Using Argumentation Logic for Firewall Configuration Management

Arosha K. Bandara
Centre for Research in Computing
Dept. of Computing, The Open University,
Milton Keynes MK7 6AA, UK,
Email: a.k.bandara@open.ac.uk

Antonis C. Kakas
Department of Computing,
University of Cyprus,
Nicosia, Cyprus
Email: antonis@ucy.ac.cy

Emil C. Lupu, Alessandra Russo
Department of Computing,
Imperial College London,
London SW7 2AZ, UK,
Email: e.c.lupu, a.russo@imperial.ac.uk

***Abstract*—Firewalls remain the main perimeter security protection for corporate networks. However, network size and complexity make firewall configuration and maintenance notoriously difficult. Tools are needed to *analyse* firewall configurations for errors, to verify that they correctly implement security requirements and to *generate* configurations from higher-level requirements. In this paper we extend our previous work on the use of formal argumentation and preference reasoning for firewall policy analysis and develop means to automatically generate firewall policies from higher-level requirements. This permits both *analysis* and *generation* to be done within the same framework, thus accommodating a wide variety of scenarios for authoring and maintaining firewall configurations. We validate our approach by applying it to both examples from the literature and real firewall configurations of moderate size ($\approx$ 150 rules).**

I. Introduction

Despite numerous developments in intrusion detection, application level traffic filtering and other network security techniques, *firewalls* remain the main perimeter protection technique for many corporate or academic organisations. In their most common occurrence firewalls are configured through an ordered set of rules specifying which types of traffic should be permitted or denied based on IP header information. As a wide variety of protocols need to be accommodated and as corporate networks have grown, the number of rules in a typical firewall configuration has been steadily growing.

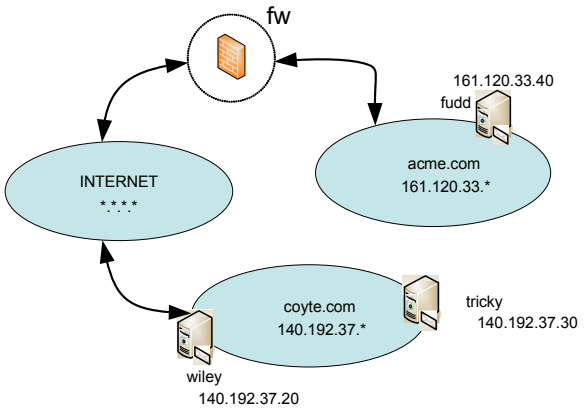
Based on data collected between 2000 and 2001, Wool reported [1] average configurations of 144 rules (including rules for Network Address Translation), and reaching into their thousands. At these sizes firewall configurations become difficult to specify and to maintain without adequate tool support. Beyond the size of the rule set, maintenance is difficult because changes are more frequently required and are made by different administrators. Consequently, it becomes increasingly difficult to predict the effect of specific changes, such as adding or removing a rule, to determine to what extent the rule set implements the organisation's security policies and preserves desired security properties, and to maintain a coherent strategy for organising such rule-sets. Errors in firewall configurations have therefore become increasingly frequent [1], [2], span a broader range [3] and are likely even when experienced administrators are responsible for their specification [3].

A firewall implements a significant subset of an organisation's security *requirements* as stated in its *security policy*. Consequently, firewall configurations comprise rules derived from this *high-level security policy*. Tools that can derive firewall configuration rules from higher-level, abstract specifications reduce the burden on the administrator, the size and complexity of the specification and the amount of interpretation necessary to implement the security policy. The process is therefore not only less cumbersome but also less error prone. Because firewall configuration rules implement the organisational security policy, they are often called *firewall policies*, or *policies* for short. We will therefore use the terms *high-level policy* when referring to more abstract, higher level specifications and *firewall policies or rules* when referring to the firewall configuration rules.

Figure 1 shows an example presented in [4] where a firewall is used to protect hosts in an enterprise network (acme.com) from malicious network traffic together with the set of rules that control the behaviour of the firewall. In this example, the default security requirement of blocking all traffic is implemented by rules 8 and 11. To implement *acme.com's* high-level policy, *'allow FTP connections from all hosts in the coyote.com network except for the host called tricky.coyote.com'*, the network administrator must identify the hosts and networks involved and correctly position new rules in the firewall policy set. In this case, the requirement is implemented by rules 5 and 6. However, if the administrator had inverted the order of these rules, the effect would have been to allow FTP connections from all hosts in the coyote.com network *including* the host tricky.coyote.com.

Although a number of tools and techniques for firewall analysis exist (briefly reviewed in Section II), their ability to generate firewall rules from higher-level specifications remains limited; a particular challenge being the ability to generate a correct rule ordering. The reverse engineering of firewall rules into higher-level specifications that ignore rule ordering seems to have been comparatively more successful [5].

Firewall policy *analysis* is a broad term which usually covers checking for *deployment-independent anomalies* that occur regardless of the network topology or the high-level security requirements, checking for *deployment-specific anomalies* that are specific to the network deployment, the desired security



(a) Network topology

Order	Ptrcl	Src IP	Src Port	Dst IP	Dst Port	Action
1	tcp	140.192.37.20	any	*.*.*.*	80	block
2	tcp	140.192.37.*	any	*.*.*.*	80	allow
3	tcp	*.*.*.*	any	161.120.33.40	80	allow
4	tcp	140.192.37.*	any	161.120.33.40	80	block
5	tcp	140.192.37.30	any	*.*.*.*	21	block
6	tcp	140.192.37.*	any	*.*.*.*	21	allow
7	tcp	140.192.37.*	any	161.120.33.40	21	allow
8	tcp	*.*.*.*	any	*.*.*.*	any	block
9	udp	140.192.37.*	any	161.120.33.40	53	allow
10	udp	*.*.*.*	any	161.120.33.40	53	allow
11	udp	*.*.*.*	any	*.*.*.*	any	block

(b) Firewall rules

Fig. 1. Example firewall configuration [4]

properties or both. Firewall policy *generation* entails defining a notation for high-level security requirements, a formal procedure for deriving firewall rules as well as deriving the ordering of those rules in the configuration. Ordering is required to define relative rule precedence for overlapping rules, but may also be influenced by other factors such as performance and traffic profiles for non-overlapping rules.

Although different techniques for firewall analysis have been proposed, we favour a logic-based approach because the analysis process can be traced to logical derivations that can be explained to the user, and because several modes of reasoning including both deductive and abductive derivations can be conducted. This enables checking specifications against a wider set of properties including both the high-level security requirements of the organisation and the absence of anomalies identified in the literature. However, classical logic-based techniques are difficult to use for this purpose because reasoning must occur not only on the rules themselves but also on their relative priorities. Generating a firewall rule-set from policy requirements entails generating the ordering (i.e., priorities) between the rules and this represents a significant challenge. In this paper we show how this challenge can be overcome and how a logic based framework was used to generate *anomaly free* firewall configurations (including the rule ordering) for firewalls with approximately 150 rules.

This paper is structured as follows: Section II presents related work on firewall configuration and analysis; Section III describes the logic representation of the firewall rules and argumentation primitives used in subsequent sections; Section IV describes the generation of firewall rules; Section V describes how our firewall configuration framework might be used in practice whilst Section VI describes our evaluation results. We present our conclusions and directions for future work in Section VII.

II. Related Work

A reasonably large array of techniques have been proposed for the analysis of firewall configurations including techniques based on testing [6], dedicated algorithms [7], [4], [8], [9], binary decision diagrams [10], model-checking [11], constraint logic programming [12], [13], and logic-programming with priorities [14].

Guttman [7] proposes an algorithm that addresses the localisation problem, by deriving local *postures* that correctly enforce the global policies. Postures represent the policy decisions at inbound or outbound interface of routers. However, the rules considered are declarative and the algorithm does not address issues of ordering between the rules, which we seek to address in our work.

Hazelhurst [10] advocates the use of binary decision diagrams (BDD's) to represent a rule-set as a Boolean expression and proposes a collection of algorithms for the analysis and validation of firewall rule-sets. *Fang* [8] is a firewall analysis engine based on the combination of a graph algorithm and a rule-set simulator. Fang parses relevant configuration files from all packet filtering routers on the network after reading a topology configuration file (created by the user in a subset of Firmato's MDL language [15]). It then allows the user to query the firewall policy with questions like *can host A contact host B using service C?*

Wool [9] improves on the Fang system with the Lumeta Firewall Analyser. This tool simulates and analyses the security policy implemented in the firewall. The user no longer needs to create a topology configuration file since this is retrieved automatically using the firewall's routing table information. Lumeta provides built-in queries since it was found that during testing the users did not know which queries to issue in order to elicit pertinent information.

Bartal et al. developed the Firmato system [15], which separates the security policy design from specific firewall vendors' low-level rules. This is achieved using a high-level language to define and analyse the policy that is subsequently mapped to the firewall rule-set. Firmato separates the security policy design from the network topology and includes a rule illustrator which allows a graphical representation of the rule-set.

Eronen and Zitting [12] developed a tool for analysing firewall rules based on constraint logic programming (CLP)

techniques. They highlight two problems: (i) the need to express the high-level security policy in the firewall's low-level language and (ii) the need to analyse what a set of firewall rules actually does. The authors found that by using logic programming and a generic inference engine (Eclipse), it was easy to state rules and add to the system's knowledge base. Also based on the Eclipse CLP environment Uribe and Cheung [13] propose a firewall analysis system that caters for the analysis of network intrusion detection systems (NIDS) and networks combining both firewalls and NIDS's.

Al-Shaer and Hamed [4], [3] developed a set of techniques and algorithms called Firewall Policy Advisor Tools. These use a firewall policy model (the *policy tree*) and a classification of various anomaly types. The authors introduce their firewall policy editor and state that making changes to an existing firewall policy can be more difficult than creating a new one. This is due to the rules being ordered and therefore any additions or modifications need to be handled properly to avoid introducing anomalies. The policy editor aids the process of determining the proper position at which the new rule should be inserted.

Chomsiri and Pornavalai [16] develop an application to analyse and minimise the rule-sets of various commercial firewall products. Their method uses Relational Algebra and a Raining 2-D box model. By comparing more than two rules at a time, the technique allows all potential deployment independent anomalies to be discovered. The authors argue that their technique improves on that of Al-Shaer and Hamed [4] for removing redundancy anomalies.

Marmorstein and Kearns [17] describe a passive firewall analysis tool for iptables (Linux). The tool takes as inputs the firewall rule-set and the contents of a query file. Their analysis engine is constructed using a Multi-way Decision Diagram (MDD) library.

In our previous work [14], we described a technique based on Argumentation for Logic Programming with Priorities (LPP). This allows firewall administrators to use higher-level abstractions to specify their policy requirements. It also allows preferences to be specified which in turn allows the system to reason on the relative priorities of rules. Further, by using LPP, the results of queries are more useful to the user since they specify the rules that support any given conclusion. The argumentation based approach is able to perform the same analyses as supported by the other techniques described above albeit using a declarative representation of firewall policies that is more easily extended. Additionally, deployment specific properties of the network can also be easily checked. In this paper we show how we have extended our approach to automatically generate anomaly-free rule sets from a set of firewall policy requirements.

III. Logic-based Framework for Representing Firewall Configurations

The firewall administrator's task is to realise the organisational (high-level) security policy using the firewall's (low-level) rule-set. Rubin et al. [18] emphasise both the importance of the correct initial specification of the firewall rules and of their continued maintenance. This is especially true since over time, the firewall rule-set can become untidy and unwieldy, making it complicated and difficult to understand - even to its author. This can lead to the introduction of a variety of errors and conflicts. Al-Shaer et al. [3], present a classification of such conflicts and highlight the likelihood of even an experienced network administrator making errors in the implementation of firewall rule sets.

In this section we present a logic-based framework for representing firewall configurations that addresses some of these issues by reducing the burden on the administrator in the following ways: (1) logical abstractions of network elements allow policies to be specified at a higher level of abstraction; (2) automated analysis capabilities allow administrators to determine the effects of changes in high-level policies on the the firewall's behaviour; (3) automated rule generation means that administrators no longer have to worry about mapping their high-level policies into low-level rule or the precise order in which the low-level rules must be placed into the firewall.

Our approach uses a logical framework of argumentation, within which we map and formalise firewall configurations such that we can perform the firewall policy analysis and rule generation described above. The framework captures the operational semantics of firewall policies declaratively through the logical semantics of argumentation based preference reasoning.

In general, an argumentation framework is a pair $< T, A >$ where T is a theory in some background (monotonic) logic, equipped with an entailment relation, $\models$, and A is a binary relation on the subsets of T. These subsets of T form the *arguments* of the framework and A is an *attacking* relation between arguments. For any two arguments Δ_1 and Δ_2 $(\Delta_1, \Delta_2 \in T)$ we say that Δ_1 attacks Δ_2 when $(\Delta_1, \Delta_2) \in A$.

The semantics of an argumentation framework is based upon the notion of an admissible argument. These are arguments which counter attack all other arguments that attack them and hence (informally) they are at least as preferred as any other conflicting argument.

Definition 1 (Admissibility of Arguments): Given an argumentation framework $\langle T, A \rangle$ an argument, Δ, is admissible iff (i) Δ does not attack itself, and (ii) for all arguments Δ', if Δ' attacks Δ then Δ also (counter-) attacks Δ'.

The admissible arguments capture the preferred conclusions of the theory (or policy) and thus give a natural way for capturing preference based reasoning.

We use a particular framework of argumentation as realised by the framework of Logic Programming with Priorities (LPP) and its concrete form of Logic Programming without Negation as Failure (LPwNF) [19], [20]. This is particularly suited to representing firewall configurations since the firewall rules and their ordering can be naturally translated into the LPP framework as shown in the next sections.

For the purposes of this paper, a *logic program with priorities* in the LPwNF framework denoted by, $\mathcal{T}$, consists of four parts:

TABLE I
PREDICATE AND FUNCTION SYMBOLS FOR LOGIC-BASED REPRESENTATION OF FIREWALL CONFIGURATIONS

Symbol	Description
$ipaddr(Name, [A, B, C, D])$	Predicate specifying the IP address A.B.C.D of a host or network $Name$. Finite domain variables are used to represent address ranges.
$traffic(Name, [Protocol, SrcPort, DstPort])$	Predicate specifying the protocol, source and destination ports for the traffic type $Name$. Finite domain variables are used to represent port ranges
$action(Action, Pkt)$	Predicate specifying the action (allow or block) to be performed for a packet defined using the $pkt/3$ function.
$ruleorder(Name1, Name2)$	Predicate specifying that the rule called $Name1$ should be given higher priority than the rule called $Name2$. This is an abducible predicate.
$subset(Pkt1, Pkt2)$	Auxiliary predicate that is true iff the packets represented by $Pkt1$ are a subset of those represented by $Pkt2$. $Pkt1$ and $Pkt2$ are defined using the $pkt/3$ function.
$fwrule(Name, Action, Pkt)$	Function specifying a firewall rule called $Name$ that defines the $Action$ (allow or deny) for a packet characterised using the $pkt/3$ function.
$pkt(TrafficName, SrcName, DestName)$	Function specifying the signature for traffic $TrafficName$ from $SrcName$ to $DestName$

(i) a basic part logic program $\mathcal{P}$, consisting of rules of the form:
$$Name : L \leftarrow L_0, \ldots, L_n, (n > 0)$$
where, for our application, $L, L_1, \ldots, L_n$ are positive literals. As usual in Logic Programming a rule containing variables is a compact representation of all the ground rules obtained from this under the Hebrand universe. Each ground rule has a unique (parametric) name, $Name$, which is a term, given at the front of the rule.

(ii) a higher part $\mathcal{H}$, specifying conditional, dynamic priorities amongst rules in $\mathcal{P}$ or $\mathcal{H}$. Rules in $\mathcal{H}$ have the form:
$Name : prefer(rule1, rule2) \leftarrow L1, \ldots, Ln, (n > 0)$
to be read: if (some instance of) the conditions $L1, \ldots, Ln$ hold, then (the corresponding instance of) the rule named by $rule1$ has higher priority than (the corresponding instance of) the rule named by $rule2$. The priority rule is also given a name, $Name$;

(iii) an auxiliary background part $\mathcal{B}$, which is a normal logic program defining (auxiliary) predicates occurring in the conditions of rules in $\mathcal{P}, \mathcal{H}$ but not in the conclusions of any rule in $\mathcal{P}$;

(iv) a notion of incompatibility which, for our purposes, can be assumed to be given as a set of rules defining the predicate $conflict/2$, e.g.,
$$conflict(rule1, rule2)$$
which states that (an instance of) the rule named by $rule1$ is incompatible with the corresponding instance of the rule named by $rule2$. The incompatibility relation is symmetric and always includes that L is incompatible with $\neg L$ and that $prefer(r, s)$ is incompatible with $prefer(s, r)$ for any two rule names r, s.

Further details of the formal definition of the argumentation framework can be found in the Appendix.

To represent firewall configurations as a LPwNF theory, we must specify three types of information: the high-level policies, the network elements and traffic types relevant to the policies, and any additional properties to be satisfied when generating firewall rule ordering. Table I lists the symbols used to represent this information in our formalism.

A. High-level Policies

These form the basic part $\mathcal{P}$ of the LPwNF theory, and specify whether the action of the firewall for particular type of traffic from a given source to destination should be *allow* or *block*. We specify these policies using the $action/2$ predicate which takes as arguments, an action constant (either allow or block) and the packet definition (specified using the $pkt/3$ function). We can use this predicate to define rules for the high-level policy *"allow FTP connections from all hosts in the coyote.com network"* as follows:

$$fwrule(coyote\_ftp\_reqs, allow, pkt(ftp, coyote, any)) :$$
$$action(allow, pkt(ftp, coyote, any)).$$

Note that the policy can be specified using names for traffic type, source and destination rather than low-level protocol, port number and IP address information. Also, high-level policies need not specify any rule ordering. The $Name$ part of the rule is specified using the $fwrule/3$ function which can be used to identify the firewall rule elsewhere. We use the following rules to propagate the action specified in a given high-level policy to all sub-packets that match the policy:

$$fwrule(coyote\_ftp\_reqs, allow, SubPacket) :$$
$$action(allow, SubPacket)) \leftarrow$$
$$subset(SubPacket, pkt(ftp, coyote, any)).$$
$$fwrule(tricky\_ftp\_reqs, block, SubPacket) :$$
$$action(block, SubPacket)) \leftarrow$$
$$subset(SubPacket, pkt(ftp, tricky, any)).$$

Finally, we use the $conflict/2$ predicate to capture the operational behaviour of firewalls where a single rule is always applied to a packet even if multiple rules match. It is therefore inconsistent for two different rules to apply to the same packet:

$$conflict(fwrule(R_x, \_, Pkt), fwrule(R_y, \_, Pkt)) \leftarrow$$
$$R_x \neq R_y.$$

In Section IV we show how this conflict is resolved in the argumentation reasoning framework by making use of abduction to derive the ordering between conflicting rules. The ordering is computed to match firewall behaviour where the first matching rule is applied.

B. Network Elements and Traffic Types

This information forms the auxiliary part $\mathcal{B}$ of the LPwNF theory, mapping the high-level names used in the policies to low-level IP address, protocol and port numbers. We represent both host and network IP addresses, ports and protocols for types of traffic and IP packet headers using $ipaddr/2$, $traffic/2$ and $pkt/3$ respectively. Using these symbols, the representation of the *acme.com* network together with the host *tricky.coyote.com* shown in Figure 1 is:

$ipaddr(acme, [161, 120, 33, D]) \leftarrow D \ in \ 0..255.$
$ipaddr(tricky, [140, 192, 37, 20]).$

Additionally, we would represent the ports and protocols relevant to FTP traffic as follows:

$traffic(ftp, [tcp, SP, 21]) \leftarrow SP \ in \ 1..65536.$

We can also specify logical groupings of network elements and traffic types using the $addr\_group/2$ and $traffic\_group/2$ symbols. For example, assuming we have defined the traffic types ftp, scp and $tftp$, it is possible to group these together as the type $file\_transfer\_protocols$ as follows:

$traffic\_group(file\_transfer\_protocols, ftp).$
$traffic\_group(file\_transfer\_protocols, scp).$
$traffic\_group(file\_transfer\_protocols, tftp).$

The $subset/2$ relation takes these logical groupings into account when deciding if a particular packet is a subset of another.

Finally, we can combine these definitions together with the $pkt/3$ predicate to represent file transfer traffic from *tricky.coyote.com* to *acme* as $pkt(ftp, tricky, acme)$.

C. Rule Ordering Properties

These properties form the higher part $\mathcal{H}$ of the LPwNF theory and are used when generating the ordering of the firewall rules. They can be classified into two types, application-independent properties and application-specific properties.

Application-independent properties. An example of an application independent property of the final rule order would be the absence of shadowing anomalies. As shown in [21], a rule R_x is shadowed by a rule R_y if R_x matches a subset of packets matched by R_y, R_x has lower precedence than R_y and the actions of R_x and R_y are different. Therefore to avoid this anomaly, we must make sure that R_x has higher precedence. This can be specified in our formalism as follows:

$order(avoid\_shadow, R_x, R_y) :$
$prefer(order(basic, R_x, R_y), order(basic, R_y, R_x)) \leftarrow$
$\quad fwrule(R_x, Action1, Pkt1) \wedge$
$\quad fwrule(R_y, Action2, Pkt2) \wedge$
$\quad R_x \neq R_y \wedge Action1 \neq Action2$
$\quad subset(Pkt1, Pkt2).$

Application-specific properties. These are properties that are specific to the network configuration and organisation in which the firewall is deployed. For the example in Figure 1, the following rule specifies that rules that allow WWW traffic from host *wiley.coyote.com* have higher precedence than rules that block WWW traffic from the *coyote.com* network:

$order(www\_wiley, R_x, R_y) :$
$prefer(order(basic, R_x, R_y), order(basic, R_y, R_x)) \leftarrow$
$\quad fwrule(R_x, allow, pkt(www, wiley, \_)) \wedge$
$\quad fwrule(R_y, block, pkt(www, coyote, \_)).$

In the next section we will show how the logic framework presented above can be used to generate firewall configurations.

IV. Generating Firewall Configurations

As described previously, generating a firewall configuration entails mapping the high-level security policies to low-level rules as well as deriving the ordering of those rules in the configuration. The combination of the high-level policy and the network information specifications defined in the previous section and deductive reasoning can be used to fulfil the first part of the generation process. However, deriving the rule order requires that we complete the higher part $\mathcal{H}$ of the LPwNF theory, in order to resolve the conflict that specifies it is incompatible for two firewall rules to apply to the same packet (see Section III-A). To deal with such incompleteness we can use *abductive reasoning* to help us fill in the missing information such that the rule ordering properties we defined are satisfied. Abductive reasoning is a well understood technique in logic programming and AI [22] which can be used to derive a set of assertions, Δ_A, such that given a domain description, D, and some desired goal, G, $(D \cup \Delta_A) \models G$ holds.

Formally, the integration of abductive reasoning within an argumentation framework is done by first isolating the incompleteness in some of the predicates of the theory, which are called *abducible predicates*. We will denote the set of abducible predicates by $\mathcal{A}$. We then extend the argument rules of the theory, $\mathcal{T}$, by a set of generic rules of the form $Name : \Delta_A$ for any (ground) literal $Delta_A$ whose predicate is an abducible predicate ($\Delta_A \in \mathcal{A}$). In addition, we add in the theory generic priority rules that give any (existing) rule whose conclusion is incompatible with Δ_A higher priority over the new argument rule $name : \Delta_A$. We can then apply the same argumentation base semantics, as described above, allowing us now to include in our admissible arguments missing information in the form of abductive hypotheses supported by these new arguments for abducible literals.

We apply this to the problem of generating a rule order for the firewall policy, by introducing an abducible predicate, $ruleorder(R_x, R_y)$, which captures the abductive hypothesis that rule R_x has higher precedence (i.e. is stronger) than rule R_y:

$order(basic, R_x, R_y) :$
$prefer(fwr(R_x, Action1, Pkt), fwr(R_y, Action2, Pkt))$
$\quad \leftarrow R_x \neq R_y \wedge ruleorder(R_x, R_y).$

We also include an incompatibility definition to ensure that the orderings $ruleorder(R_x, R_y)$ and $ruleorder(R_y, R_y)$ cannot be part of the same admissible argument:

$conflict(order(basic, R_x, R_y), order(basic, R_y, Rx)).$

This incompatibility definition, together with the rule ordering properties defined previously will constrain the possible

abducible hypotheses that can be included in an admissible argument.

We can now generate the rule ordering by invoking the argumentation reasoning procedure with the goal of satisfying each of the firewall requirements specified in the basic part of the LPwNF theory. The admissible arguments generated for these queries will include the $order/3$ terms that specify the relative order between rules such that the defined properties are satisfied. We utilise a simple finite domain constraint expression to translate the relative ordering between rule pairs into a total order for the rule set.

In the next section we summarise how our framework would be used in practice.

V. Usage

The framework we have developed provides a comprehensive environment for firewall configuration authoring and maintenance. In essence, based on the network information, the high-level security policy requirements and the desirable properties for the firewall configuration, the framework can be used in three ways: a) to review a firewall configuration by querying the formal model for reachable nodes, types of traffic allowed/denied etc. b) to analyse a firewall configuration in order to detect anomalies and to detect if deployment specific properties have been violated and c) to generate a firewall configuration that can be translated to the format specific to the firewall used. Whilst a) is typically achieved by using the argumentation framework in a deductive reasoning mode both b) and c) are achieved through using argumentation together with abductive reasoning. The analysis performed in b) takes into account the rule ordering of the actual configuration as shown in detail in [14], whilst in c) a new rule order is being generated.

It is relatively simple to acquire network topology information and to automatically translate it to the formal representation. Security policy requirements can be specified in terms of traffic flow requirements across the firewall. Higher-level concepts such as sub-networks, address groups, service specific port ranges, rule-sets, etc. can be easily defined at the logical level or read from existing configurations. The firewall administrators would thus only need to specify and/or modify these requirements ignoring rule ordering aspects.

Desirable properties such as absence of anomalies or deployment specific properties (i.e., specific to the network topology) need to be formally specified. However, these specifications are not likely to change often and would typically be loaded from libraries. These properties are used as goals during the analysis phase where the abductive proof procedure is used to identify counter examples that violate these properties. The same properties are used as background knowledge during the generation phase. Thus, the generated firewall configuration satisfies the required properties. If all properties cannot be satisfied the abductive proof procedure will fail to find any admissible ordering.

Note that the framework remains general. Thus it is possible to specify not only precedence relationships between individual firewall rules but also precedence relationships between precedence relationships. This allows to define how precedence relationships should be defined based on external factors. For example, the framework can be easily extended to have a default behaviour such that precedence is given to rules that correspond to the most frequent traffic types thus optimising the firewall configuration to the expected traffic profile.

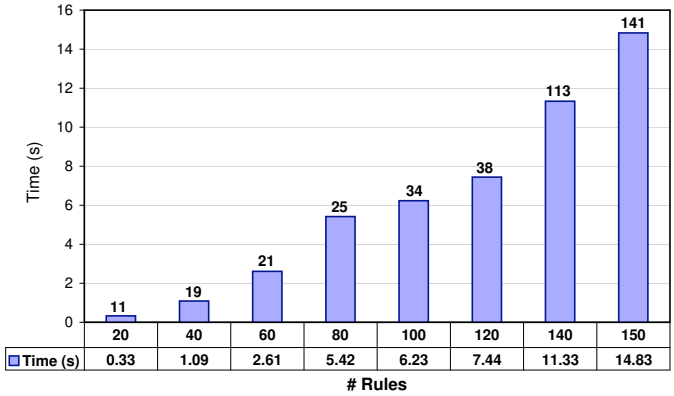


Fig. 2. Performance results for rule generation: Number of rules vs. time to generate rule ordering

VI. Evaluation

The approach to generating firewall configurations described above has a number of advantages when compared to the techniques presented in the literature.

- **Declarative:** The logic-based declarative specification of firewall requirements and configuration properties makes it easy for administrators to understand the policy set.
- **Modular:** The approach presented is modular as we can add or remove properties to be used in the rule order generation with minimal impact on the specification. Properties may concern deployment specific information, which depends on the network topology, or background rules about the relative frequency of traffic flows to optimise rule matching. Achieving similar extensions to techniques such as those proposed by Al-Shaer et al., would involve modifying the underlying algorithms for each new property.
- **Explanatory:** The derivation performed in order to reach the rule ordering can be documented and shown to the administrator in terms of the rules applied at each step.

We have implemented our approach by building on the GORGIAS implementation of the LPwNF proof procedure. Testing has been carried out using both simple examples, such as that presented in Figure 1 and a larger enterprise firewall specification containing approximately 150 policies and involving 1800 host/network definitions. In each case were able to generate rule orderings that eliminated both shadowing and redundancy anomalies. Other anomaly types, such as correlation and generalisation, are also detected by our analysis

procedure but not removed since they are not considered to be errors in the firewall configuration.

For the enterprise firewall specification we have started with the actual configuration which was translated to the logic based representation described in Section III. This provided the background knowledge together with the properties specifying the desirability of avoiding shadowing and redundancy anomalies. The rule generation procedure was then executed to derive the relative ordering of the rules. The correctness of the generated configuration was then manually checked.

Whilst we have yet to fully analyse the performance results from these tests, we present some preliminary findings in Figure 2. These results were obtained by taking subsets of the larger 150 rule firewall configuration and recording the time taken to generate the anomaly-free rule ordering. The experiment was run on a machine equipped with a AMD 64bit processor and 4GB RAM, running SWI-Prolog (v.5.6.14) on Ubuntu Linux (v.2.6.20.3). The label on each bar indicates the number of rule interactions in each set (i.e., number of rule pairs that had the potential to cause an anomaly). The results show that even for the largest policy set, containing the most interactions, we are able to generate the ordering in less than 15 seconds. Whilst this is slower than the rule insertion techniques for producing anomaly free orderings proposed in [3], it remains within acceptable limits for what is in essence an off-line activity.

The number of rules tested is close to the average firewall size according to [1] however many configurations are much larger. It would be desirable to evaluate our implementation against larger requirements sets; first, to evaluate its applicability and usefulness in other practical settings and second, to evaluate and allay concerns regarding its scalability for future use. However, larger firewall specifications occur mainly in corporate environments and thus far we have had difficulties procuring such specifications. Our experience is that randomly simulated rule sets do not provide realistic empirical results. In the general case, abductive reasoning in logic programming with priorities is NP-complete. However, firewall rules have a simple form and the domains for all rule variables are well defined. Our empirical measurements suggest that the approach could be applied to much larger cases and no efforts have so far been made to optimise the implementation. We aim to continue the evaluation of this approach both from an empirical and a theoretical point of view.

VII. Conclusions and Future Work

Argumentation logic permits non-monotonic reasoning with conflicting rules, and complex reasoning over the priorities of those rules. This permits to perform both analysis and generation of anomaly free firewall configuration rules thereby significantly reducing the burden and complexity of maintaining firewall configurations. Its advantages over other techniques lie in its modularity allowing customisation to the specifics of the network, the security policy and the properties desired of the resulting configuration, in its flexibility allowing analysis and review through both deductive and abductive reasoning over the specification and in its explanatory power as each logic derivation can be traced and presented to the user. Thus, we believe the above features make our approach an attractive alternative to existing techniques for firewall configuration management. However, these advantages appear to be gained at a significant performance cost (although we have not attempted any optimisations). The performance achieved so far appears to remain within acceptable limits although further evaluation work is needed on larger specifications, which are unfortunately difficult to obtain.

We have demonstrated in this paper that automated generation of firewall configurations is possible with a large degree of flexibility and whilst remaining agnostic to the specifics of the deployment architecture. Nevertheless, much work remains to be done. Although our framework has the ability to generate rule orderings optimised according to expected traffic profiles further work is required in order to implement this optimally. Although we have argued that it is desirable to generate firewall configurations from high-level specifications, the current logical framework still uses primitives close to the specification level of firewall rules. Although we do not anticipate particular challenges in defining higher-level abstractions in terms of the low-level predicates used in the logic based representation, further work is required. Our previous work on policy refinement [23] also provides valuable lessons on the realisation of higher-level specifications. Finally, modern firewalls provide stateful inspection, support for Virtual Private Networks, Network Address Translation, intrusion detection etc. and operate in a distributed setting. Further work is required to incorporate these features in the logical representation, analysis and generation, and to evaluate their impact on the complexity of the reasoning procedure.

Acknowledgment

We acknowledge financial support in part from the EC IST EMANICS Network of Excellence (#26854). This research is continuing through participation in the International Technology Alliance sponsored by the U.S. Army Research Laboratory and the U.K. Ministry of Defence.

References

[1] A. Wool, "A quantitative study of firewall configuration errors," *IEEE Computer*, vol. 37, no. 6, pp. 62–67, 2004.

[2] ——, "How not to configure your firewall: A field guide to common firewall configurations," in *Proc 15th Conf. on Large Systems Administration*. USENIX, 2001.

[3] E. Al-Shaer, H. Hamed, R. Boutaba, and M. Hasan, "Conflict classification and analysis of distributed firewall policies," *IEEE J. on Selected Areas in Communications*, vol. 23, no. 10, pp. 2069–2084, Oct. 2005.

[4] E. S. Al-Shaer and H. H. Hamed, "Firewall policy advisor for anomaly discovery and rule editing," in *Proc IFIP/IEEE Symp. on Integrated Network Management, Colorado Springs, USA*, 2003, pp. 17–30.

[5] A. Tongaonkar, N. Inamdar, and R. Sekar, "Inferring higher level policies from firewall rules," in *Proc 21st Conf. on Large Systems Administration, Dallas, Texas, USA, Nov*. USENIX, 2007, pp. 17–26.

[6] A. El-Atawy, T. Samak, Z. Wali, E. Al-Shaer, F. Lin, C. Pham, and S. Li, "An automated framework for validating firewall policy enforcement," in *Proc 8th IEEE Int. Workshop on Policies for Distributed Systems and Networks, Bologna, Italy, June*, 2007, pp. 151–160.

[7] J. D. Guttman, "Filtering postures: Local enforcement for global policies," in *Proc IEEE Symp. on Security and Privacy*, 1997, pp. 120–129.

[8] A. J. Mayer, A. Wool, and E. Ziskind, "Fang: A firewall analysis engine," in *Proc IEEE Symp. on Security and Privacy, Berkeley, USA*, 2000, pp. 177–187.
[9] A. Wool, "Architecting the lumeta firewall analyzer," in *Proc 10th USENIX Security Symp.*, Berkeley, USA, 2001, p. 7.
[10] S. Hazelhurst, A. Attar, and R. Sinnappan, "Algorithms for improving the dependability of firewall and filter rule lists," in *Proc. Int. Conf. on Dependable Systems and Networks*. Washington, DC, USA: IEEE Computer Society, 2000, pp. 576–585.
[11] L. Yuan, H. Chen, J. Mai, C.-N. Chuah, Z. Su, and P. Mohapatra, "FIREMAN: a toolkit for firewall modeling and analysis," in *Proc IEEE Symp. on Security and Privacy*, May 2006, p. 15.
[12] P. Eronen and J. Zitting, "An expert system for analyzing firewall rules," in *Proc 6th Nordic Workshop on Secure IT Systems (NordSec 2001), Copenhagen, Denmark*, Nov 2001, pp. 100–107.
[13] T. E. Uribe and S. Cheung, "Automatic analysis of firewall and network intrusion detection system configurations," in *Proc ACM Workshop on Formal Methods in Security Eng.*, Washington, DC, 2004, pp. 66–74.
[14] A. K. Bandara, A. C. Kakas, E. C. Lupu, and A. Russo, "Using argumentation logic for firewall policy specification and analysis," in *Proc 17th IFIP/IEEE Int. Workshop on Distributed Systems: Operations and Management*. Springer, 2006, pp. 185–196.
[15] Y. Bartal, A. J. Mayer, K. Nissim, and A. Wool, "Firmato: A novel firewall management toolkit," *ACM Trans. Comput. Syst.*, vol. 22, no. 4, pp. 381–420, 2004.
[16] T. Chomsiri and C. Pornavalai, in *Proc Int. Conf. on Security & Management (SAM 2006), Las Vegas, USA, June*.
[17] R. M. Marmorstein and P. Kearns, "A tool for automated iptables firewall analysis," in *Proc USENIX Annual Technical Conference, FREENIX Track*. USENIX, 2005, pp. 71–81.
[18] A. D. Rubin, D. Geer, and M. J. Ranum, *Web security sourcebook*. New York, NY, USA: John Wiley & Sons, Inc., 1997.
[19] A. C. Kakas, P. Mancarella, and P. M. Dung, "The acceptability semantics for logic programs," in *Proc 11th Int. Conf. on Logic Programming*, Italy, 1994, pp. 504–519.
[20] Y. Dimopoulos and A. C. Kakas, "Logic programming without negation as failure," in *Proc Int. Logic Programming Symp.*, Portland, USA, 1995, pp. 369–383.
[21] H. Hamed and E. Al-Shaer, "Dynamic rule-ordering optimization for high-speed firewall filtering," in *Proc ACM Symp. on Information, Computer and Communications Security*, F.-C. Lin, D.-T. Lee, B.-S. Lin, S. Shieh, and S. Jajodia, Eds. Taiwan: ACM, 2006, pp. 332–342.
[22] G. Paul, "Approaches to abductive reasoning: An overview," *Artificial Intelligence Review*, vol. 7, pp. 109–152, 1993.
[23] A. K. Bandara, E. C. Lupu, A. Russo, N. Dulay, M. Sloman, P. Flegkas, M. Charalambides, and G. Pavlou, "Policy refinement for diffserv quality of service management," in *Proc 9th IFIP/IEEE Int. Symp. on Integrated Network Management, Nice, France, May*. IEEE, 2005, pp. 469–482.

APPENDIX
LOGICAL FRAMEWORK FOR ARGUMENTATION

The work presented in this paper uses a particular framework of argumentation as realised by the framework of Logic Programming with Priorities (LPP) and its concrete form of Logic Programming without Negation as Failure (LPwNF) [19], [20].

This appendix provides further details on the formalisation of of argumentation and preference entailment. We start by re-introducing the basic definitions (detailed in Section III) where a *logic program with priorities* in the LPwNF framework denoted by, $\mathcal{T}$, is defined to consist of four parts:

(i) a basic part logic program $\mathcal{P}$, consisting of rules of the form: $Name : L \leftarrow L_0, \ldots, L_n, (n > 0)$, where for our application, $L, L_1, \ldots, L_n$ are positive literals. Each ground rule has a unique (parametric) name, $Name$.

(ii) a higher part $\mathcal{H}$, specifying conditional, dynamic priorities amongst rules in $\mathcal{P}$ or $\mathcal{H}$. Rules in $\mathcal{H}$ have the form: $Name : prefer(rule1, rule2) \leftarrow L1, \ldots, Ln, (n > 0)$ which can be read: if (some instance of) the conditions $L1, \ldots, Ln$ hold, then (the corresponding instance of) the rule named by $rule1$ has higher priority than (the corresponding instance of) the rule named by $rule2$. The rule itself is named $Name$;

(iii) an auxiliary background part $\mathcal{B}$, which is a normal logic program defining (auxiliary) predicates occurring in the conditions of rules in $\mathcal{P}, \mathcal{H}$ but not in the conclusions of any rule in $\mathcal{P}$;

(iv) a set of rules defining the predicate $conflict/2$, e.g., $conflict(rule1, rule2)$ which states that (an instance of) the rule named by $rule1$ is incompatible with the corresponding instance of the rule named by $rule2$.

The local priority information, given at the level of the rules of a theory, is lifted to give a preference over sets of rules that compose arguments and counter arguments. This preference, together with the incompatibility relation synthesize the attacking relation of the argumentation framework of LPwNF as follows.

Definition 2 (Attacking Relation of Arguments in LPwNF): Let T be an LPwNF theory and Δ, Δ' subsets of T. Then Δ' *attacks* Δ (or Δ' is a counter argument of Δ) iff there exists a literal, L, and subsets Δ_1 of Δ' and Δ_2 of Δ such that:

(i) $\Delta_1 \models_{LP} L$ and $\Delta_2 \models_{LP} LC$, minimally

(ii) $(\exists r \in \Delta_1, s \in \Delta_2 s.t. \Delta_2 \models_{LP} prefer(s, r))$
$\Leftarrow (\exists r \in \Delta_1, s \in \Delta_2 s.t. \Delta_1 \models_{LP} prefer(r, s))$

where $\models_{LP}$ is the entailment relation of the underlying logic programming framework, LC is any literal such that $incompatible(L, LC)$ holds and minimally, means that $\Delta \models_{LP} L$ and that L cannot be derived from any proper subset of Δ.

The second condition in this definition states that an argument Δ' for L attacks an argument Δ for a contrary conclusion LC only if the set of rules that it uses to prove L are at least of the same strength (under the priority relation priority/2) as the set of rules in Δ used to prove the contrary. Note that the attacking relation is typically not symmetric.

The admissible arguments of any LPwNF theory, $\mathcal{T}$, given the above general definition of admissibility, allow us to define a preference entailment, $\models_{PR}$, as follows:

Definition 3 (Preference Entailment in LPwNF): Given an LPwNF theory, $\mathcal{T}$, and a literal, L, the (skeptical) preference entailment, $\mathcal{T} \models_{\mathcal{PR}} \mathcal{L}$, holds iff:

(i) there exists a (maximal) admissible sub-theory $\mathcal{T}'$ of $\mathcal{T}$ such that $\mathcal{T}' \models_{\mathcal{LP}} \mathcal{L}$, and

(ii) for any $\overline{L}$ that is incompatible with L there does not exist an admissible sub-theory $\mathcal{T}''$ of $\mathcal{T}$ such that $\mathcal{T}'' \models_{\mathcal{LP}} \overline{\mathcal{L}}$.

When only the first condition of the above is satisfied we say that the theory $\mathcal{T}$ *credulously prefers or possibly prefers* L.