An Application of Defeasible Logic Programming for Firewall Verification and Reconfiguration

Pritom Rajkhowa[1], Shyamanta M. Hazarika[1], and Guillermo R. Simari[2]

[1] Department of Computer Science & Engineering
Tezpur University, Tezpur 784028, India
{pritomr,smh}@tezu.ernet.in

[2] Department of Computer Science & Engineering
Universidad Nacional del Sur, Bahia Blanca, Argentina
grs@cs.uns.edu.ar

Abstract. Firewalls are the frontier defense in network security. Firewalls provide a set of rules that identify how to handle individual data packets arriving at the network. Firewall configuration is increasingly becoming difficult. Filter properties called *anomalies* hint at possible conflicts between rules. An argumentation framework could provide ways of handling such conflicts. Verification of a firewall involve finding out whether anomalies exist or not. Reconfiguration involves removing critical anomalies discovered in the verification phase. In this paper, we show how a Defeasible Logic Programming approach with an underlying argumentation based semantics could be applied for verification and reconfiguration of a firewall.

Keywords: Defeasible Logic Programming, stateless firewall, stateful firewall, anomaly, argumentation.

1 Introduction

Firewalls are the frontier defense in network security. Firewalls filter out unwanted packets coming from or going to the secured network. Firewall rules are specified in order of priority and are of the form:

<order> : if <network-conditions> take <action>

However, managing firewall configuration is increasingly becoming complex. Errors in firewall configuration includes conflict among the existing rules, failing to specify all the required rules that enforce a certain level of security, inappropriate rule ordering, invalid syntax etc. The complexity and interdependency of policy rules makes firewall policy management a challenging task; continuous evolution of networks making it even more difficult. Filter properties called *anomalies* that hint at possible misconfiguration have therefore been introduced by Network Management researchers [1]. Verification of a firewall involves finding out whether anomalies exist or not. Anomalies make a firewall do not conform to the policy specification [2]. Reconfiguration of a firewall involves removing critical anomalies discovered in the verification phase.

Formal methods have been used in firewall anomaly detection. Considerable work on approaches based on logic has been undertaken. A formal logic in understanding meaning of firewall rules was proposed [3]. One of the earliest approach proposed in [4] represents firewall rule sets as Binary Decision Diagrams (BDDs), allowing the rule set to be analysed as boolean expressions. Similar to this, Multi Terminal Interval Decision Diagrams were used [12] to enable efficient packet classification. [7] presented a tool based on constraint logic programming (CLP) for analyzing firewall rules. More recently, firewalls anomalies are being seen as spatial properties. Thanasegaran et al [11] explored the spatial relationships amongst rules in a bit-vector based spatial calculus, BISCAL to detect and classify the conflicts. Villemaire and Hallé [8] and Hazarika [10] showed that the condition part of a rule in a rule-based firewall can be viewed as a spatial region while their sequential application lends a temporal aspect. With this notion they introduced spatio-temporal logics for firewalls; anomalies are properties within the logic. Model-checking such properties can account for anomalies in a firewall. [9] proposed a logic based on notions related to visibility for defining firewall anomalies. [5] describes technique based on argumentation for Logic Programming with Priorities. [6] use a system of meta-level argumentation for firewall configuration and resolving conflict.

In this paper, we show how defeasible argumentation could be applied in the firewall domain to yield interesting results through representation and reasoning about conflicts and reconfiguration. Our work differs from previous approaches using argumentation [5,6], as we express firewall properties within a *defeasible* logic in order to explore defeasibility in firewall policy. Defeasible rules are defined as rules that provide a weak link liable to defeat or overrule by some rule after all has been considered; e.g. check a rule for anomaly if only it is causing conflict with some rule in the rules set. It is unnecessary to go and check each rule for anomaly. Under a consideration that each rule is conflict free unless it is defeated by some conflict; we show how Defeasible Logic Programming (DeLP) could be exploited for validation and reconfiguration of a firewall.

2 Background: DeLP and Firewall Anomalies

2.1 Defeasible Logic Programming

DeLP [15] is an alternative form of declarative programming. DeLP is a blend of Logic Programming with Defeasible Argumentation, allowing representation of tentative knowledge and leaving for the inference mechanism the task of finding the conclusions that the knowledge base warrants. Two kinds of rules considered by DeLP, makes it different from Logic Programming, from which it inherits the formal characterization of programs as sets of rules. The rules considered include strict rules and defeasible rules. Strict rules are assumed to represent sound knowledge. Defeasible rules are assumed to represent tentative knowledge which may be defeated by other information. DeLP functions by answering queries ($\mathcal{Q}$). Warranted arguments constructed using rules and facts (considered as special cases of strict rules) specify the answer to the query. An answer is yielded by the inference mechanism based on the warrant procedure that is run upon generation of all the possible arguments. The generated arguments may support or contradict the query $\mathcal{Q}$. The characteristic warrant procedure of DeLP, based on Defeasible Argumentation, enables comparison as well as selection of only one of the two contradicting arguments.

Definition 1. Defeasible Logic Program*: A Defeasible Logic Program* $\mathcal{P}$ *is a set* (Π, Δ)*, where* Π *stands for the union of strict rules* Π_{R} *and facts* Π_{F}*;* Δ *denotes defeasible rule. Strict Rules are rules in the classical sense, i.e., whenever the permission of the rules is given, we are allowed to apply the rule and get the conclusion. Strict rule are of the form* $p \leftarrow q_1,\ q_2,\ q_3,..,\ q_{n-1},\ q_n$*. Defeasible Rules are of the form* $p \prec q_1,\ q_2,\ q_3,..,\ q_{n-1},\ q_n$*. Defeasible Rules are contingent rules that get defeated by contrary evidence. Facts (strict rules with empty body) are known truth that are treated as ground literals.*

Construction of arguments in DeLP is a result of the literal derivation and provides a tentative support for the claims. An argument $\mathcal{A}$ for a query $\mathcal{Q}$, (denoted $\langle\mathcal{A},\mathcal{Q}\rangle$) can be considered as a proof for $\mathcal{Q}$ where $\mathcal{A}$ is a set (possibly empty) of ground defeasible rules in conjunction with a set that satisfy the additional constraints of non-contradiction (i.e.an argument s should not allow the derivation of contradictory literals) and minimality (i.e., the set of defeasible information used to derive $\mathcal{Q}$ should be minimal). Mechanism similar to the usual query-driven SLD derivation from logic programming involving backward chaining on both strict and defeasible rules is used to obtain arguments. Incomplete and tentative information of a program $\mathcal{P}$ may lead to an attack on argument $\langle\mathcal{A},\mathcal{Q}\rangle$ by other arguments which may be derived from the same program $\mathcal{P}$. An argument$\langle\mathcal{B},\mathcal{R}\rangle$ is considered to be a counter-argument for $\langle\mathcal{A},\mathcal{Q}\rangle$ if a subargument $\langle\mathcal{A}',\mathcal{Q}'\rangle$ (with $\mathcal{A}'$ belonging to $\mathcal{A}$) in $\langle\mathcal{A},\mathcal{Q}\rangle$ exists, such that $\langle\mathcal{B},\mathcal{R}\rangle$ and $\langle\mathcal{A}',\mathcal{Q}'\rangle$ cannot be accepted simultaneously as acceptance of both will allow inference of contradictory conclusions from $\Pi \cup \mathcal{A}' \cup \mathcal{B}$. The attacking argument $\langle\mathcal{B},\mathcal{R}\rangle$ is termed defeater for $\langle\mathcal{A},\mathcal{Q}\rangle$ if $\langle\mathcal{B},\mathcal{R}\rangle$ is preferred over $\langle\mathcal{A}',\mathcal{Q}'\ \rangle$. Specificity is the commonly used criterion, however, other criteria can also be adopted.

A recursive process is prompted by the search for defeaters in DeLP thus resulting in the formation of a *dialectical tree.* The original argument at issue forms the root and every defeater of the root argument forms a children node. To avoid circular situations during computation of branches in the dialectical tree additional restrictions are added which guarantee that the tree is finite. The children nodes in the tree can be marked as defeated (**D** nodes) or as undefeated(**U** nodes). Marking of the dialectical tree is similar to the AND-OR tree where leaves are always marked as undefeated nodes; inner nodes can be marked as undefeated or as defeated. An undefeated root (original argument, $\langle\mathcal{A},\mathcal{Q}\rangle$) of the tree, after being subjected to the above process, is deemed as acceptable or warranted. DeLP solving for a query $\mathcal{Q}$ with respect to a given program $\mathcal{P}$ accounts for determining whether $\mathcal{Q}$ is supported by a warranted argument. Given query $\mathcal{Q}$ there are four possible answers. 'Yes' if there is at least one warranted argument $\mathcal{Q}$ that follows from $\mathcal{P}$.; 'No' if there is one warrant argument for $\sim\mathcal{Q}$; 'Unknown' if $\mathcal{Q}$ is not present in the program; 'Undecided' if neither $\mathcal{Q}$ nor $\sim\mathcal{Q}$ are supported by warranted arguments in $\mathcal{P}$.

To bring home the point of argumentation in DeLP, let us discuss the argumentation of buying a used car. The car is to be brought only if it is in good condition. The DeLP program has five defeasible rules and three facts.
R1: goodCondition(X) $\prec$ well_maintain(X)
R2: $\sim$goodCondition(X) $\prec$ longrun(X)
R3: goodCondition(X)$\prec$ well_maintain(X),longrun(X)
R4: buy(X)$\prec$car(X),good_condition(X)
R5: $\sim$buy(X) $\prec$ car(X),$\sim$good_condition(X)
F1: car(ford)
F2: well_maintain(ford)
F3: longrun(ford)
Rules R1 and R3 represent the good condition of the car, if the car is well maintained irrespective of long run. Rule R2 states that car is not in good condition if it runs long distance. Rules R4 states that we can buy when it is used but well maintained. Rule R5 represent the scenario when we should not buy the car. Facts F1, F2 and F3 represents that ford is a well maintained car though it runs quite a long distance. DeLP program help us to conclude by performing the query 'buy(ford)' that we have a warranted argument supporting that we should buy that car. The dialectical tree is shown in Figure 1. Dialectical tree shows

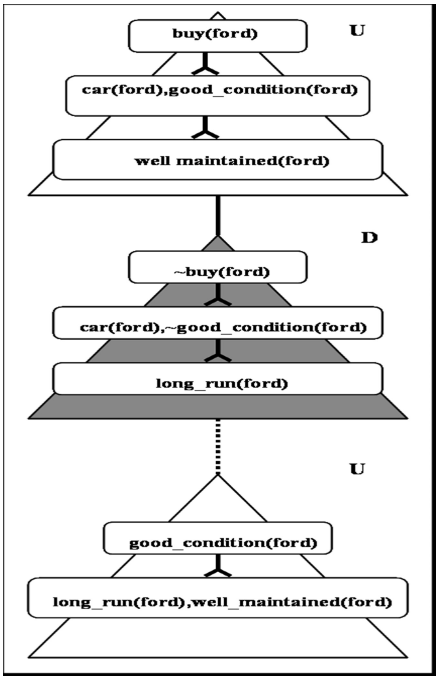
buy(ford)
U
car(ford),good_condition(ford)
well maintained(ford)
D
~buy(ford)
car(ford),~good_condition(ford)
long_run(ford)
U
good_condition(ford)
long_run(ford),well_maintained(ford)

Fig. 1. Dialectical analysis associated with the query 'buy(ford)'

how DeLP, which has support for both logical programming and argumentation, can be used for commonsense reasoning of buying a car.

2.2 Firewall Anomalies

Firewall is a set of ordered filtering rules configured primarily based on predefined security policy. Table 1 shows an example of a basic firewall policy. Each rule is formed of a *condition* and an *action*. The most common and basic firewall are based on mainly five fields: protocol, source IP address, source port, destination IP address, destination port. A rule condition is a set of fields; any fields in IP, UDP or TCP headers may be used. An action is taken for the packets matching these fields. Filtering actions are either to *accept*, which passes the packet or to *deny*, which causes the packet to be discarded.

Table 1. A basic firewall policy example.

Order	Protocol	Source IP	Source Port	Destination IP	Destination Port	Action
R1	TCP	150.172.37.20	any	*.*.*.*	80	deny
R2	TCP	150.172.37.*	any	*.*.*.*	80	accept
R3	TCP	150.172.37.[10,30]	any	171.120.32.[10,40]	21	accept
R4	TCP	150.172.37.*	any	171.120.32.40	80	deny
R5	TCP	150.172.37.30	any	*.*.*.*	21	deny
R6	TCP	150.172.37.[30,60]	any	171.120.32.[40,80]	21	accept
R7	TCP	150.172.37.[25,45]	any	171.120.32.[30,65]	21	deny
R8	TCP	*.*.*.*	any	*.*.*.*	any	deny
R9	UDP	150.172.37.*	any	171.120.32.40	53	accept
R10	UDP	*.*.*.*	any	171.120.32.40	53	accept
R11	UDP	*.*.*.*	any	*.*.*.*	any	deny
R12	TCP	150.172.37.[20,80]	any	*.*.*.*	any	deny
R13	TCP	150.172.37.[20,35]	any	171.120.32.[50,65]	21	accept
R14	TCP	192.168.37.[15,40]	any	171.120.32.[150,165]	21	accept
R15	TCP	192.168.37.[25,60]	any	171.120.32.[120,155]	21	deny

In order to build a model for intra-firewall anomaly, one need to determine all relations that may exist between two filtering rules. This has been addressed among other by [1]; and a set of five relations have been identified. Filtering policy within a firewall is dependent on the ordering of filtering rules. Note that for a set of *completely disjoint* filter rules, the ordering is insignificant. This is not usually the case and therefore ordering is important. Else, some rules may always be 'screened' by other rules producing an incorrect policy. Intra-firewall policy anomaly is the existence of such discrepancies. Anomalies are properties of filters that hint at possible misconfiguration.

3 Formal Characterization

In order to model the firewall within our framework, we derive a formal characterization of the firewall rules and anomalies. Depending on the relation that exists between two rules or a combination of rules with another rule, their orders and their actions, the nature of the anomaly may vary.

Definition 2. *A* stateless firewall rule *of order i, is given as $R_i = \langle f_1, f_2, f_3, f_4, f_5, Action \rangle$; where each f_j is a filtering field. The filtering fields are: protocol, source IP, source port, destination IP and destination port. Action $\in$ {allow, deny} describes the action to be taken if a packet matches the filtering fields in the rule.*

3.1 Anomalies

Simple Anomalies. The set-theoretic relations between the filtering fields of two rules enables one to determine all the relations that can exist between two rules. Al-Shaer et al [1] defined five possible relations that may exist between filtering rules such that there exists exactly one relation between two rules. These result in *simple* anomalies as listed below. Al-Shaer et. al. [1] give formal definitions of possible anomalies between rules in terms of rule relations.

Definition 3. Simple Shadowing Anomaly*: A rule Y is simply shadowed if there is rule X, preceding Y in firewall rule set, such that all the packets that match Y already match X and specify incompatible action.*

Definition 4. Simple Redundancy Anomaly*: A redundant rule X perform same action on same set of packets as another rule Y; therefore it the redundant rule is removed, the security policy will not be affected.*

Definition 5. Generalization*: A rule Y is generalization of a rule X, preceding Y in firewall rule set, such that all the packets of Y is a superset match of X and specify incompatible action.*

Definition 6. Correlation Anomaly*: Rule X and Y are correlated if some fields of X are subset to corresponding fields in Y.*

Second Order Anomalies. Alfaro et al [16] have shown that there could be anomalies where more than a pair of rules are involved. We refer to these as *second order* anomalies.

Definition 7. Second-order generalization*: A rule X and a group of rules Y exhibit second-oder generalization, if decision of rule X is overridden by combination of later rules Y_1 ... Y_n .*

Definition 8. Second-order shadowing*: A rule X and a group of rules Y exhibit second-oder shadowing, if rule X is shadowed by a combination of later rules Y_1 ... Y_n .*

Intra-state Protocol Anomalies. Apart from the above mentioned anomalies another type of anomaly viz. intra-state protocol anomalies may be observed in the stateful filters [17]. This anomaly is basically related to the inner logic of transport layer protocol states[1].

Definition 9. *A* stateful firewall rule *of order i, is given as* $R_i = (f_1, f_2, f_3, f_4, f_5, state, Action)$ *where each* f_j *is a filtering field. The filtering fields are: protocol, source IP, source port, destination IP and destination port.* $state \in \{New, Establish, Related\}$. *Action* $\in$ $\{$*allow, deny*$\}$ *describes the action to be taken if a packet matches the filtering fields in the rule.*

For example in web applications TCP is used as a transport layer for three way handshake connection establishment with the server. The following operations may be distinguished in three way handshake connection scheme to establish connection between the client and the server.

-To start the connection the client sends a SYN packet to a server at LISTEN stage;
-The client waits till it receives the proper SYN-ACK packet from the server;
-As and when the client receive the ACK reply from the server, it again sends back an ACK packet to the server and goes to the ESTABLISHED state of connection.

Table 2. Firewall Rule Table

Rule	*Protocol*	*SrcIPt*	*SrcPort*	*DestIP*	*DestPort*	*State*	*Action*
R1	tcp	192.168.10.[10,20]	any	10.1.2.1	80	New, Establish	accept
R2	tcp	10.1.2.1	any	192.168.10.[10,20]	80	New	deny

The problem occurs, for instance, when the two rules R1 and R2 are present in the firewall. In this case the client will be able to send the SYN packet to the server due to the rule R1. Once this initial SYN packet from the client is received by the firewall it make an entry in the state table and wait for the ACK packet from the server till timeout. But the ACK packet sent back by the server will be filtered out by the firewall rule R2. This will always prevent the establishment of the protocol and will create an overhead in the state table by unnecessarily occupying an entry. In this scenario either both the rules should deny or accept the packets.

[1] Eventhough, a great amount of work has been done for detection of anomalies in firewall configuration, majority of the methods have been limited to stateless cases c.f. [1,13]. Few approaches [18] involve description of stateful firewall models. However, most often than not, this involved straight forward adaptation of the management processes which were previously designed for stateless firewalls. The principle of the approach described here is similar to [17], and derives its origin from the specification of a general automata which describes the different states involved in traffic packages throughout the filtering processing. In our approach,we use defeasible reasoning technique to detect and resolve intra state anomaly in case of stateful firewall

3.2 Anomaly Removal Policies

Given the anomalies stated above, we adopt the following anomaly removal policies.

Policy 1. Shadowing Anomaly Removal*: If rule Y is shadowed by X. Swap between two rules to remove the shadowing anomaly without affecting firewall policy. From Table 1,we can find that Rules R4 is shadowed by Rule R2.To remove shadow anomaly,we should swap their poistion R2 and R4 in rule set If rule Y is shadowed by set of rules* $X_i, X_{i+1}, \ldots X_j$ *such that i<j, placing the rule Y above* X_i *removes shadowing anomaly. For instance from Table 1 , combination of R3 and R6 shadows R7,therefore to remove the anomaly we have to swap the position and rule set.*

Policy 2. Redundancy Anomaly Removal*: If rule Y is redundant with X. Depending on various conditions, following actions are required*

a. *If rule X exactly match with rule Y, then we can remove rule X. Removal of rule will not affect the underlying security policy.*
b. *If rule Y is a subset of X, then security policy will not be affected by removal of Y.*
c. *If rule Y is redundant with set of rules* X_i, X_{i+1}, $\ldots$ X_j *such that* $i < j$, *then remove Y.*
   This may be elucidated by the following example from Table 1.If Rule R13 is a subset of Rule R3 and R6 then the rule set fire wall policy will remain unaffected if we remove Rule R13.Similarly R5 can be removed from rule set without effecting firewall policy as R5 is a subset of R12.

Policy 3. Correlated Anomaly Removal*: If rule X is correlated with Y, then we needed to split the overlapping portion of the rule X and Y to construct a new rule . For Example,to remove the correlated anomaly existing between R14 and R15 we split the rule into three different rules such that the overlapping portion is transformed into a new rule Rule new and the unintersecting part of the two rules R14 and R15 are termed as* $R14_{NEW}$ *and* $R15_{NEW}$ *respectively.*

Policy 4. Protocol Anomaly*: If there is a protocol anomaly between rule X and Y. It can be resolved by the following steps:*

a. *If we find protocol anomaly rule pairs in firewall, then we compare the risk value of the destination hosts; preference will be given to the action associated with the destination with greater risk value.*
b. *If risk value of both the host are equal then priority of both the host are considered; preference will be given to the action associated with the destination with greater priority value.*

For example in above scenario hosts A and B have priority values p_i and p_j respectively in such a way that $p_i < p_j$. In that case the action of the rule where B is the destination will replace the action of the rule where A is the destination.

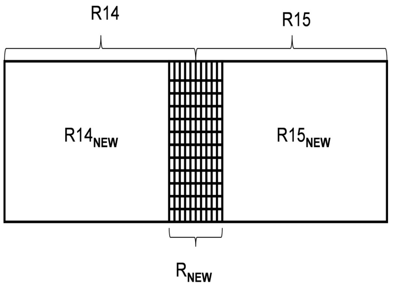

Fig. 2. Corelated Rule Spliting: R14$_{NEW}$; R15$_{NEW}$ and rule for overlap portion to replace R14 and R15

4 Using DeLP for Verification and Reconfiguration

4.1 DeLP Based Architecture

Figure 3 presents a framework for anomaly detection and removal using defeasible argumentation. A Rule Relation Analysis Engine (RRAE) determines the relation between the filtering fields of the rules to derive the ground facts. Anomaly detection and anomaly removal is through DeLP program specifically written for each. These DeLP program are interpreted over a DeLP Interpreter[2] A Rule Analysis and Reconfiguration Engine (RARE) queries the DeLP Interpreter.

Generating the Ground Facts. RRAE generates the facts either through pairwise interaction of fields of the rules OR through a nested operation when *correlated* rules exists. The basic firewall rule relation provision (Definition 10), specifies the set of ground literals for the defeasible logic program. This is for detection of simple anomalies.

Definition 10. *A* basic firewall rule relation provision *is a four tuple* $\langle \mathfrak{R}, \bar{\mathfrak{R}}, \delta, \Pi_{\mathrm{F}} \rangle$ *where,* $\mathfrak{R} = (R_1 \ldots R_n)$ *is the set of n rules in the firewall;* $\bar{\mathfrak{R}} = (R_{i+1} \ldots R_n)$ *for every* R_i *from* $\mathfrak{R}$*,* $i \geqq 1$*.* δ *maps set-theoretic relations between each field of* R_i *and* R_j*;* $i < j \leqq n$*;* $R_j \in \bar{\mathfrak{R}}$ *(ground facts) to* Π_{F}*.*

For second order anomalies, we would generate the relation between a set of correlated rules and a third rule. This is stated by the following formal characterization of another firewall rule relation provision.

Definition 11. *A* secondary firewall rule relation provision *is a four tuple* $\langle \mathfrak{R}, \mathfrak{CR}, \mathfrak{P}, \delta, \Pi_{\mathrm{F}} \rangle$ *where,* $\mathfrak{R} = (R_1 \ldots R_n)$ *is the set of n rules in the firewall;* $\mathfrak{CR} = (CR_1 \ldots CR_{n-1})$*, where* CR_i *is the set of correlated rule pairs of* R_i *in*

[2] The DeLP Interpreter used here has been developed at LIDIA Universidad Nacional del Sur, Bahia Blanca, Argentina.

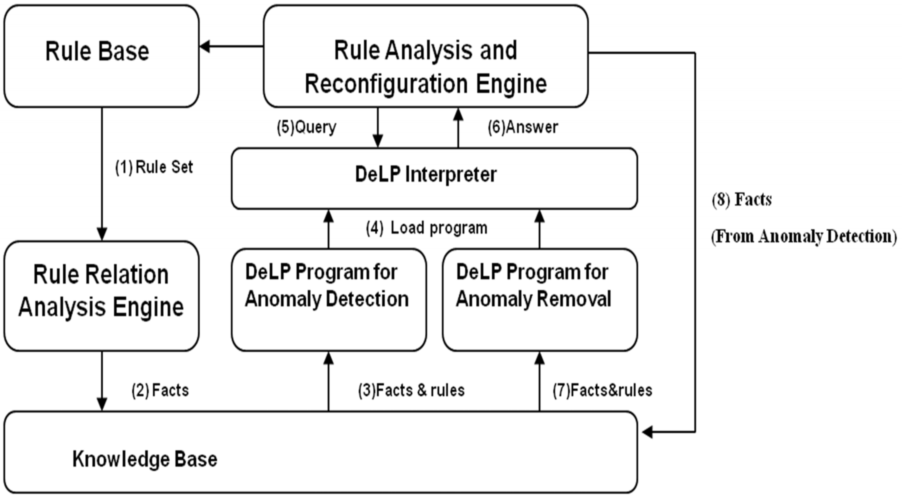

Fig. 3. Architecture of the DeLP Based Verification and Reconfiguration Framework

$\mathfrak{R}$ *i.e.,* $CR_i = \{R_j \ldots R_m\}(j > i; m \leq n)$. $\mathfrak{P}$ *is the set of linear combination of elements of each* CR_i *i.e,* $\mathfrak{P} = \{R_i \cup R_j, R_i \cup R_j \cup R_{j+1}, \ldots, R_i \cup \cup R_j \ldots \cup R_m\}$. δ *maps set-theoretic relations (ground facts) between each field of elements of* $\mathfrak{P}$ *and* $R_k \in \mathfrak{R}$*; where* R_k *is after* R_m *(i.e.,* $k \geq m + 1$*;* $R_k, R_m \in \mathfrak{R}$*.*

Suppose rule Ri and Rj where i<j represent two rules in a firewall rule set. $\mathcal{P}$=(Π, △) represent the defeasible program in context of rule Ri and Rj where Π is the set of facts representing relation between corresponding fields. △ is the set of defeasible rules to detect anomaly between Ri and Rj. We perform $\mathcal{Q}$ = ⟨∼conflictfreerule(Ri)⟩ on $\mathcal{P}$. If answer for $\mathcal{Q}$ is ⟨yes⟩ then we run a query sequence $\mathcal{Q}$s =⟨ shadowing(Ri ,Rj), redundant(Ri,Rj), correlated (Ri,Rj), generalization (Ri,Rj), protocolanomaly(Ri,Rj) ⟩. To illustrate the working of the DeLP framework, we present below representative queries, one each for detection and reconfiguration. Without loss of generality, these are based on the example firewall in Table 1 and Table 2.

Example 1. For query ⟨∼conflictfreerule(R2)⟩; Argument $\mathcal{A}1$ is undefeated by $\mathcal{A}2$ and $\mathcal{A}3$. Argument $\mathcal{A}1$ is attacked by argument $\mathcal{A}2$ which is properly defeated by $\mathcal{A}$ 1. On the other hand $\mathcal{A}1$ is blocking defeater of $\mathcal{A}3$. As seen from the dialectical tree in Figure 4, answer for query ⟨∼conflictfreerule(R2)⟩ is yes. Answer for query ⟨shadowing(R2,R4)⟩ is 'yes'; argument $\mathcal{A}4$ defeat $\mathcal{A}5$ properly.

Table 3. Δ_1: Defeasible Rule Table for Verification

Rule
conflictfreerule(X)≺validrule(X)
∼conflictfreerule (X)—≺ redundant(X,Y),validrule(X)
∼conflictfreerule (X)—≺ shadowing(X,Y),validrule(X)
∼conflictfreerule (X)—≺ correlated(X,Y),validrule(X)
∼conflictfreerule (X)—≺ generalization(X,Y),validrule(X)
∼conflictfreerule (X)—≺ protocolanomaly(X,Y),validrule(X)
validrule(X) ← hasprotocol(X),hassourceip(X),hasdestinationip(X), hasdestinationport(X),hassourceport(X),hasaction(X)
redundant(X,Y)—≺equalprotocol(X,Y),subsetsrcip(X,Y),subsetdestip(X,Y), subsetsrcport(X,Y),subsetdestport(X,Y),equalaction(X,Y)
shadowing(X,Y) —≺ equalprotocol(X,Y),subsetsrcip(X,Y),subsetdestip(X,Y), subsetsrcport(X,Y),subsetdestport(X,Y),differentaction(X,Y)
∼shadowing(X,Y) —≺ correlated(X,Y)
correlated(X,Y)—≺ equalprotocol(X,Y),subsetsrcip(X,Y), subsetsrcport(X,Y),differentaction(X,Y)
correlated(X,Y)—≺ equalprotocol(X,Y),subsetdestip(X,Y), subsetdestport(X,Y),differentaction(X,Y)
correlated(X,Y)—≺equalprotocol(X,Y),subsetsrcip(X,Y),subsetsrcport(X,Y), subsetdestip(X,Y),differentaction(X,Y)
correlated(X,Y)—≺ equalprotocol(X,Y),subsetdestip(X,Y),subsetdestport(X,Y), subsetportip(X,Y),differentaction(X,Y)
generalization(X,Y)—≺ equalprotocol(X,Y),supersetsrcip(X,Y),supersetdestip(X,Y), supersetsrcport(X,Y),supersetdestport(X,Y)
protocolanomaly(X,Y)—≺ equalprotocol(X,Y),srcdestip(X,Y), destsrcip(X,Y),differentaction(X,Y)

$$\mathcal{A}1 = \left\{ \begin{array}{c} \sim conflictfreerule(R2) -\!\!\prec shadowing(R2, R4), \\ validrule(R2) \\ shadowing(R2, R4) -\!\!\prec equalprotocol(R2, R4), \\ subsetsrcip(R2, R4), subsetdestip(R2, R4), \\ subsetsrcport(R2, R4), subsetdestport(R2, R4), \\ differentaction(R2, R4) \end{array} \right\}$$

$$\mathcal{A}2 = \left\{ \begin{array}{c} \sim shadowing(R2, R4) -\!\!\prec correlated(R2, R4) \\ correlated(R2, R4) -\!\!\prec equalprotocol(R2, R4), \\ subsetsrcip(R2, R4), subsetsrcport(R2, R4) \\ differentaction(R2, R4) \end{array} \right\}$$

$$\mathcal{A}3 = \left\{ \begin{array}{c} conflictfreerule(R2) \prec validrule(R2) \\ validrule(R2) -\!\!\prec hasprotocol(R2) \\ hassourceip(R2), hasdestinationip(R2), \\ hasdestinationport(R2), hassourceport(R2) \\ hasaction(R2) \end{array} \right\}$$

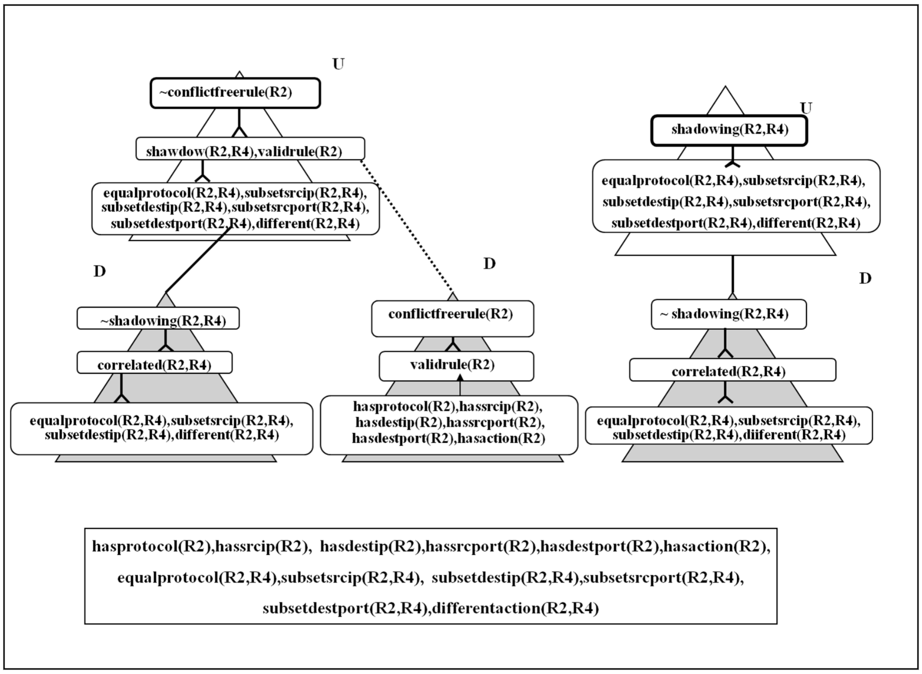
U
~conflictfreerule(R2)
shawdow(R2,R4),validrule(R2)
equalprotocol(R2,R4),subsetsrcip(R2,R4), subsetdestip(R2,R4),subsetsrcport(R2,R4), subsetdestport(R2,R4),different(R2,R4)
D
~shadowing(R2,R4)
correlated(R2,R4)
equalprotocol(R2,R4),subsetsrcip(R2,R4), subsetdestip(R2,R4),different(R2,R4)
D
conflictfreerule(R2)
validrule(R2)
hasprotocol(R2),hassrcip(R2), hasdestip(R2),hassrcport(R2), hasdestport(R2),hasaction(R2)
U
shadowing(R2,R4)
equalprotocol(R2,R4),subsetsrcip(R2,R4), subsetdestip(R2,R4),subsetsrcport(R2,R4), subsetdestport(R2,R4),different(R2,R4)
D
~ shadowing(R2,R4)
correlated(R2,R4)
equalprotocol(R2,R4),subsetsrcip(R2,R4), subsetdestip(R2,R4),diiferent(R2,R4)
hasprotocol(R2),hassrcip(R2), hasdestip(R2),hassrcport(R2),hasdestport(R2),hasaction(R2), equalprotocol(R2,R4),subsetsrcip(R2,R4), subsetdestip(R2,R4),subsetsrcport(R2,R4), subsetdestport(R2,R4),differentaction(R2,R4)

Fig. 4. DeLP dialectical tree on the left support the conclusion that rule R2 is not conflictfree; whereas the DeLP dialectical tree on the right support the conclusion 'shadowing(R2,R4)'.

$$\mathcal{A}4 = \left\{ \begin{array}{c} shadowing(R2, R4) \prec\!\!\!- equalprotocol(R2, R4) \\ subsetsrcip(R2, R4), subsetdestip(R2, R4), \\ subsetsrcport(R2, R4), subsetdestport(R2, R4) \\ differentaction(R2, R4) \end{array} \right\}$$

$$\mathcal{A}5 = \left\{ \begin{array}{c} \sim shadowing(R2, R4) \prec\!\!\!- correlated(R2, R4) \\ correlated(R2, R4) \prec\!\!\!- equalprotocol(R2, R4), \\ subsetsrcip(R2, R4), subsetsrcport(R2, R4), \\ differentaction(R2, R4) \end{array} \right\}$$

4.2 DeLP for Anomaly Removal

The defeasible program $\mathcal{P}_2(\Pi, \triangle_2)$ is used to remove anomalies among rules in the firewall rule set. Whenever '∼conflictfreerule' is true, we have information about type of anomaly. Depending on the 'type' we have different reconfiguration policies which is reflected in the rule set in Table 4. Associated information with regards to 'type' of anomaly such as 'risk value' and 'priority' between rules is used here analogous to 'contextual query' of [14].

Table 4. Δ_2: Defeasible Rule Table for Reconfiguration

Rule
split (X)—≺ reconfigure(X),correlated(X,Y)
split (X)—≺ reconfigure(X),correlated(X,Y),groupofRule(X)
remove (X)—≺ reconfigure(X),redundant(X,Y)
∼remove (X)—≺ reconfigure(X),redundant(X,Y),subset(X,Y)
remove (Y)—≺ reconfigure(X),redundant(X,Y),subset(X,Y),∼remove (X)
remove (X)—≺ reconfigure(X),redundant(X,Y),subset(Y,X)
remove (X)—≺ reconfigure(X),redundant(X,Y),equal(X,Y)
∼remove (Y)—≺ reconfigure(X),redundant(X,Y),subset(X,Y)
∼remove (X)—≺ reconfigure(X),redundant(X,Y),equal(X,Y)
changeaction(X)—≺ reconfigure(X),protocolanomaly(X,Y)
∼changeaction(X)—≺ reconfigure(X),protocolanomaly(X,Y),riskvaluegreater(X,Y)
changeaction(Y)—≺ reconfigure(X),protocolanomaly(X,Y),riskvaluegreater(X,Y),∼changeaction(X)
changeaction(X)—≺ reconfigure(X),protocolanomaly(X,Y),riskvalueequal(X,Y)
changeaction(Y)—≺ reconfigure(X),protocolanomaly(X,Y), riskvalueequal(X,Y),prioritygreater(X,Y),∼changeaction(X)
reconfigure(X)—≺ ∼conflictfreerule (X)
∼reconfigure(X)—≺ ∼conflictfreerule (X),generalization(X,Y)

Example 2. Reconfiguration of rule R1 depends on whether it is involved in conflict with any other rules or the type of conflict it is involved in. For query <reconfiguration(R1)> answer is no as argument $\mathcal{A}6$ is properly defeated by argument $\mathcal{A}7$. See dialectical tree in Figure 5. Dialectical tree on the left support to conclude that eventhough '∼conflictfreerule' is true, in presence of 'generalization', '∼reconfigure' is warranted. Similarly dialectical tree on the right supports the conclusion that rule R12 need not be 'remove' eventhough 'redundant(R12,R5)' is true; whereas 'remove(R5) is warranted. Answer for query <remove(R5)> is yes as argument $\mathcal{A}8$ is defeated by $\mathcal{A}9$ which is defeated by $\mathcal{A}10$. Since defeater for $\mathcal{A}10$ is not present, $\mathcal{A}9$ is reinstated.

$$\mathcal{A}6 = \left\{ reconfigure(R1) \mathrel{-\!\!\prec} {\sim} conflictfreerule(R1) \right\}$$

$$\mathcal{A}7 = \left\{ \begin{array}{c} {\sim} reconfigure(R1) \mathrel{-\!\!\prec} {\sim} conflictfreerule(R1), \\ generalization(R1, R2) \\ generalization(R1, R2) \mathrel{-\!\!\prec} equalprotocol(R1, R2), \\ supersetsrcip(R1, R2), supersetdestip(R1, R2), \\ supersetsrcport(R1, R2), supersetdestport(R1, R2) \end{array} \right\}$$

$$\mathcal{A}8 = \left\{ \begin{array}{c} remove(R5) \mathrel{-\!\!\prec} reconfigure(R12), \\ redundant(R12, R5), subset(R12, R5), \\ {\sim} remove(R12) \\ reconfigure(R12) \mathrel{-\!\!\prec} {\sim} conflictfreerule(R12) \end{array} \right\}$$

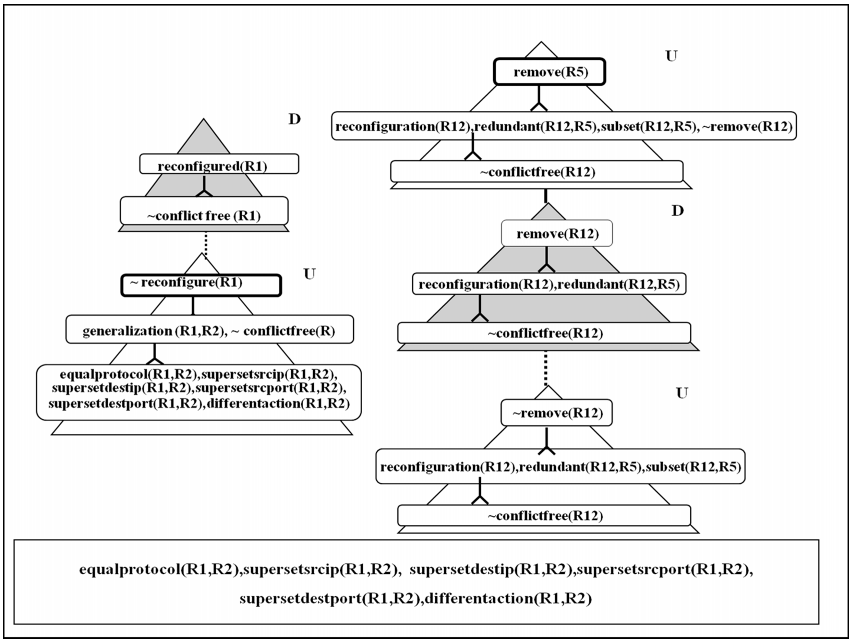
remove(R5)
U
reconfiguration(R12),redundant(R12,R5),subset(R12,R5),~remove(R12)
~conflictfree(R12)
D
remove(R12)
reconfiguration(R12),redundant(R12,R5)
~conflictfree(R12)
U
~remove(R12)
reconfiguration(R12),redundant(R12,R5),subset(R12,R5)
~conflictfree(R12)
D
reconfigured(R1)
~conflict free (R1)
U
~ reconfigure(R1)
generalization (R1,R2), ~ conflictfree(R)
equalprotocol(R1,R2),supersetsrcip(R1,R2), supersetdestip(R1,R2),supersetsrcport(R1,R2), supersetdestport(R1,R2),differentaction(R1,R2)
equalprotocol(R1,R2),supersetsrcip(R1,R2), supersetdestip(R1,R2),supersetsrcport(R1,R2), supersetdestport(R1,R2),differentaction(R1,R2)

Fig. 5. DeLP dialectical tree on the left supporting the conclusion that rule R1 need not be 'reconfigure'. DeLP dialectical tree on the right showing 'remove(R5)' is warranted.

$$\mathcal{A}9 = \left\{ \begin{array}{c} remove(R12) \prec reconfigure(R12), \\ redundant(R12, R5) \\ reconfigure(R12) \prec \sim conflictfreerule(R12) \end{array} \right\}$$

$$\mathcal{A}10 = \left\{ \begin{array}{c} remove(R12) \prec reconfigure(R12), \\ redundant(R12, R5), subset(R12, R5) \\ reconfigure(R12) \prec \sim conflictfreerule(R12) \end{array} \right\}$$

5 Summary and Final Comments

We have discussed how a DeLP approach with an underlying argumentation based semantics could be applied for verification and reconfiguration of a firewall. We have demonstrated with illustrative examples that the DeLP based architecture is capable of performing firewall analysis. Firewall anomalies identified in the literature can be automatically detected. DeLP uses argumentation reasoning and exhibit rules that support a conclusion; leading to identification of source of anomalous configuration of the firewall. Further, under a given set of anomaly resolution policies, firewalls can be automatically reconfigured without violation of initial firewall policies. Defeasible argumentation in reconfiguration recommends the action regarding how to resolve conflicts by considering different facts associated with the rules. Table 5 shows the firewall of Table 1 after verification and reconfiguration.

Table 5. Firewall after verification and reconfiguration

Order	Protocol	Source IP	Source Port	Destination IP	Destination Port	Action
R1	TCP	150.172.37.20	any	*.*.*.*	80	deny
R4	TCP	150.172.37.*	any	171.120.32.40	80	deny
R7	TCP	150.172.37.[25,45]	any	171.120.32.[30,65]	21	deny
R3	TCP	150.172.37.[10,30]	any	171.120.32.[10,40]	21	accept
R2	TCP	150.172.37.*	any	*.*.*.*	80	accept
R6	TCP	150.172.37.[30,60]	any	171.120.32.[40,80]	21	accept
R8	TCP	*.*.*.*	any	*.*.*.*	any	deny
R9	UDP	150.172.37.*	any	171.120.32.40	53	accept
R10	UDP	*.*.*.*	any	171.120.32.40	53	accept
R11	UDP	*.*.*.*	any	*.*.*.*	any	deny
R12	TCP	150.172.37.[20,80]	any	*.*.*.*	any	deny
R14$_{NEW}$	TCP	192.168.37.[15,25]	any	171.120.32.[155,165]	21	accept
R15 $_{NEW}$	TCP	192.168.37.[40,60]	any	171.120.32.[120,150]	21	deny
R$_{NEW}$	TCP	192.168.37.[26,39]	any	171.120.32.[151,154]	21	deny

In context of defeasible argumentation, the work presented here follow the general idea of an argumentative framework as in [15]. As far as our knowledge this is the first adaptation of an argumentation framework for verification and reconfiguration of a firewall. This is worthwhile for the very fact that the main advantage of this approach is the ability of defeasible argumentation to deal with the exponentially increasing size of the firewall policy by reducing unnecessary checks. We have a prototype implementation. The implementation of the system was developed using primarily Java 1.6. This includes many new features of the language including enhance version of collection framework, performance, network interface and serialization which are extensively used in our implementation. Swing is used for development of user interface. Eclipse IDE is used for development of java programs. For defeasible argumentation an abstract machine called Justification Abstract Machine (JAM)[3] is used.

The present work is limited to detection and reconfiguration of intra-firewall anomalies. We envisage that a defeasible argumentative framework can be effectively used for verification and reconfiguration of inter-firewall anomalies, i.e., anomalies arising out of two or more firewalls operating together in a network. Such a framework could be along the lines of meta-level argumentation [6]. This is part of ongoing research.

[3] JAM was specially developed by LIDIA Universidad Nacional del Sur(Bahia Blanca,Argentina) for efficient implementation of DeLP. It is available online. (`http://lidia.cs.uns.edu.ar/delp_client`) [15].

Acknowledgment. This research is supported by DIT, Govt. of India through its project 'Design and Development of a Dynamic Firewall Solution' under Grant No. 13(1)/2011-IG.

References

1. Al-Shaer, E.S., Hamed, H.: Management and translation of filtering security policies. In: IEEE International Conference On Communications (ICC 2003) (2003)
2. Liu, A.X.: Formal Verification of Firewall Policies. In: Proceedings of the 2008 IEEE International Conference on Communications (ICC), Beijing, China (2008)
3. Govaerts, J., Bandara, A., Curran, K.: A formal logic approach to firewall packet filtering analysis and generation. Artificial Intelligence Review 29(3), 223–248 (2008)
4. Hazelhurst, S., Fatti, A., Henwood, A.: Binary decision diagram representations of firewall and router access lists. Technical report, Department of Computer Science, University of the Witwatersrand (1998)
5. Bandara, A.K., Kakas, A.C., Lupu, E.C., Russo, A.: Using argumentation logic for firewall configuration management. In: IFIP/IEEE International Symposium on Integrated Network Management, IM 2009, pp. 180–187. IEEE (2009)
6. Applebaum, A., Li, Z., Syed, A.R., Levitt, P.K.S., Rowe, J., Sklar, E.: Firewall configuration: An application of multiagent metalevel argumentation. In: Proceedings of the 9th Workshop on Argumentation in Multiagent Systems (2012)
7. Eronen, P., Zitting, J.: An expert system for analyzing firewall rules. In: Proc. of the 6th Nordic Workshop on Secure IT Systems, NordSec 2001 (2001)
8. Villemaire, R., Hall, S.: Strong Temporal, Weak Spatial Logic for Rule Based Filters. In: TIME 2009, pp. 115–121 (2009)
9. Khorchani, B., Villemaire, R., Hall, S.: Firewall anomaly detection with a model checker for visibility logic. In: NOMS 2012, pp. 466–469 (2012)
10. Hazarika, S.M.: Carving Rule-based Filters within a Spatio-temporal Logic. In: Proceedings of the National Workshop on Security 2010, pp. 30–35 (2010)
11. Thanasegaran, S., Yin, Y., Tateiwa, Y., Katayama, Y., Takahashi, N.: A topological approach to detect conflicts in firewall policies. In: IEEE International Parallel and Distributed Processing Symposium, pp. 1–7 (2009)
12. Christiansen, M., Emmanuel, F.: An MITDD based firewall using decision diagrams for packet filtering. Telecommun. Systems 27(2-4), 297–319 (2004)
13. Mayer, A., Wool, A., Ziskind, E.: Fang: A Firewall Analysis Engine. In: Proceedings of 21st IEEE Symposium on Security & Privacy, Oakland, CA (2000)
14. Tucat, M., Garcia, A.J., Simari, G.R.: Using Defeasible Logic Programming with Contextual Queries for Developing Recommender Servers. In: Proceedings of the AAAI Fall Symposium (2009)
15. Garca, A., Simari, G.: Defeasible Logic Programming: An Argumentative Approach. Theory and Practice of Logic Programming 4(1), 95–138 (2004)
16. Garcia-Alfaro, J., Boulahia-Cuppens, N., Cuppens, F.: Complete analysis of configuration rules to guarantee reliable network security policies. International Journal of Information Security, 1615–5262
17. Cuppens, F., Cuppens-Boulahia, N., Garcia-Alfaro, J., Moataz, T., Rimasson, X.: Handling Stateful Firewall Anomalies. In: Gritzalis, D., Furnell, S., Theoharidou, M. (eds.) SEC 2012. IFIP AICT, vol. 376, pp. 174–186. Springer, Heidelberg (2012)
18. Gouda, M., Liu, A.: A model of stateful firewalls and its properties. In: DSN, Yokohama, Japan, pp. 128–137 (2005)