [1] e.g., the *shadowing* anomaly, i.e., an upstream firewall blocks the traffic accepted by a downstream firewall, the *spuriousness* anomaly, i.e., an upstream firewall permits the traffic denied by a downstream firewall, the *redundancy* anomaly, i.e., a downstream firewall blocks the traffic already blocked by an upstream firewall, the *correlation* anomaly, i.e., two rules with conflicting actions, where all the packets matched by one of the rules are a subset of the packets of the second one, where one rule is in an upstream firewall and the other is in a downstream firewall. Algorithm 2 shows the pseudo-code of the process that distributes the rules in the various firewalls by avoiding all the above anomalies.

The distribution module provides two types of high-level firewalls' configurations to the user that can decide which one to apply. Algorithm 2 shows the pseudo-code for the distribution module, where both ordering configurations are included. The first configuration is the *max configuration* and provides the firewalls' configurations that allow or deny the sources to send messages to the destinations, by using *all possible paths.* This configuration is the ideal one when we want to configure a robust network. In case a particular part of the network is unavailable, then the packets can use alternative routes. The firewalls of these alternative routes are prepared to route/filter the traffic appropriately. In the max configuration, we expect redundancies between firewalls as this configuration generates repetitions of rules. The second configuration is the *min configuration* and provides the firewalls' configurations that allow or deny the sources to send messages to the destinations, by using *a specific path.* This configuration populates the firewalls only with the needed rules and avoids all types of inter-firewalls anomalies. In this case, we expect the traffic flow to be faster and efficient but the traffic cannot be routed/filtered by alternative firewalls in case of unavailable paths.

Example 2. Let us now continue with the example introduced in Example 1 and show the application of the distribution module that creates the firewalls' high-level configurations. The results are passed to the distribution module, and the max configuration is selected. The first step is to understand which are the paths for a certain package to go from the source to the destination. For req_3 we have

```
// $p$ is a policy rule
if $configuration ==$ "max" then
    foreach $path$ in $paths(p.source, p.dest)$ do
        if $p.action ==$ "allow" then
            foreach $f$ in $firewall(path)$// $firewall(path)$ is the set of all the
               firewalls in $path$
             do
                $f.rule.add(p)$
            end
        end
        else $most\_upstream(firewall(path)).rule.add(p)$ ;
    end
end
if $configuration ==$ "min" then
    $path = routing(p.source, p.dest)$
    if $p.action ==$ "allow" then
        foreach $f$ in $firewall(path)$ do
            $f.rule.add(p)$
        end
    end
    else $most\_upstream(firewall(path)).rule.add(p)$ ;
end
```

Algorithm 2: Firewalls' population algorithm

two paths: the first uses Fw_1[7] and Fw_3 and the second uses Fw_2; for req_1 we have two paths: the first uses Fw_1 and Fw_3 and the second uses Fw_2; and for req_4 we have two paths: the first uses Fw_1 and the second uses Fw_2 and Fw_3. req_3 should be applied in all paths, as it is an "*allow*" action, thus, it should be placed in all firewalls. For req_1 and req_4 because their action is "*deny*", the rules are put in the most upstream firewall along each path. The output of the distribution module, with max configuration, is as follows:

Fw_1

1. $req_3(allow, bob, subnet2, tcp)$
2. $req_1(deny, subnet1, subnet2, all)$
3. $req_4(deny, subnet1, subnet3, all)$

Fw_2

1. $req_3(allow, bob, subnet2, tcp)$
2. $req_1(deny, subnet1, subnet2, all)$
3. $req_4(deny, subnet1, subnet3, all)$

Fw_3

1. $req_3(allow, bob, subnet2, tcp)$

In case the min configuration is selected, the result is:

1. $req_3(allow, bob, subnet2, tcp)$

where req_1 and req_4 are removed, as there is no rule with a lower priority than them that "allows" a related requirement, and we expect to have the "deny all" rule at the end of the firewalls. The distribution module puts req_3 in Fw_2. □

[7] For the sake of simplicity we denote the firewalls with Fw.

4.3 Translation Module

The results of the distribution module are passed to the translation module. The inputs of the translation module are the high-level configurations for each firewall. This module translates the high-level configurations into low-level network layer information. The terms used in the high-level requirements refer to the names of sources, destinations, and specific services, while the low-level network layer terms are the traditional IP addresses, port numbers, and protocols used by the firewall rules. The translation is performed using information from the network topology and the knowledge base. For example, "Employees" is mapped to a set of IP addresses, "College Subnet" is mapped to a subnetwork, and "illegal sites" is a set of IP addresses and ports. The results of this module are the configuration files for each firewall, that can be loaded to the various firewalls.

5 Implementation and Validation

The main component of *ArgoFiCo* is the resolution and ordering module, which finds the various anomalies and orders the rules. The ordering of the rules is made through the use of argumentation reasoning. Our tool permits the user to provide the various high-level requirements and the network topology. The user gets as result from our tool the firewalls' configurations. *ArgoFiCo* implements the argumentation reasoning through the use of the Gorgias tool. In order for the administrator to simulate the *ArgoFiCo* results, we provide a direct call to an open-source network emulator, called *Core*[8]. The use of the network emulator provides the administrator with a graphical representation of the flow of information and how the network requirements are applied.

The use of argumentation reasoning permits *ArgoFiCo* to provide *explanations* together with the given results. The explanations are provided in the form of text, explaining why certain requirements were removed, changed, or ordered in a particular way, by giving the solved anomalies and the used resolution strategies. The explanations make the administrator part of the process and provide insights for possible errors or correctness of the given high-level requirements.

We tested our framework using various realistic scenarios. *ArgoFiCo* always configured correctly the rules of the firewalls, for both max and min configurations. The given firewalls' configurations were tested using the network emulator that confirmed the correctness of the rules.

Example 3. Let us now show with the use of a more complex example how our tool is able to automatically generate the distributed firewalls' configurations. We provide the network topology of our case study in Figure 4, where there are six subnetworks and six different firewalls. In particular, the scenario described previously in Example 1 and 2 is a small part of this example. We present below all the provided high-level requirements:

1. $req_1(allow, grace, peggy, all)$

[8] `https://github.com/coreemu/core`

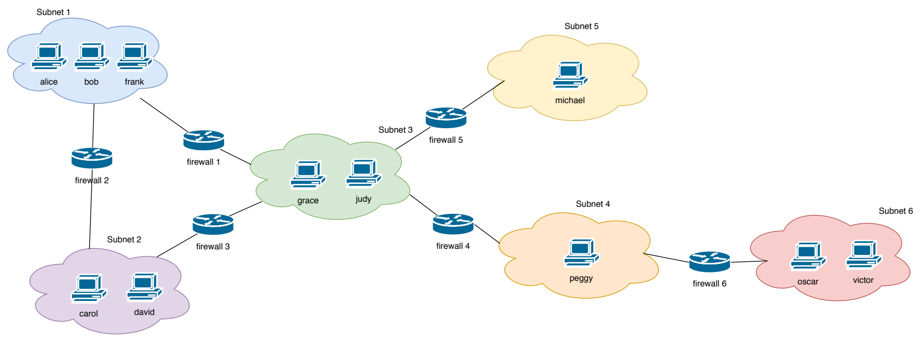

Fig. 4. Network topology for Example 3

2. $req_2(deny, grace, peggy, all)$
3. $req_3(allow, olivia, david, all)$
4. $req_4(allow, subnet4, subnet1, all)$
5. $req_5(deny, subnet4, alice, udp)$
6. $req_6(allow, subnet4, bob, tcp)$
7. $req_7(allow, subnet6, judy, udp)$
8. $req_8(deny, oscar, subnet3, all)$
9. $req_9(allow, alice, subnet5, all)$
10. $req_{10}(allow, alice, michael, tcp)$
11. $req_{11}(deny, peggy, david, all)$
12. $req_{12}(allow, victor, bob, udp)$
13. $req_{13}(allow, victor, bob, udp)$

We present below the identified anomalies and their resolution strategies:

– Conflict 1 - shadowing: req_1 is exactly matching with req_2 and their actions are different. Due to the strategy resolution employed, req_1 is removed;
– Conflict 2 - irrelevance: req_3 contains a host that is not in the current topology, thus this requirement is irrelevant and is removed;
– Conflict 3 - shadowing: req_5 is included in req_4 and their actions are different. Since req_5 is more specific, then it has higher priority than req_4, ($req_5 > req_4$);
– Conflict 4 - redundancy: req_6 is included in req_4 and their actions are the same. Since req_6 is more specific and allows the traffic, then it is removed;
– Conflict 5 - correlation: there is an intersection between the packets of req_7 and req_8 and their actions are different. Due to the resolution strategy employed, req_8 has higher priority than req_7, ($req_8 > req_7$);
– Conflict 6 - redundancy: req_{10} is included in req_9 and their actions are the same. Since req_{10} is more specific and allows the traffic, then it is removed;
– Conflict 7 - redundancy: req_{12} is exactly matching with req_{13} and their actions are the same. Thus, req_{12} is removed.

We present below the results of the resolution and ordering module that provide the ordered high-level requirements.

1. $req_2(deny, grace, peggy, all)$
2. $req_5(deny, subnet4, alice, udp)$
3. $req_4(allow, subnet4, subnet1, all)$
4. $req_8(deny, oscar, subnet3, all)$
5. $req_7(allow, subnet6, judy, udp)$
6. $req_9(allow, alice, subnet5, all)$
7. $req_{11}(deny, peggy, david, all)$
8. $req_{13}(allow, victor, bob, udp)$

where req_1, req_3, req_6, req_{10}, req_{12} are removed.

This ordered list of requirements is passed to the distribution module, which for the max configuration gives the following high-level firewalls' configurations.

Fw_1
1. $req_4(allow, subnet4, subnet1, all)$
2. $req_9(allow, alice, subnet5, all)$
3. $req_{13}(allow, victor, bob, udp)$

Fw_2
1. $req_4(allow, subnet4, subnet1, all)$
2. $req_9(allow, alice, subnet5, all)$
3. $req_{13}(allow, victor, bob, udp)$

Fw_3
1. $req_4(allow, subnet4, subnet1, all)$
2. $req_9(allow, alice, subnet5, all)$
3. $req_{13}(allow, victor, bob, udp)$

Fw_4
1. $req_2(deny, grace, peggy, all)$
2. $req_5(deny, subnet4, alice, udp)$
3. $req_4(allow, subnet4, subnet1, all)$
4. $req_7(allow, subnet6, judy, udp)$
5. $req_{11}(deny, peggy, david, all)$
6. $req_{13}(allow, victor, bob, udp)$

Fw_5
1. $req_9(allow, alice, subnet5, all)$

Fw_6
1. $req_8(deny, oscar, subnet3, all)$
2. $req_7(allow, subnet6, judy, udp)$
3. $req_{13}(allow, victor, bob, udp)$

When the min configuration is called, the results are as below:

Fw_2
1. $req_4(allow, subnet4, subnet1, all)$
2. $req_9(allow, alice, subnet5, all)$
3. $req_{13}(allow, victor, bob, udp)$

Fw_4
1. $req_5(deny, subnet4, alice, udp)$
2. $req_4(allow, subnet4, subnet1, all)$

3. $req_7(allow, subnet6, judy, udp)$
4. $req_{13}(allow, victor, bob, udp)$

Fw_5

1. $req_9(allow, alice, subnet5, all)$

Fw_6

1. $req_8(deny, oscar, subnet3, all)$
2. $req_7(allow, subnet6, judy, udp)$
3. $req_{13}(allow, victor, bob, udp)$

The min configuration removes the requirements: req_2, req_{11}, as they cannot be triggered, given that there is no relevant "allow" requirement after them, and the default "*deny all*" rule will be added in all the firewalls. Fw_1 and Fw_3 are not populated and have the default configuration that is the "*deny all*" rule. $\square$

6 Conclusion and Future Work

In this paper, we introduced a tool that automatically configures distributed firewalls in a network. This tool will alleviate the work of network administrators, as it permits to configure anomaly free firewalls given the network topology and its high-level requirements. Our approach provides the administrators the needed level of flexibility when configuring the firewalls, as they can decide how the ordering of the rules is done, by specifying the distribution option. The administrators can choose if they want to focus on the reachability aspects of the network and make it more robust by using the max configuration, or if they want to have a more slim and fast routing/filtering by using the min configuration. The main goal of the tool is to help the administrators during the firewalls' configuration by avoiding human errors.

The introduced automatic tool is based on a formal framework, called *ArgoFiCo* that uses abduction and argumentation reasoning. We presented the three modules that compose *ArgoFiCo*: the resolution and ordering module, the distribution module, and the translation module. The use of argumentation permits *ArgoFiCo* to provide an explanation together with the given result in order to explain to the administrators the configurations and to provide further help to them. The explanations provide the administrators with new insights and can be used to capture any human error in the high-level requirements. We also provide a direct execution to an open-source network emulator that shows graphically how the network will behave.

Currently, the inputs for *ArgoFiCo* are provided by the network administrators. We plan to fully automate this process and to integrate it with a reinforcement learning algorithm, in order to learn from the administrators' experience the best resolution strategy to use. The current version of our tool is able to generate firewalls' configurations by scratch (for empty or not yet configured firewalls) or by replacing the old configurations with the new ones. In the future, we plan to extend the tool to update existing firewalls' configurations, without the need for re-configuring all the firewalls. It would be interesting to apply this tool in high complexity commercial networks and to perform usability testing.

Acknowledgments

Erisa Karafili was partially supported by the European Union's H2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No. 746667.

References

1. Al-Shaer, E., Hamed, H., Boutaba, R., Hasan, M.: Conflict classification and analysis of distributed firewall policies. IEEE J. on Selected Areas in Communications **23**(10), 2069–2084 (2005)
2. Arunkumar, S., Pipes, S., Makaya, C., Bertino, E., Karafili, E., Lupu, E., Williams, C.: Next generation firewalls for dynamic coalitions. In: IEEE SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI. pp. 1–6 (2017)
3. Bandara, A.K., Kakas, A., Lupu, E.C., Russo, A.: Using argumentation logic for firewall policy specification and analysis. In: Large Scale Management of Distributed Systems. pp. 185–196 (2006)
4. Bartal, Y., Mayer, A., Nissim, K., Wool, A.: Firmato: A novel firewall management toolkit. ACM Transactions on Computer Systems **22**(4), 381–420 (2004)
5. Basile, C., Valenza, F., Lioy, A., Lopez, D.R., Pastor Perales, A.: Adding support for automatic enforcement of security policies in nfv networks. EEE/ACM Trans. Netw. **27**(2), 707–720 (2019)
6. Basile, C., Canavese, D., Lioy, A., Valenza, F.: Inter-technology conflict analysis for communication protection policies. In: Lopez, J., Ray, I., Crispo, B. (eds.) Risks and Security of Internet and Systems. pp. 148–163. Springer International Publishing, Cham (2015)
7. Bringhenti, D., Marchetto, G., Sisto, R., Valenza, F., Yusupov, J.: Automated optimal firewall orchestration and configuration in virtualized networks. In: NOMS 2020 - IEEE/IFIP Network Operations and Management Symposium, Budapest, Hungary, April 20-24, 2020. pp. 1–7. IEEE (2020)
8. Cullen, A., Karafili, E., Pilgrim, A., Williams, C., Lupu, E.: Policy support for autonomous swarms of drones. In: ETAA@ESORICS 2018. pp. 56–70 (2018)
9. Garcia-Alfaro, J., Cuppens, F., Cuppens-Boulahia, N., Preda, S.: Mirage: A management tool for the analysis and deployment of network security policies. Data Privacy Management and Autonomous Spontaneous Security **6514**, 203–215 (2011)
10. Kakas, A., Moraitis, P.: Argumentation based decision making for autonomous agents. In: AAMAS '03. pp. 883–890 (2003)
11. Kakas, A.C., Kowalski, R.A., Toni, F.: Abductive logic programming. J. Log. Comput. **2**(6), 719–770 (1992)
12. Kakas, A.C., Mancarella, P., Dung, P.M.: The acceptability semantics for logic programs. In: ICLP. pp. 504–519 (1994)
13. Karafili, E., Kakas, A., Spanoudakis, N., Lupu, E.: Argumentation-based Security for Social Good. In: AAAI Fall Symposium Series (2017)
14. Karafili, E., Lupu, E.: Enabling data sharing in contextual environments: Policy representation and analysis. In: SACMAT 2017. pp. 231–238 (2017)
15. Karafili, E., Lupu, E., Arunkumar, S., Bertino, E.: Argumentation-based policy analysis for drone systems. In: IEEE SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI. pp. 1–6 (2017)

16. Karafili, E., Spanaki, K., Lupu, E.: An argumentation reasoning approach for data processing. Computers in Industry **94**, 52–61 (2018)
17. Karafili, E., Valenza, F., Chen, Y., Lupu, E.: Towards a framework for automatic firewalls configuration via argumentation reasoning. In: NOMS 2020 - IEEE/IFIP Network Operations and Management Symposium, Budapest, Hungary, April 20-24, 2020. pp. 1–4. IEEE (2020)
18. Karafili, E., Wang, L., Kakas, A., Lupu, E.: Helping forensic analysts to attribute cyber-attacks: An argumentation-based reasoner. In: PRIMA 2018. pp. 510–518 (2018)
19. Karafili, E., Wang, L., Lupu, E.: An argumentation-based reasoner to assist digital investigation and attribution of cyber-attacks. Forensic Science International: Digital Investigation **32**, 300925 (2020)
20. Liu, A.X., Gouda, M.G.: Complete Redundancy Detection in Firewalls. In: DBSeC. pp. 193–206 (2005)
21. Moffett, J., Sloman, M.: Policy hierarchies for distributed systems management. IEEE Journal on Selected Areas in Communications **11**(9), 1404–1414 (1993)
22. Valenza, F., Basile, C., Canavese, D., Lioy, A.: Classification and analysis of communication protection policy anomalies. IEEE/ACM Trans. Netw. **25**(5), 2601–2614 (2017)
23. Valenza, F., Spinoso, S., Basile, C., Sisto, R., Lioy, A.: A formal model of network policy analysis. In: RTSI. pp. 516–522 (2015)
24. Verizon: Data Breach Investigations Report (2019)
25. Verma, P., Prakash, A.: FACE: A Firewall Analysis and Configuration Engine. In: SAINT05. pp. 74–81 (2005)