Reasoning About Firewall Policies Through Refinement and Composition

Ultan Neville and Simon N. Foley

Department of Computer Science,
University College Cork, Ireland

Abstract. An algebra is proposed for constructing and reasoning about anomaly-free firewall policies. Based on the notion of refinement as safe replacement, the algebra provides operators for sequential composition, union and intersection of policies. The algebra is used to specify and reason about iptables firewall policy configurations. A prototype policy management toolkit has been implemented.

Keywords: Firewalls · Algebra · iptables · Anomalies · Policy-composition

1 Introduction

Firewall configuration management is complex and error-prone, and a misconfigured policy may permit accesses that were intended to be denied or vice-versa. We regard the specification of a firewall policy as a process that evolves. Threats to, and access requirements for, resources behind a firewall do not usually remain static, and over time, a policy or distributed policy configuration may be updated on an ad-hoc basis, possibly by multiple specifiers/administrators. This can be problematic and may introduce anomalies; whereby the intended semantics of the specified access controls become ambiguous.

In this paper, we present a firewall policy algebra $\mathcal{FW}_1$ for constructing and reasoning over anomaly-free policies. The algebra allows policies to be composed in such a way that the result upholds the access requirements of each policy involved; and permits one to reason as to whether some policy is a safe (secure) replacement for another policy in the sense of [11, 14]. The proposed algebra is used to reason about iptables firewall policy configurations. A partial mapping for the iptables filter table is given in the algebra. iptables is a command line utility used to define policies for the Linux kernel firewall Netfilter [1]. We focus on stateful firewall policies that are defined in terms of constraints on source/destination IP/port ranges, the TCP, UDP and ICMP protocols, and additional filter condition attributes.

The primary contribution of this paper is an algebra $\mathcal{FW}_1$, that can be used to reason about firewall policies using refinement and composition operators. The effectiveness of the algebra is demonstrated by its application to anomaly detection, and standards compliance.

The paper is organised as follows. Section 2 introduces the notion of adjacency, which is at the heart of reasoning about/composing firewall rules that involve IP/port ranges. In Section 3 we define datatypes for firewall rule attributes, such as IP/port ranges. Section 4 defines the firewall policy algebra $\mathcal{FW}_1$. In Section 5, we use $\mathcal{FW}_1$ to reason about firewall policies in practice. Section 6 describes a prototype policy management toolkit for iptables and presents some preliminary results. Related work is outlined in Section 7 and Section 8 concludes the paper. The Z notation [19] is used to present the algebra and has been syntax- and type-checked using the *f*UZZ tool.

2 A Theory of Adjacency

A firewall policy is conventionally defined as a sequence of order-dependent rules. A rule is composed of filter conditions and a target action. Filter conditions usually consist of fields/attributes from IP, TCP/UDP headers; with the most commonly used attributes being source/destination IP/port, and network protocol. Target actions are usually *allow* or *deny* [3,8].

Range-based filter condition attributes (IPs/ports) have logical mappings to intervals of $\mathbb{N}$. For example, the port range that includes all ports from SSH up and including HTTP can be written as the interval $[22 \mathinner{..} 80]$. Consider as part of a running example, a system that is capable of enforcing firewall rules where the filter condition attribute for the rules is destination port range. Then if we had a rule that allowed all ports from SSH to HTTP, it may look like: $(i,\ [22 \mathinner{..} 80],\ allow)$, where i is the index of the rule in the policy, $[22 \mathinner{..} 80]$ is the required port range, and *allow* means that network traffic matching this pattern be permitted traversal of the firewall. Suppose we had a second rule, that specifies *allow* everything from Quote Of The Day (QOTD) up to and including FTP Control. Then $(j,\ [17 \mathinner{..} 21],\ allow)$, specifies that for the rule at index j; the required port range $[17 \mathinner{..} 21]$ is allowed. Intuitively, we can see that the port ranges for the rules at index i and index j are *adjacent*, and we may want to join rules i and j into a single rule that looks like $(k,\ [17 \mathinner{..} 80],\ allow)$. This notion of adjacency becomes more complex when we consider comparing/composing firewall rules comprising $2 \mathinner{..} n$ filter condition attributes.

2.1 The Adjacency Specification

In this section we define the filter condition attribute relationships of *adjacency, disjointness* and *subsumption.* These relationships are at the heart of adjacency, and ultimately the $\mathcal{FW}_1$ algebra.

Let $\mathcal{IV}[min, max]$ be the set of all intervals on the natural numbers, from min up to and including max. Intervals are defined by their corresponding sets.

$$\mathcal{IV}[min, max] == \{S : \mathbb{P}\,\mathbb{N} \mid \exists \bot, \top : S \bullet \forall x : S \bullet min \leq \bot \leq x \leq \top \leq max\}$$

For example, $\mathcal{IV}[1,3]$ gives $\{[1 \mathinner{..} 1], [1 \mathinner{..} 2], [1 \mathinner{..} 3], [2 \mathinner{..} 2], [2 \mathinner{..} 3], [3 \mathinner{..} 3]\}$. For ease of exposition and when no ambiguity arises, we may write an interval as a pair

$[\bot \mathrel{..} \top]$, rather than by the set it defines. Let *IPv*4 define the set of all possible IPv4 address ranges, and similarly, let *Port* define the set of all possible network port ranges, where $IPv4 == \mathcal{IV}[0, 2^{32} - 1] \land Port == \mathcal{IV}[0, 2^{16} - 1]$.

Adjacency Two intervals are adjacent if their union defines a single interval. We generalize this to any attribute of type X, whereby for $a, b \in X$, if $a \wr_X b$, then a and b are adjacent in the set X.

$$
\begin{array}{l}
[X] \\
\hline
\_ \wr_{\_} \_ : \mathbb{P}\, X \rightarrow\!\!\!\!\!+ (X \leftrightarrow X) \\
\hline
\forall\, a, b : X \bullet \\
\quad a \wr_X a \land (a \wr_X b \Rightarrow b \wr_X a)
\end{array}
$$

For example, interval $[1 \mathrel{..} 2]$ is adjacent to interval $[3 \mathrel{..} 3]$, thus $[1 \mathrel{..} 2] \wr_{\mathcal{IV}[1,3]} [3 \mathrel{..} 3]$. It follows for $a, b \in \mathbb{N}$ that $a \wr_{\mathbb{N}} b \Leftrightarrow (a = b \lor a + 1 = b \lor b + 1 = a)$, and given $S, T \in \mathbb{P}\, X$ then $S \wr_{\mathbb{P}\, X} T \Leftrightarrow true$.

Disjointness Two intervals are disjoint if they don't intersect. Given $a, b \in X$, $a \mid_X b$ denotes a and b are disjoint in X.

$$
\begin{array}{l}
[X] \\
\hline
\_ \mid_{\_} \_ : \mathbb{P}\, X \rightarrow\!\!\!\!\!+ (X \leftrightarrow X) \\
\hline
\forall\, a, b : X \bullet \\
\quad \neg\, (a \mid_X a) \land (a \mid_X b \Rightarrow b \mid_X a)
\end{array}
$$

For example, $[1 \mathrel{..} 2]$ and $[3 \mathrel{..} 3]$ are disjoint, thus $[1 \mathrel{..} 2] \mid_{\mathcal{IV}[1,3]} [3 \mathrel{..} 3]$. It follows for $a, b \in \mathbb{N}$ that $a \mid_{\mathbb{N}} b \Leftrightarrow a \neq b$, and given $S, T \in \mathbb{P}\, X$ then $S \mid_{\mathbb{P}\, X} T \Leftrightarrow S \cap T = \emptyset$.

Subsumption An interval I subsumes (covers) an interval J, if $J \subseteq I$. For $a, b \in X$, if $a \overset{X}{\leftarrow} b$ then b covers a in X. The properties of reflexivity, transitivity and antisymmetry define $\overset{X}{\leftarrow}$ as a non-strict partial order over X [5].

$$
\begin{array}{l}
[X] \\
\hline
\_ \overset{\_}{\leftarrow} \_ : \mathbb{P}\, X \rightarrow\!\!\!\!\!+ (X \leftrightarrow X) \\
\hline
\forall\, a, b, c : X \bullet \\
\quad a \overset{X}{\leftarrow} a \land (a \overset{X}{\leftarrow} b \land b \overset{X}{\leftarrow} c \Rightarrow a \overset{X}{\leftarrow} c) \land (a \overset{X}{\leftarrow} b \land b \overset{X}{\leftarrow} a \Rightarrow a = b)
\end{array}
$$

For example, $[1 \mathrel{..} 3]$ covers $[3 \mathrel{..} 3]$, thus $[3 \mathrel{..} 3] \overset{\mathcal{IV}[1,3]}{\longleftarrow} [1 \mathrel{..} 3]$. It follows for $a, b \in \mathbb{N}$ that $a \overset{\mathbb{N}}{\leftarrow} b \Leftrightarrow a = b$, and given $S, T \in \mathbb{P}\, X$, then $S \overset{\mathbb{P}\, X}{\leftarrow} T \Leftrightarrow S \subseteq T$.

For a set X and $S \in \mathbb{P}\, X$, the flattening function $\lceil S \rceil$ gives the cover-set for the elements of S.

$$
\begin{array}{l}
[X] \\
\hline
\lceil \_ \rceil : \mathbb{P}\, X \rightarrow\!\!\!\!\!+ \mathbb{P}\, X \\
\hline
\forall\, S : \mathbb{P}\, X \bullet \\
\quad \lceil S \rceil = S \setminus \{a, a' : S \mid a \overset{X}{\leftarrow} a' \land a \neq a' \bullet a\}
\end{array}
$$

For example, $\lceil \mathcal{IV}[1,3] \rceil = \{[1\,..\,3]\}$. We define a *difference* operator for $S, T \in \mathbb{P}\,X$, where $S \setminus_{\mathbb{P}\,X} T$ gives the relative compliment of T in S.

$$
\begin{array}{|l}
[X] \\
\hline
\_\setminus\_\_ : \mathbb{P}(\mathbb{P}\,X) \nrightarrow \mathbb{P}\,X \times \mathbb{P}\,X \rightarrow \mathbb{P}\,X \\
\hline
\forall\, S, T : \mathbb{P}\,X \bullet \\
\quad S \setminus_{\mathbb{P}\,X} T = \lceil \{a : S;\ c : X \mid c \overset{X}{\leftarrow} a \wedge (\forall\, b : T \bullet \neg\,(c \overset{X}{\leftarrow} b)) \bullet c\} \rceil \\
\hline
\end{array}
$$

For example, $\lceil \mathcal{IV}[1,3] \rceil \setminus_{\mathcal{IV}[1,3]} \{[1\,..\,1], [3\,..\,3]\} = \{[2\,..\,2]\}$.

3 Filter Condition Attribute Datatypes

In this section we define the datatypes used to construct the filter condition attributes for the $\mathcal{FW}_1$ policy model.

3.1 The Adjacency Datatype

For a set X, the Adjacency datatype $\alpha[X]$, is the set of all closed subsets of X partitioned by adjacency.

$$\alpha[X] == \{S : \mathbb{P}\,X \mid (\forall\, a, b : S \mid a \neq b \bullet \neg\,(a \wr_X b))\}$$

For example $\alpha[\mathcal{IV}[1,3]]$ gives $\{\{[1\,..\,1]\}, \{[1\,..\,2]\}, \{[1\,..\,3]\}, \{[2\,..\,2]\}, \{[2\,..\,3]\}, \{[3\,..\,3]\}, \{[1\,..\,1],[3\,..\,3]\}\}$, and $\alpha[\mathit{IPv4}]$ defines the set of all closed subsets for the intervals of the IPv4 address range partitioned by adjacency.

Adjacency Ordering An ordering can be placed over Adjacency-sets, and is defined as follows.

$$
\begin{array}{|l}
[X] \\
\hline
\bot, \top : \alpha[X] \\
\mathbf{not} : \alpha[X] \rightarrow \alpha[X] \\
\_ \leq \_ : \alpha[X] \leftrightarrow \alpha[X] \\
\_ \otimes \_, \\
\_ \oplus \_ : \alpha[X] \times \alpha[X] \rightarrow \alpha[X] \\
\hline
\bot = \emptyset \wedge \top = \lceil X \rceil \\
\forall\, S, T : \alpha[X] \bullet \\
\quad \mathbf{not} S = \top \setminus_{\alpha[X]} S\ \wedge \\
\quad S \leq T \Leftrightarrow (\forall\, a : S \bullet \exists\, b : T \bullet a \overset{X}{\leftarrow} b)\ \wedge \\
\quad S \otimes T = \lceil \bigcup \{U : \alpha[X] \mid \forall\, c : U \bullet \exists\, a : S;\ b : T \bullet c \overset{X}{\leftarrow} a \wedge c \overset{X}{\leftarrow} b\} \rceil\ \wedge \\
\quad S \oplus T = \bigcap \{U : \alpha[X] \mid \forall\, c : U \bullet \exists\, a : S;\ b : T \bullet a \overset{X}{\leftarrow} c \vee b \overset{X}{\leftarrow} c\} \\
\hline
\end{array}
$$

The elements $\bot, \top \in \alpha[X]$ define the least and greatest bounds, respectively, on $\alpha[X]$, where for any $S \in \alpha[X]$, then $\bot \leq S \leq \top$. Adjacency negation defines a valid complement operator in $\alpha[X]$, where $(S \oplus \mathbf{not}\, S) = \top$ and $(S \otimes \mathbf{not}\, S) = \bot$.

Adjacency Intersection Under this ordering, the *meet*, or intersection $S \otimes T$ of $S, T \in \alpha[X]$ is defined using subsumption, as the cover-set for the generalized union of all Adjacency-sets, where each element of $(S \otimes T)$ is covered by an element in *both* S and T. Intuitively, this means that the values of the meet are all non-empty intersections of each value in S with each value in T. Under the ordering relation $\leq$, $\otimes$ provides a *greatest lower bound* (glb) operator, and $S \otimes T$ is covered by both S and T, that is $(S \otimes T) \leq S$ and $(S \otimes T) \leq T$.

Adjacency Union The *join* of $S, T \in \alpha[X]$ is defined using subsumption, as the generalized intersection of all Adjacency-sets, where each element of $(S \oplus T)$ covers an element in *either* S or T. Intuitively, this means that the values of the join are exactly a union of the elements from both S and T. Given the definition of ordering using subsumption, it follows that the Adjacency join provides a *lowest upper bound* (lub) operator. Since $\oplus$ provides a lub operator we have $S \leq (S \oplus T)$ and $T \leq (S \oplus T)$.

Proposition The poset $(\alpha[X], \leq)$ forms a distributive lattice with compliment operator **not**. This follows from the definition of $\leq$ as a subsumption ordering/an antisymmetric preorder, the properties of **not**, the intuitive definition of the meet of $S, T \in \alpha[X]$ as all non-empty intersections of each value in S with each value in T, and the intuitive definition of the join operation as an exact union of the elements from both S and T [18].

Given the adjacency, disjointness and subsumption relations; then for $S, T \in \alpha[X]$, we define $S \wr_{\alpha[X]} T \Leftrightarrow \mathit{true} \wedge S \mid_{\alpha[X]} T \Leftrightarrow S \otimes T = \bot \wedge S \overset{\alpha[X]}{\leftarrow} T \Leftrightarrow S \leq T$.

3.2 The Duplet Datatype

A duplet is an ordered pair, where the set of all duplets for types X, Y, is defined as $\delta[X, Y]$, where $\delta[X, Y] == X \times Y$. For example, $\delta[\mathcal{IV}[1,1], \mathcal{IV}[1,2]]$ gives $\{([1\,..\,1],[1\,..\,1]), ([1\,..\,1],[1\,..\,2]), ([1\,..\,1], [2\,..\,2])\}$, and $\delta[\alpha[\mathit{IPv}4], \alpha[\mathit{Port}]]$ gives the set of all duplets for adjacency-free IP/port-ranges.

Recall the earlier example of the firewall system that supports only destination port range filter conditions. Suppose we want to extend the expressiveness of the policy rules for this system to include a definition for destination IP range. Then $\alpha[\delta[\alpha[\mathit{IPv}4], \alpha[\mathit{Port}]]]$, is the set of all closed subsets of adjacency-free IP/port-range duplets, partitioned by adjacency. Consider two policy requirements, where network traffic is to be allowed to the IP range $[1\,..\,3]$ on ports $[1\,..\,3]$, and to the IP range $[2\,..\,4]$ on ports $[2\,..\,4]$. Then modelling this using sets of adjacency-free duplets, we have $S, T \in \alpha[\delta[\alpha[\mathit{IPv}4], \alpha[\mathit{Port}]]]$, where $S == \{(\{[1\,..\,3]\}, \{[1\,..\,3]\})\}$ and $T == \{(\{[2\,..\,4]\}, \{[2\,..\,4]\})\}$.

Duplet Disjointness A pair of duplets are disjoint if the attributes in the first coordinate are disjoint, or the attributes in the second coordinate are disjoint. For $(a_1, b_1), (a_2, b_2) \in \delta[X, Y]$, then:

$$(a_1, b_1) \mid_{\delta[X,Y]} (a_2, b_2) \Leftrightarrow (a_1 \mid_X a_2 \vee b_1 \mid_Y b_2)$$

For example, $\neg\ (S \mid_{\alpha[\delta[\alpha[IPv4],\alpha[Port]]]} T)$.

Duplet Adjacency A pair of duplets are adjacent if the attributes in the first coordinate are adjacent, and the attributes in the second coordinate are not disjoint. Thus, we have:

$$(a_1, b_1) \wr_{\delta[X,Y]} (a_2, b_2) \Leftrightarrow (a_1 \wr_X a_2 \land \neg\ (b_1 \mid_Y b_2))$$

For example, $S \wr_{\alpha[\delta[\alpha[IPv4],\alpha[Port]]]} T$.

Duplet Subsumption A duplet is distinguished from a standard ordered pair, whereby we explicitly define orderings separately in each coordinate. For example, suppose we wanted to join adjacent policies S and T, then under a 'standard' Cartesian product ordering we have $S \oplus T = \{(\{[1\,..\,4]\}, \{[1\,..\,4]\})\}$. This obviously results in an overly permissive policy, conversely; an overly restrictive policy if we were composing deny rules. A duplet (a_1, b_1) covers a duplet (a_2, b_2) in $\delta[X, Y]$, if a_1 covers a_2 in X, and b_2 covers b_1 in Y. Thus:

$$(a_2, b_2) \overset{\delta[X,Y]}{\leftarrow} (a_1, b_1) \Leftrightarrow (a_2 \overset{X}{\leftarrow} a_1 \land b_1 \overset{Y}{\leftarrow} b_2)$$

Then we have $S \oplus T = \{(\{[1\,..\,4]\}, \{[2\,..\,3]\}), (\{[1\,..\,3]\}, \{[1\,..\,1]\}), (\{[2\,..\,4]\}, \{[4\,..\,4]\})\}$. Thus, the join, or union $(S \oplus T)$ of S and T, defines the adjacency-free coalescence of all duplets from S and T. For reasons of space, we do not give the implementation definition for this operation.

3.3 The Stateful/Protocol Datatype

In this section, we define the network protocols of interest for the model and encode a notion of *state*. The iptables command line utility allows the end-user to specify one (or all) of seven different protocols in a rule [1]. For reasons of space, we focus only on the TCP, UDP and ICMP protocols.

Let *Flags* be the set of TCP flags, where *Flags* ::= syn | ack | fin | psh | rst | urg. The TCP protocol is defined as the set of all sets of pairs of sets of *Flags*, whereby for each pair; the first set contains the flags that are to be examined in a packet-header, and the second set contains the flags that must be set (in a packet-header). In [1], these are referred to as the *comp* and *mask* values for a packet, respectively. Let *TCP* be the set of all sets of comp/mask pairs, where $TCP == \mathbb{P}(\mathbb{P}\, Flags \times \mathbb{P}\, Flags)$. Let $[TypesCodes]$ be the set of all valid ICMP Type/Code pairs. For simplicity and reasons of space, we do not consider how the values of *TypesCodes* may be constructed, other than to assume that the usual human-readable notation can be used, such as (8,0) and (17,0) $\in TypesCodes$. The iptables conntrack modules' statelist [1] may be defined as follows. Let *State* be the set of connection tracking states for a packet/connection, where *State* ::= new | established | related | invalid | untracked.

Let *Protocol* define the set of all protocols, given as the set of all duplets over *TCP*, UDP ($[0\,..\,1]$), ICMP ($\mathbb{P}\ TypesCodes$) and the set of all sets of connection tracking states ($\mathbb{P}\, State$).

$$Protocol == \delta[TCP, \delta[\{0, 1\}, \delta[\mathbb{P}\ TypesCodes, \mathbb{P}\ State]]]$$

Proposition The *Protocol* datatype forms a product-lattice structure. This follows from the definition of *Protocol* as the product of powerset/binary lattices [5].

4 The $\mathcal{FW}_1$ Policy Algebra

In this section we define an algebra $\mathcal{FW}_1$, for constructing and reasoning about anomaly-free firewall policies. We focus on stateful firewall policies that are defined in terms of constraints on source/destination IP/port ranges, the TCP, UDP and ICMP protocols.

A filter condition is a five-tuple (s, $sprt$, d, $dprt$, p), representing network traffic originating from source IP range s, with source port range $sprt$, destined for destination IP range d, with destination port range $dprt$, using stateful-protocols p. Let FC define the set of all filter conditions, where:

$$FC == \delta[\alpha[IPv4], \delta[\alpha[Port], \delta[\alpha[IPv4], \delta[\alpha[Port], Protocol]]]]$$

A firewall policy defines the filter conditions that may be allowed or denied by a firewall. Let *Policy* define the set of all firewall policies, whereby:

$$Policy == \{A, D : \alpha[FC] \mid \forall\, a : A;\ d : D \bullet a \mid_{FC} d\}$$

A firewall policy $(A, D) \in Policy$ defines that a filter condition $f \in A$ should be allowed by the firewall, while a filter condition $f \in D$ should be denied. Given $(A, D) \in Policy$ then A and D are disjoint: this avoids any contradiction in deciding whether a filter condition should be allowed or denied. Given that A and D are also both adjacency-free; then *Policy* defines the set of *anomaly-free* firewall policies in the sense that they contain no redundancy, shadowing, or other anomalies [4].

Note that $(A, D) \in Policy$ need not partition $\lceil FC \rceil$: the allow and deny sets define the filter conditions to which the policy *explicitly* applies, and an *implicit* default decision is applied for those filter conditions in $\lceil FC \rceil \setminus_{\alpha[FC]} (A \oplus D)$. For the purposes of modelling iptables firewalls it is sufficient to assume *default deny*, though we observe that $\mathcal{FW}_1$ can also be used to reason about *default allow* firewall policies. The policy destructor functions *allow* and *deny* are analogous to functions *first* and *second* for ordered pairs:

$$\begin{array}{|l}
allow, deny : Policy \to \alpha[FC] \\
\hline
\forall\, A, D : \alpha[FC] \bullet \\
\quad allow\,(A, D) = A \wedge deny\,(A, D) = D
\end{array}$$

Policy Refinement An ordering can be defined over firewall policies, whereby given $P, Q \in Policy$ then $P \sqsubseteq Q$ means that P is no less restrictive than Q, that is, any filter condition that is denied by Q is denied by P. Intuitively, policy P is considered to be a *safe replacement* for policy Q, in the sense of [11, 14] and any firewall that enforces policy Q can be reconfigured to enforce policy P without any loss of security. The set *Policy* forms a lattice under the safe replacement ordering and is defined as follows.

$$\begin{array}{|l}
\mathcal{FW}_1 \\
\hline
\bot, \top : \mathit{Policy} \\
\_ \sqsubseteq \_ : \mathit{Policy} \leftrightarrow \mathit{Policy} \\
\_ \sqcap \_, \\
\_ \sqcup \_ : \mathit{Policy} \times \mathit{Policy} \to \mathit{Policy} \\
\hline
\bot = (\emptyset, \lceil FC \rceil) \land \top = (\lceil FC \rceil, \emptyset) \\
\forall P, Q : \mathit{Policy} \bullet \\
\quad P \sqsubseteq Q \Leftrightarrow ((\mathit{allow}\, P \leq \mathit{allow}\, Q) \land (\mathit{deny}\, Q \leq \mathit{deny}\, P)) \land \\
\quad P \sqcap Q = (\mathit{allow}\, P \otimes \mathit{allow}\, Q, \mathit{deny}\, P \oplus \mathit{deny}\, Q) \land \\
\quad P \sqcup Q = (\mathit{allow}\, P \oplus \mathit{allow}\, Q, \mathit{deny}\, P \otimes \mathit{deny}\, Q) \\
\hline
\end{array}$$

Formally, $P \sqsubseteq Q$ iff every filter condition allowed by P is allowed by Q and that any filter conditions explicitly denied by Q are also explicitly denied by P. Note that in this definition we distinguish between filter conditions *explicitly* denied in the policy versus those *implicitly* denied by default. This means that, everything else being equal, a policy that explicitly denies a filter condition is considered more restrictive than a policy that relies on the implicit default-deny for the same network traffic pattern. Safe replacement is defined as the Cartesian product of Adjacency orderings over allow and deny sets and it therefore follows that $(\mathit{Policy}, \sqsubseteq)$ is a poset.

$\bot$ and $\top$ define the most restrictive and least restrictive policies, that is, for any $P \in \mathit{Policy}$ we have $\bot \sqsubseteq P \sqsubseteq \top$. Thus, for example, any firewall enforcing a policy P can be safely reconfigured to enforce the (not very useful) firewall policy $\bot$.

Policy Intersection Under this ordering, the meet $P \sqcap Q$, of two firewall policies P and Q is defined as the policy that denies any filter condition that is explicitly denied by *either* P or Q, but allows filter conditions that are allowed by *both* P and Q. Intuitively, this means that if a firewall is required to enforce both policies P and Q, it can be configured to enforce the policy $(P \sqcap Q)$ since $P \sqcap Q$ is a safe replacement for both P and Q, that is; $(P \sqcap Q) \sqsubseteq P$ and $(P \sqcap Q) \sqsubseteq Q$. Given the definition of safe replacement as a product of two Adjacency lattices, it follows that the policy meet provides the glb operator. Thus, $P \sqcap Q$ provides the 'best'/least restrictive safe replacement (under $\sqsubseteq$) for both P and Q.

Policy Union The join of two firewall policies P and Q is defined as the policy that allows any filter condition allowed by *either* P or Q, but denies filter conditions that are explicitly denied by *both* P and Q. Intuitively, this means that a firewall that is required to enforce either policy P or Q can be safely configured to enforce the policy $(P \sqcup Q)$. Since $\sqcup$ provides a lub operator we have $P \sqsubseteq (P \sqcup Q)$ and $Q \sqsubseteq (P \sqcup Q)$.

Proposition The set of all policies *Policy* forms a lattice under safe replacement. This follows from the definition of $\sqsubseteq$ as a Cartesian product of two Adjacency lattice orderings.

4.1 Constructing Firewall Policies

The lattice of policies $\mathcal{FW}_1$ provides us with an algebra for constructing and interpreting firewall polices. The following constructor functions are used to build primitive policies. Given a set of adjacency-free filter conditions A, then $(\mathsf{Allow}\, A)$ is a policy that allows filter conditions in A, and $(\mathsf{Deny}\, D)$ is a policy that explicitly denies filter conditions in D.

$$\begin{array}{|l}
\mathsf{Allow},\\
\mathsf{Deny} : \alpha[FC] \rightarrow \mathit{Policy}\\
\hline
\forall\, S : \alpha[FC] \bullet\\
\quad \mathsf{Allow}\, S = (S, \emptyset) \wedge \mathsf{Deny}\, S = (\emptyset, S)
\end{array}$$

This provides what we refer to as a *weak* interpretation of allow and deny. Network traffic patterns that are not explicitly mentioned in parameter S are default-deny and therefore are not specified in the deny set of the policy. The following provides us with a *strong* interpretation for these constructors:

$$\begin{array}{|l}
\mathsf{Allow}^+,\\
\mathsf{Deny}^+ : \alpha[FC] \rightarrow \mathit{Policy}\\
\hline
\forall\, S : \alpha[FC] \bullet\\
\quad \mathsf{Allow}^+\, S = (S, \mathbf{not}\, S) \wedge \mathsf{Deny}^+\, S = (\mathbf{not}\, S, S)
\end{array}$$

In this case $(\mathsf{Allow}^+\, A)$ allows filter conditions specified in A, while explicitly denying all other filter conditions, and $(\mathsf{Deny}^+\, D)$ denies filter conditions specified in D while allowing all other filter conditions.

Proposition A firewall policy $P \in \mathit{Policy}$ can be decomposed into it's corresponding allow and deny sets, and re-constructed using the algebra; for any $(A, D) \in \mathit{Policy}$, since A and D are disjoint then:

$$\begin{aligned}
(\mathsf{Allow}^+\, A) \sqcup (\mathsf{Deny}\, D) &= (A, \lceil FC \rceil \setminus_{\alpha[FC]} A) \sqcup (\emptyset, D)\\
&= (A, D)\\
&= (\mathsf{Allow}\, A) \sqcap (\mathsf{Deny}^+\, D)
\end{aligned}$$

5 Reasoning About Policies in Practice

Sequential Composition A firewall policy is conventionally constructed as a sequence of rules, whereby for a given network packet, the decision to allow or deny that packet is checked against each policy rule, starting from the first, in sequence, and the first rule that matches gives the result that is returned. The algebra $\mathcal{FW}_1$ can be extended to include a similar form of sequential composition of policies. The policy constructions above can be regarded as representing the individual rules of a conventional firewall policy.

Let $(\mathsf{Allow}\, A) \,⨟\, Q$ denote a sequential composition of an allow rule $(\mathsf{Allow}\, A)$ with policy Q with the interpretation that a given network packet matched in

A is allowed; if it does not match in A then policy Q is enforced. The resulting policy either: allows filter conditions in A (and denies all other filter conditions), or allows/denies filter conditions in accordance with policy Q. We define:

$$\begin{aligned}(\mathsf{Allow}\ A) \fatsemi Q &= (\mathsf{Allow}^{+}\ A) \sqcup Q \\ &= ((A \oplus \mathit{allow}(Q)), ((\lceil FC \rceil \setminus_{\alpha[FC]} A) \otimes \mathit{deny}(Q))) \\ &= ((A \oplus \mathit{allow}(Q)), (\mathit{deny}(Q) \setminus_{\alpha[FC]} A))\end{aligned}$$

which is as expected. A similar definition can be provided for the sequential composition $(\mathsf{Deny}\ D) \fatsemi Q$, whereby a given network packet that is matched in D is denied; if it does not match in D then policy Q is enforced. We define:

$$\begin{aligned}(\mathsf{Deny}\ D) \fatsemi Q &= (\mathsf{Deny}^{+}\ D) \sqcap Q \\ &= (\mathit{allow}(Q) \setminus_{\alpha[FC]} D, \mathit{deny}(Q) \oplus D)\end{aligned}$$

While in practice its usual to write a firewall policy in terms of many constructions of allow and deny rules, in principle, any firewall policy $P \in \mathit{Policy}$ can be defined in terms of one allow policy ($\mathsf{Allow}\ \mathit{allow}(P)$) and one deny policy ($\mathsf{Deny}\ \mathit{deny}(P)$) and since the allow and deny sets of P are disjoint we have $P \fatsemi Q = (\mathsf{Deny}\ \mathit{deny}(P)) \fatsemi (\mathsf{Allow}\ \mathit{allow}(P)) \fatsemi Q$. We define this as:

$$\begin{array}{|l} \_ \fatsemi \_ : \mathit{Policy} \times \mathit{Policy} \rightarrow \mathit{Policy} \\ \hline \forall \mathcal{FW}_1;\ P, Q : \mathit{Policy} \bullet \\ \quad P \fatsemi Q = (Q \sqcup (\mathsf{Allow}^{+}\ (\mathit{allow}(P)))) \sqcap (\mathsf{Deny}^{+}\ (\mathit{deny}(P))) \end{array}$$

Let Rule define the set of all firewall rules, where $\mathit{Rule} ::= \mathsf{allow}\ \langle\langle FC \rangle\rangle \mid \mathsf{deny}\ \langle\langle FC \rangle\rangle$. We define a rule interpretation function as:

$$\begin{array}{|l} \mathcal{I} : \mathit{Rule} \rightarrow \mathit{Policy} \\ \hline \forall f : FC \bullet \\ \quad \mathcal{I}(\mathsf{allow}\, f) = \mathsf{Allow}(\{f\}) \wedge \mathcal{I}(\mathsf{deny}\, f) = \mathsf{Deny}(\{f\}) \end{array}$$

A firewall policy is defined as a sequence of rules $\langle r_1, r_2, .., r_n \rangle$, for $r_i \in \mathit{Rule}$, and is encoded in the policy algebra as $\mathcal{I}(r_1) \fatsemi \mathcal{I}(r_2) \fatsemi .. \fatsemi \mathcal{I}(r_n)$.

Policy Negation The policy negation of $P \in \mathit{Policy}$ allows filter conditions explicitly denied by P and explicitly denies filter conditions allowed by P. We define:

$$\begin{array}{|l} \mathbf{not} : \mathit{Policy} \rightarrow \mathit{Policy} \\ \hline \forall \mathcal{FW}_1;\ P : \mathit{Policy} \bullet \\ \quad \mathbf{not}\ P = (\mathsf{Allow}^{+}\ (\mathit{deny}\ (P))) \sqcup (\mathsf{Deny}\ (\mathit{allow}\ (P))) \end{array}$$

From this definition it follows that $(\mathbf{not}\ P)$ is simply $(\mathit{deny}\,(P), \mathit{allow}\,(P))$ and thus $\mathbf{not}\,(\mathsf{Deny}\ D) = (\mathsf{Allow}\ D)$ and $\mathbf{not}\,(\mathsf{Allow}\ A) = (\mathsf{Deny}\ A)$. Note however, that in general policy negation does not define a complement operator in the algebra $\mathcal{FW}_1$, that is, it not necessarily the case that $(P \sqcup \mathbf{not}\ P) = \top$ and $(P \sqcap \mathbf{not}\ P) = \bot$.

5.1 Anomaly Analysis

A firewall policy is conventionally constructed as a sequence of order-dependent rules, and when a network packet matches with two or more policy rules, the policy is anomalous [3,4,8]. By definition, the adjacency-free allow and deny sets of some $P \in \mathit{Policy}$ are disjoint, therefore P is anomaly-free by construction. We can however define anomalies using the algebra; by considering how a policy changes when composed with other policies.

Redundancy A policy P is redundant given policy Q if their composition results in no difference between the resulting policy and Q, in particular, if $P \mathbin{⨾} Q = Q$.

Shadowing Some part of policy Q is shadowed by the entire policy P in the composition $P \mathbin{⨾} Q$ if the filter condition constraints that are specified by P contradict the constraints that are specified by Q, in particular, if $(\mathbf{not}\ P) \mathbin{⨾} Q = Q$. This is a very general definition for shadowing. Perhaps a more familiar interpretation of this definition is one where the policy P is a specific allow/deny rule that shadows a part or all of the policy with which it is composed. Recall that $(\mathbf{not}(\mathsf{Allow}\, A)) = (\mathsf{Deny}\, A)$ and, for example, in $(\mathsf{Allow}\, A) \mathbin{⨾} Q$ all or part of policy Q is shadowed by the rule/primitive policy $(\mathsf{Allow}\, A)$ if Q denies the filter conditions specified in A, that is, $(\mathsf{Deny}\, A) \mathbin{⨾} Q = Q$. Similarly, in $(\mathsf{Deny}\, D) \mathbin{⨾} Q$ part or all of policy Q is shadowed by the rule/primitive policy $(\mathsf{Deny}\, D)$ if $(\mathbf{not}\, (\mathsf{Deny}\, D)) \mathbin{⨾} Q = Q$. Further definitions for shadowing may be constructed using the algebra. For example, an initial interpretation of the generalisation anomaly [3] in the composition $P \mathbin{⨾} Q$; is where Q is generalised by P if all of P shadows (specifically) part of Q. We are currently investigating how this and other anomalies can be reasoned about within the algebra.

Inter-policy Anomalies Anomalies can also occur between the different policies of distributed firewall configurations [4]. In the following, assume that P is a policy on an *upstream* firewall and Q is a policy on a *downstream* firewall.

An inter-redundancy anomaly exists between policies P and Q if some part of Q is redundant to some part of P, whereby the target action of the redundant filter conditions is *deny*. Given some set of filter conditions A denied by P, and some set of filter conditions B denied by Q, if $(\mathsf{Deny}\, A) \mathbin{⨾} (\mathsf{Deny}\, B) = (\mathsf{Deny}\, A)$ then there exists an inter-redundancy between P and Q.

An inter-shadowing anomaly exists between policies P and Q if some part of Q's allows are shadowed by some part of P's denies. Given some set of filter conditions A denied by P, and some set of filter conditions B allowed by Q, if $(\mathsf{Deny}\, A) \mathbin{⨾} (\mathsf{Allow}\, B) = (\mathsf{Deny}\, A)$, then there is an inter-shadowing anomaly between P and Q.

An inter-spuriousness anomaly exists between policies P and Q if some part of Q's denies are shadowed by some part of P's allows. Again, given some set of filter conditions A allowed by P, and some set of filter conditions B denied by Q, if $(\mathsf{Allow}\, A) \mathbin{⨾} (\mathsf{Deny}\, B) = (\mathsf{Allow}\, A)$, then there exists an inter-spuriousness anomaly between P and Q.

5.2 Standards Compliance

RFC 5735 [7], details fifteen IPv4 address blocks/ranges that have been assigned by the Internet Assigned Numbers Authority (IANA) for specialized/global purposes. Some of these address spaces may appear on the Internet, and may be used legitimately outside a single administrative domain, however, while the assigned values of the ranges do not directly raise security issues; unexpected use may indicate an attack [7]. For example, packets with a source IP address from the private address space 172.16.0.0/12, arriving on the Wide Area Network interface of a network router, may be considered spoofed, and may be part of a Denial of Service (DoS), or Distributed DoS attack.

RFC 5735 Compliance An IP spoof-mitigation compliance policy $\mathsf{RFC5735}$ is defined. Best practice recommendations are implemented for each of the fifteen specialized IP ranges in [7], resulting in one hundred and twenty iptables *deny* rules. In [10], we defined this *deny* ruleset for a firewall management tool, we do not give the definition here for reasons of space. The compliance policy terminates with a final iptables rule that specifies all other traffic be permitted. To model these iptables rules in the algebra, we define some additional filter condition attributes and provide a more formal definition of $\mathsf{RFC5735}$.

An Extended Firewall Policy An attribute for the iptables filter table chains may be defined as $Chain ::= \mathsf{input} \mid \mathsf{output} \mid \mathsf{forward}$. Direction-based filtering may be given as $Dir ::= \mathsf{ingress} \mid \mathsf{egress}$, and the set of all sets of interfaces on a machine may be given as $\mathbb{P}\, Iface$, where for simplicity, we assume elements of $Iface$ resemble $\mathsf{eth0}$, $\mathsf{wlan0}$, $\mathsf{tun0}$, etc. Let $AdditionalFC$ be the set of all duplets for additional filter condition attributes of interest for this paper, whereby:

$$AdditionalFC == \delta[\mathbb{P}\, Chain, \delta[\mathbb{P}\, Dir, \mathbb{P}\, Iface]]$$

A revised definition for the set of all filter conditions $FC_{\mathcal{I}}$ is given as:

$$FC_{\mathcal{I}} == \delta[\alpha[IPv4], \delta[\alpha[Port], \delta[\alpha[IPv4], \delta[\alpha[Port], \delta[Protocol, AdditionalFC]]]]]$$

A revised definition for the set of all policies $Policy_{\mathcal{I}}$ is given as:

$$Policy_{\mathcal{I}} == \{A, D : \alpha[FC_{\mathcal{I}}] \mid \forall\, a : A;\ d : D \bullet a \mid_{FC_{\mathcal{I}}} d\}$$

The compliance policy $\mathsf{RFC5735} \in Policy_{\mathcal{I}}$, defines the minimum requirement for what it means for some perimeter network firewall policy to mitigate the threat of IP spoofing for all traffic, in accordance with RFC 5735. Thus, we have for all $P \in Policy_{\mathcal{I}}$, if $P \sqsubseteq \mathsf{RFC5735}$, then P complies with the best practice recommendations outlined in [7] for IP spoof-mitigation.

6 Encoding and Evaluating iptables Policies

A prototype policy management toolkit has been implemented in Python for iptables. We reason over $Policy_{\mathcal{I}}$ policies using $(\fatsemi, \sqcup, \sqsubseteq)$; time-based performance-analysis tests were conducted. The test-bed for the experiments was a 64-Bit

Ubuntu 14.04 LTS OS, running on a Dell Latitude E6430, with a quad-core Intel i5-3320M processor and 4GB of RAM. Every experiment was conducted three times; the median result chosen for inclusion in this paper. Overall, the results are promising.

Evaluating Sequential Policy Composition Two datasets were generated for experimentation. Each dataset consists of iptables policies of size $2^4 \mathinner{..} 2^{11}$. One dataset contains policies where no rule is adjacent to any other rule (other than itself), and the other dataset consists of policies where every new rule is adjacent to the previous rule; to ensure the maximum number of possible rules are generated as a result of composition. The rules all have a target action of *allow*. The implementation parses the system's currently enforced iptables ruleset $\langle r_1, r_2 \mathinner{..} r_n\rangle$ by chain, and then normalizes each rule to a primitive/singleton policy $\langle \mathcal{I}(r_1), \mathcal{I}(r_2) \mathinner{..} \mathcal{I}(r_n)\rangle$. The overall policy for the chain is evaluated as $\mathcal{I}(r_1) \fatsemi \mathcal{I}(r_2) \fatsemi \mathinner{..} \fatsemi \mathcal{I}(r_n)$. For reasons of space, we give the results for the sequential composition experiments as lists of tuples ($\mathcal{P}$,$\mathcal{T}(\mathcal{P})$), where $\mathcal{P}$ is the policy named by the number of iptables rules it was constructed from for the experiment, and $\mathcal{T}(\mathcal{P})$ is the time taken in seconds for the evaluation of the sequential composition of the rules in $\mathcal{P}$. For the adjacent dataset we have [(2^4, 0.80), (2^5, 2.02), (2^6, 5.13), (2^7, 15.32), (2^8, 51.18), (2^9, 183.42), (2^{10}, 707.15), (2^{11}, 2792.81)]. We observe that the evaluation time for the sequential composition of 2^9 rules is around three minutes, and $\mathcal{T}(2^{11})$ is approximately forty six minutes. For the non-adjacent dataset, we have [(2^4, 0.07), (2^5, 0.13), (2^6, 0.29), (2^7, 0.67), (2^8, 1.73), (2^9, 4.98), (2^{10}, 16.09), (2^{11}, 57.81)], and we see that for the largest ruleset, 2^{11}, $\mathcal{T}(2^{11})$ is approximately one minute.

Evaluating Policy Union Experiments were conducted to test policy lub, whereby each policy in the adjacent dataset was split into two policies, where the first policy contains the odd (index) rules from the original policy, and the second policy contains the even (index) rules from the original policy. Then for each P, $Q \in \mathit{Policy}_{\mathcal{I}}$ in this split dataset, the time taken for the operation $P \sqcup Q$ is encoded in the matrix in Table 1. The times taken for composition of policies of equal size are approximately the same as (slightly less than) those for the results given in the adjacent sequential composition dataset. This is highlighted through the diagonal in the matrix, and is as expected; given that we used all *allow* rules.

Evaluating Policy Compliance A further dataset consisting of iptables policies of size $2^4 \mathinner{..} 2^{11}$ was generated to test policy compliance. Each policy in this dataset was RFC 5735 compliant by construction. Results are again given as a list of tuples ($\mathcal{P}$,$\mathcal{T}(\mathcal{P})$), where $\mathcal{P}$ is the policy named by the number of iptables rules it was constructed from for the experiment, and $\mathcal{T}(\mathcal{P})$ is the time taken in seconds for the evaluation of $\mathcal{P} \sqsubseteq \mathsf{RFC5735}$. We have [($2^4$, 1.07 $\times 10^{-3}$), (2^5, 1.62 $\times 10^{-3}$), (2^6, 2.23 $\times 10^{-3}$), (2^7, 3.50 $\times 10^{-3}$), (2^8, 5.24 $\times 10^{-3}$), (2^9, 1.03 $\times 10^{-2}$), (2^{10}, 2.76 $\times 10^{-2}$), (2^{11}, 4.95 $\times 10^{-2}$)], and we see that for each $\mathcal{P} \in \mathit{Policy}_{\mathcal{I}}$ in this compliance-dataset $\mathcal{T}(\mathcal{P})$ is negligible.

Table 1. Time taken to compute $P \sqcup Q$ (in seconds)

$P \backslash Q$	2^3	2^4	2^5	2^6	2^7	2^8	2^9	2^{10}
2^3	$\boxed{0.65}$	0.79	0.81	0.99	1.40	2.51	5.73	16.93
2^4	0.79	$\boxed{1.86}$	2.09	2.32	2.91	4.50	8.83	22.19
2^5	0.81	2.09	$\boxed{4.97}$	5.45	6.78	9.17	15.50	32.89
2^6	0.99	2.32	5.45	$\boxed{14.70}$	17.01	21.93	32.29	57.47
2^7	1.40	2.91	6.78	17.01	$\boxed{48.85}$	58.44	76.94	119.28
2^8	2.51	4.50	9.17	21.93	58.44	$\boxed{179.87}$	217.34	294.56
2^9	5.73	8.83	15.50	32.29	76.94	217.34	$\boxed{699.11}$	839.49
2^{10}	16.93	22.19	32.89	57.47	119.28	294.56	839.49	$\boxed{2722.63}$

7 Related Work

In [3], a firewall policy is modelled as a single rooted tree, relations between rules are defined on a pairwise basis, and definitions for firewall configuration anomalies are provided. In [4], the work is extended to distributed firewall policies. In [8], a firewall policy is modelled as a linked-list, and in [13] rule relations within a policy are modelled in a directed graph. In [20] Binary Decision Diagrams are used to model firewall rulesets. We model a firewall policy as an ordered pair of disjoint adjacency-free sets, where the set of policies *Policy* forms a lattice under $\sqsubseteq$, and each $P \in \mathit{Policy}$ is anomaly-free by construction. In [3, 4, 8, 13, 20] an algorithmic approach is taken to detect/resolve anomalies. We follow an algebraic (as opposed to algorithmic) approach towards modelling anomalies in a single policy, and across a distributed policy configuration through policy composition. In earlier work [12], we developed the algebra $\mathcal{FW}_0$, and used it to reason over host-based and network access controls in OpenStack. In the $\mathcal{FW}_0$ algebra, we focused on stateless firewall policies that are defined in terms of constraints on individual IPs, ports and protocols. In this paper, the algebra $\mathcal{FW}_1$ is defined over stateful firewall policies constructed in terms of constraints on source/destination IP/port ranges, the TCP, UDP and ICMP protocols, and additional filter condition attributes. We argue that $\mathcal{FW}_1$ gives a more expressive means for reasoning over OpenStack security group and perimeter firewall configurations. In [16], *cloud calculus* is used to capture the topology of cloud computing systems and the global firewall policy for a given configuration. This paper could extend the work in [16], given that $\mathcal{FW}_1$ may be used in conjunction with cloud calculus to guarantee anomaly-free dynamic firewall policy reconfiguration, where the ordering relation $\sqsubseteq$ may give a viable alternative for the given equivalence relation defined over 'cloud' terms for the formal verification of firewall policy preservation after a live migration. In [21], a firewall policy algebra is proposed. However, the authors note that an anomaly-free composition is not guaranteed as a result of using their algebraic operators. Our work differs, in that policy composition under the $\sqcap, \sqcup$ and $\fatsemi$ operators defined in this paper all result in anomaly-free policies. In [2], an abstract model for Netfilter is proposed, and a language to specify firewall configurations is introduced that is similar to the XML-based access control language supported by Or-BAC presented in [9]. In [17], a formal model of Netfilter is defined, and the properties of reachability and cyclicity within firewall policy configurations are investigated. In [6], a theorem-proving approach is used to reason about firewall policies. The proposed algebra $\mathcal{FW}_1$ is used to reason about and compose anomaly-free policies and therefore we do not have to worry about dealing with conflicts that may arise. Anomaly conflicts are dealt with in composition by computing anomaly-free policies, rather than using techniques such as [15] to resolve conflicts in policy decisions. Encoding a definition for Network Address Translation in $\mathcal{FW}_1$ is a topic for future research.

8 Conclusion

A policy algebra $\mathcal{FW}_1$ is defined in which firewall policies can be specified and reasoned about. At the heart of this algebra is the notion of safe replacement, that is, whether it is secure to replace one firewall policy by another. The set of policies form a lattice under safe replacement and this enables consistent operators for safe composition to be defined. Policies in this lattice are anomaly-free by construction, and thus, composition under glb and lub operators preserves anomaly-freedom. A policy sequential composition operator is also proposed that can be used to interpret firewall policies defined more conventionally as sequences of rules. The algebra can be used to characterize anomalies, such as shadowing and redundancy, that arise from sequential composition. Best practice policy compliance may be defined using $\sqsubseteq$. The algebra $\mathcal{FW}_1$ provides a formal interpretation of the network access controls for a partial mapping to the iptables filter table. $\mathcal{FW}_1$ is a generic algebra and can also be used to model other firewall systems. The results in this paper are described in terms of the algebra $\mathcal{FW}_1$, for stateful firewall policies that are defined in terms of constraints on source/destination IP/port ranges, the TCP, UDP and ICMP protocols, and additional filter condition attributes.

Acknowledgement This work was supported, in part, by Science Foundation Ireland under grant SFI 10/CE/I1853 and Irish Research Council/Chist-ERA.

References

1. Linux iptables - CLI for configuring the Linux kernel firewall, Netfilter. *http://www.netfilter.org/projects/iptables/index.html.*
2. P. Adão, C. Bozzato, G. Dei Rossi, R. Focardi, and F. L. Luccio. Mignis: A semantic based tool for firewall configuration. In *Proceedings of the 2014 IEEE 27th Computer Security Foundations Symposium*, pages 351–365. IEEE, 2014.
3. E. Al-Shaer and H. Hamed. Firewall policy advisor for anomaly discovery and rule editing. In *Integrated Network Management*, volume 246 of *IFIP Conference Proceedings*, pages 17–30. Kluwer, 2003.

4. E. Al-Shaer, H. Hamed, R. Boutaba, and M. Hasan. Conflict classification and analysis of distributed firewall policies. *IEEE Journal on Selected Areas in Communications*, 23(10):2069–2084, 2005.
5. G. Birkhoff. *Lattice Theory. Volume XXV of American Mathemical Society Colloquium Publications.* American Mathemical Society, 3rd edition, 1967.
6. A. D. Brucker, L. Brügger, and B. Wolff. Formal firewall conformance testing: An application of test and proof techniques. *Softw. Test. Verif. Reliab.*, 25(1):34–71, January 2015.
7. M. Cotton and L. Vegoda. Special Use IPv4 Addresses. RFC 5735, January 2010.
8. F. Cuppens, N. Cuppens-Boulahia, and J. García-Alfaro. Detection and removal of firewall misconfiguration. In *Proceedings of the 2005 IASTED International Conference on Communication, Network and Information Security*, volume 1, pages 154–162, 2005.
9. F. Cuppens, N. Cuppens-Boulahia, T. Sans, and A. Miège. A formal approach to specify and deploy a network security policy. In *Formal Aspects in Security and Trust*, pages 203–218. Springer US, 2005.
10. W. M. Fitzgerald, U. Neville, and S. N. Foley. MASON: Mobile autonomic security for network access controls. *Journal of Information Security and Applications, (JISA)*, 18(1):14–29, 2013.
11. S. N. Foley. The specification and implementation of commercial security requirements including dynamic segregation of duties. In *ACM Conference on Computer and Communications Security*, pages 125–134, 1997.
12. S. N. Foley and U. Neville. A firewall algebra for openstack. In *2015 IEEE Conference on Communications and Network Security, CNS 2015, Florence, Italy, September 28-30, 2015*, pages 541–549, 2015.
13. A. Hari, S. Suri, and G. Parulkar. Detecting and resolving packet filter conflicts. In *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 3, pages 1203–1212. IEEE, 2000.
14. J. L. Jacob. The varieties of refinement. In J. M. Morris and R. C. Shaw, editors, *Proceedings of the 4th Refinement Workshop*, pages 441–455. Springer-Verlag, 1991.
15. S. Jajodia, P. Samarati, M. L. Sapino, and V. S. Subrahmanian. Flexible support for multiple access control policies. *ACM Trans. Database Syst.*, 26(2):214–260, 2001.
16. Y. Jarraya, A. Eghtesadi, M. Debbabi, Y. Zhang, and M. Pourzandi. Cloud calculus: Security verification in elastic cloud computing platform. In *Collaboration Technologies and Systems (CTS), 2012 International Conference on*, pages 447–454. IEEE, 2012.
17. A. Jeffrey and T. Samak. Model checking firewall policy configurations. In *Policies for Distributed Systems and Networks, 2009. POLICY 2009. IEEE International Symposium on*, pages 60–67. IEEE, 2009.
18. L. Levine. (Lemma 3, Example 6.) Algebraic Combinatorics, Lecture 8, Retrieved: http://www.math.cornell.edu/~levine/18.312/alg-comb-lecture-8.pdf, March 2011.
19. J. M. Spivey. *The Z Notation: A Reference Manual.* Series in Computer Science. Prentice Hall International, 2nd edition, 1992.
20. L. Yuan, H. Chen, J. Mai, C. Chuah, Z. Su, and P. Mohapatra. Fireman: A toolkit for firewall modeling and analysis. In *Security and Privacy, 2006 IEEE Symposium on*, pages 15 pp.– 213. IEEE, 2006.
21. H. Zhao and S. M. Bellovin. Policy algebras for hybrid firewalls. Technical Report CUCS-017-07, Department of Computer Science, Columbia University, March 2007.