Malachite: Firewall Policy Comparison

Dinesha Ranathunga*, Matthew Roughan*, Phil Kernick† and Nick Falkner‡
*ACEMS, University of Adelaide, Australia
†CQR Consulting, Australia
‡School of Computer Science, University of Adelaide, Australia
Email: {dinesha.ranathunga, matthew.roughan, nick.falkner}@adelaide.edu.au, phil.kernick@cqr.com

***Abstract*—Firewalls are a crucial element of any modern day business; they protect data and resources in a communications network from unauthorised access. In particular domains, such as SCADA networks, there are guidelines for firewall configuration, but currently there are no automated means to test compliance. Our research tackles this from first principles: we ask how firewall policies can be described at a high-level, independent of firewall-vendor and network minutiae. The semantic foundations we propose allow us to compare network-wide firewall policies and check if they are equivalent; or one is contained in the other in meaningful ways. These foundations also enable policy change-impact analysis and help identify functional discrepancies between multiple policy designs from users in distinct policy sub-domains (*e.g.,* SCADA engineers, Corporate admins).**

***Index Terms*—SCADA network security, Zone-Conduit model, firewall policy, best-practice compliance, change-impact analysis**

I. Introduction

Firewalls are a standard mechanism for enforcing network security. They protect data and resources in a communications network from unauthorised access by filtering out unwanted network traffic. These filtering decisions are based on a set of ordered filtering rules defined to meet the security policy requirements of an organisation. Security administrators may naturally wish to compare the semantics of two firewall access-control policies (*i.e.,* firewall policies), for instance, to:

- compare a policy to industry recommended practices;
- study change impact, *i.e.,* how policies evolve over time;
- identify discrepancies between multiple policy designs;
- check an implemented policy is the intended policy; or
- change the underlying firewalls (*e.g.,* vendor), but ensure the policy remains the same.

The ability to compare firewall policies meaningfully is particularly important in Supervisory Control and Data Acquisition (SCADA) networks, which control the distributed assets of many critical systems such as power generation and water distribution. These networks often incorporate highly vulnerable devices such as Programmable Logic Controllers (PLCs) that control physical devices such as gas valves. As a result, in this domain there are industry-recommended standards [3], [8] to which we would like to make comparisons.

Checking firewall policies for compliance with recommended practices has challenges:

- there is a lack of common standards or languages for firewall policy specification;
- equivalent policies can be specified in many ways (*e.g.,* either through accept or deny rules);
- extraneous network details impinge on current means of specifying policies; and
- comparisons between two policies could be computationally expensive.

Our tool – *Malachite* – provides the means to automatically perform such comparisons algebraically. In building it, we have derived many of the properties and constraints that a formal description of firewall policies must satisfy.

First, firewall policies are described at a high-level, independent of vendor and network intricacies to provide a common ground for comparisons. We propose a policy algebra to efficiently compare these high-level firewall policies. The two most significant advantages of such a framework are: (i) its algebraic structure helps determine the semantics required of a policy language to make comparisons rule-order and firewall-implementation independent; and (ii) it provides a formalism to compute the composition of rule sets.

Secondly, canonical forms of policies are derived, for efficient comparisons, and we develop an implementation that can perform checks on real firewall configurations.

We demonstrate the use of our proposed algebras by comparing policies deployed in real networks. Particularly, we show its value in protecting SCADA networks from cyber attacks. Additionally, our algebras promote easy *change-impact analysis* of network-wide firewall policies by administrators.

II. Background and Related Work

Firewalls are a front-line defence mechanism against network intrusion and attacks, and guidelines have been prepared to help with their deployment [3], [8]. These suggest several high-level policy abstractions. In particular, we refer to the *Zone-Conduit abstraction* as a way of segmenting and isolating the various sub-systems [3].

A *zone* is a logical or physical grouping of an organisation's systems with similar security requirements so that a *single policy* can be defined for all zone members. A *conduit* provides the secure communication path between two zones, enforcing the policy between them [3]. A conduit could consist of multiple links and firewalls but, logically, is a single connector.

The Zone-Conduit model intuitively decouples policy from topology, relieving policies of firewall-vendor and network intricacies. It allows user to specify policies at a high-level; a feature most desirable in auto-configuration; but importantly, it provides a common ground for comparing policies.

The ISA Zone-Conduit model in its original specification is too flexible for policy specification. We use the extensions proposed in [19] to increase its precision, *e.g.,* it is necessary to enforce a 1:1 mapping between policies and conduits. This property allows us to specify a conduit-policy unambiguously using two *inter-zone flow policies*. For instance, the policy p_0 of a conduit that interconnects zones Z_1 and Z_2 can be defined using flow-policies p_A and p_B, *e.g.,*

$$\begin{aligned} Policy\ p_0\ \{\ p_A &:: \text{ Z1} \rightarrow \text{Z2: https, dns;} \\ p_B &:: \text{ Z2} \rightarrow \text{Z1: https; } \} \end{aligned}$$

We can derive the *zone-flow model*, which is a directed graph, by replacing each undirected-conduit with a *directed-conduit pair* in this manner. The policy of an entire network can then be defined as a collection of directed conduits.

Firewall analysis is a well studied problem, various debugging tools [16], [20], [21] have been developed to check firewall configuration behaviour. Tools have also been proposed [13], [22] to detect firewall policy violations and ensure intended policy is correctly implemented on distributed firewalls. These tools test a given set of predefined firewall rules, but do not compare policies and standards.

Related works use a *bottom-up* comparison to analyse single-firewall rulesets (*i.e.,* ACLs) and evaluate change impact [15]. Our interest is to compare firewall policies *not their implementations*, and ACLs contain network and vendor intricacies like IP addresses. So, ACL differences can occur even when the *policy's intent* remains unchanged.

Network policy comparisons require firewall topology to be accounted for. In some works [15], topology needs to be accounted for manually; an impractical consideration when comparing policies of complex networks. Some (bottom-up) approaches overcome the problem by automatically incorporating topology to construct a network-wide ACL tree [1], [22]. Their methods help detect implementation errors (*i.e.,* redundancies and conflicts), but lack support for policy comparisons.

A better, less tedious approach is a *top-down* comparison. To do so, a *single network-wide policy* needs to be maintained, so, security administrators can easily determine who gets in and who doesn't [12]. These network-wide policies need to be free of network-centric minutiae, so changes to the policy's intent can be clearly distinguished from changes to the network. Network-wide firewall policy changes are also more useful to administrators than *per-firewall changes* when analysing change impact, because the latter can be made redundant by other firewalls' policy through rule interactions. Current top-down solutions allow creation of network-wide policies [2], [6], [10] but lack ability to compute their semantic differences.

Our top-down comparisons use Zone-Conduit based firewall policies, to generate network-wide policy changes. These policies are decoupled from the network, so the reduced policy complexity allows them to be easily understood by humans. We make the policies firewall-implementation independent and comparable, through the formalisms discussed in §III and §IV.

We couldn't find any related works that validated firewall policies against industry-recommended practices. Doing so, is *critical* in SCADA networks where more restrictive practices are required to minimise their vulnerability to threats from less secure networks (*e.g.,* the Internet) [3], [8]. *Malachite* supports validation of firewall policies against recommended practices and provides users of assurance of compliance prior to deploying policies to the physical firewalls.

III. Policy on a single directed conduit

We now outline policy comparison on a *single directed conduit* of a network. We start by considering policies in general, and then restrict our attention to a computationally tractable (and typical) set of policies.

A firewall policy is constructed from an ordered series of rules $[p_1, p_2, \ldots, p_n]$ that act on packet sequences to *accept*, *deny*, or *modify* them. We formally define these rules starting with $S = \{all\ packet\ sequences\}$, which implicitly includes ϕ (no packets). In principle, a firewall could delay its decision on a packet sequence until it can decide if the whole sequence is valid, but practical limitations (on stored state, and packet delays) mean that decisions must be made on short sequences, and often on *single packets*. So, we define rules to operate on $S^* = \{complete\ packet\ sequences\}$, where we define *complete* in a moment.

We define the operation $s = s_1 + s_2$, to mean that s is the concatenation of the two subsequences. Both S and S^* are *closed* under + (an associative operator) and ϕ is the identity. Hence S and S^* are *monoids*.

A policy rule p is a function $p : S^* \rightarrow S^*$. The formal meaning of *complete* with respect to a family $\mathcal{F}$ of possible rules is that $\forall s_1, s_2 \in S^*$ and policy rules $p \in \mathcal{F}$

$$p(s_1 + s_2) = p(s_1) + p(s_2). \tag{1}$$

So, *complete* means a decision on two concatenated complete subsequences would be the same as that on the joint sequence.

Hence, valid policy rules $p(\cdot)$ are *monoid endomorphisms* on S^*, *i.e.,* they are mappings from S^* to itself that preserve the semigroup structure of the operator + and the identity ϕ.

A *packet sequence* also includes packet timings. For instance, imagine a packet u that is fragmented into two component packets u_a and u_b. A firewall might perform fragment reassembly in order to test the packet's validity. So it might allow $u = u_a + u_b$, but not allow either fragment by itself. This appears to break (1). However, if the firewall sees u_a "by itself" what it has really done is observe $u_a + timeout$, where the timeout is used to decide when to give up on the second fragment. And $u_a + timeout + u_b \neq u_a + u_b$, hence timing information is a crucial component of the packet sequences.

We will often define policy rules on the set $\mathcal{A}$ of *atomic* packet sequences, *i.e.,* complete packet sequences that cannot be decomposed into smaller complete subsequences (except themselves and ϕ). Typical policy rules accept or deny packets, *i.e.,* for some $A \subset \mathcal{A}$ (which can be extended to S^*)

$$p_A(s) = \begin{cases} s, & \text{if } s \in A,\ //\ accept \\ \phi, & \text{if } s \in A^c, //\ deny. \end{cases} \tag{2}$$

This type of rule doesn't allow modification or creation of packets. Real firewall rules can modify or create packets. For instance a firewall might

- update certain header fields related to QoS; or
- might defragment, or fragment packets; or
- be integrated with Network Address Translation (NAT) or Virtual Private Network (VPN) functionality.

The scope for packet modification is huge, but within a firewall, many modifications don't change fields that would affect further rules in subsequent firewalls, *e.g.,* QoS or TTL changes. Hence, we restrict rules to such modifications, and thus consider them to be in the form given in (2).

Firewall rules can also generate packets through event logging. We exclude logging from the scope of this paper – see [18] for an extended discussion of this and related issues.

We also cannot construct a policy rule for any arbitrary subset of $\mathcal{A}$, due to the limitations of technology used in a firewall. The subsets of $\mathcal{A}$ for which rules can be defined is actually a *sigma algebra* $\sigma(\mathcal{A})$ [7].

This sigma algebra is generated by the finest possible partition of $\mathcal{A}$ determined by the firewall technology. For a given firewall, $\mathcal{A}$ can be broken into sets $A_i \subset \mathcal{A}$ such that $A_i \cap A_j = \phi$ and $\cup_i A_i = \mathcal{A}$, and we can implement rules p_{A_i}, but cannot define any rule p_B s.t. B is a strict subset of A_i.

A. Positive, explicit policies

A firewall policy, in practice, is made up of multiple predicate-matching rules that can be combined using several strategies: *first match*, *last match* or *all match*. If we presume here the conservative security option: an implicit `deny-all` rule, then an accept rule q^a_m defines accept packet set $A = \{s \in \mathcal{A} \mid s \prec m\}$, where $s \prec m$ denotes *s matches predicate* m (likewise a deny packet set for deny rule q^d_m).

When we combine two (ordered) rules $(q^{a/d}_{m_1}, q^{a/d}_{m_2})$ we define operators depending on the matching order: *e.g.,*

1) first match

$$q^{a/d}_{m_1} \oslash q^{a/d}_{m_2} = \begin{cases} q^{a/d}_{m_1}, & \text{if } s \prec m_1 \\ q^{a/d}_{m_2}, & \text{if } s \prec m_2 \text{ and } s \not\prec m_1 \\ deny, & \text{otherwise,} \end{cases}$$

2) last match

$$q^{a/d}_{m_1} \ominus q^{a/d}_{m_2} = \begin{cases} q^{a/d}_{m_2}, & \text{if } s \prec m_2 \\ q^{a/d}_{m_1}, & \text{if } s \prec m_1 \text{ and } s \not\prec m_2 \\ deny, & \text{otherwise.} \end{cases}$$

The operations $\oslash$ and $\ominus$ are associative, *e.g.,* $(q_1 \oslash q_2) \oslash q_3 = q_1 \oslash (q_2 \oslash q_3)$, but *not commutative or equivalent*. This is a problem. Our interest is whether a *policy* is equivalent, *not an implementation*, else we cannot easily check, for instance, if the policy remains static when firewall-vendor changes. So, we restrict ourselves to `accept rules`, conditional on an implicit `deny-all` rule. This restriction (see theorems below)

1) is rich enough to represent all rules of the form (2); and
2) if we restrict to this family of rules, then the operators above are all commutative and equivalent.

Similar results also hold if we only allow deny rules, but this option is less secure as it is easier to accidentally leave something out of a deny list than to explicitly include it.

Theorem 1 (Commutativity)**.** *The order of matching is irrelevant iff the match rules are all* `accept rules` *plus an implicit* `deny-all` *rule (or visa versa).*

Proof. First note that if we allow only accept rules, with an implicit `deny-all`, then the operators above become

$$q^a_{m_1} \oslash q^a_{m_2} = q^a_{m_1} \ominus q^a_{m_2} = p_A,$$

where $A_1, A_2 \in \sigma(\mathcal{A})$ such that $A_i = \{s \in \mathcal{A} \mid s \prec m_i\}$, and $A = A_1 \cup A_2$. Hence when we limit ourselves to accept rules the operators are commutative (via commutativity of set intersection) and equivalent.

If we have one accept, and one deny rule, with an implicit `deny-all`, then for a non-trivial rule space we can construct an example that violates commutativity and equivalence. Take two sets $A_1, A_2 \in \sigma(\mathcal{A})$ with matching predicates m_1 and m_2 such that $A_1 \neq A_2 \neq \phi$. Then under

$$\begin{aligned} q^a_{m_1} \oslash q^d_{m_2} &= p_{A_1}, \\ q^d_{m_2} \oslash q^a_{m_1} &= p_{A_1 \setminus A_2}, \\ q^a_{m_1} \ominus q^d_{m_2} &= p_{A_1 \setminus A_2}, \\ q^d_{m_2} \ominus q^a_{m_1} &= p_{A_1}. \end{aligned}$$

Combinations yield different composite rules, implying a non-commutative and non-equivalent counter-example. ☐

Theorem 2 (Completeness)**.** *Any rule (2) for* $A \in \sigma(\mathcal{A})$ *can be represented by a series of accept rules* $(q^a_{m_1}, q^a_{m_2}, \ldots, q^a_{m_n})$.

Proof. When we constructed $\sigma(\mathcal{A})$ from the A_i we defined these as the finest partition of $\mathcal{A}$. This means that there is at least one predicate that matches A_i within the ruleset available.

By construction all elements $A \in \sigma(\mathcal{A})$ can be constructed by finite unions on the sets A_i, and any rule combination generates a union, *i.e.,* $(q^a_{m_i}, q^a_{m_j})$ defines set $A_i \cup A_j \subset \mathcal{A}$, so we can define any set A by listing rules for each member. ☐

The key advantage of restricting attention to accept rules, is that *rule order doesn't matter*, and *neither does implementation strategy*. A policy holds the same semantics, irrespective of how its rules are organised, removing dependencies from the policy specification. So, we can add or remove a rule without considering the complete rule set and the potential interactions. By being explicit, we also guard against services being accidentally enabled implicitly or by default.

We now simplify the notation we use in this context to:

1) $(p_A \oplus p_B)(s) = p_{A \cup B}(s)$ denotes within-firewall composition, or construction through parallel firewalls;
2) $(p_A \otimes p_B)(s) = p_B(p_A(s)) = p_{A \cap B}(s)$ denotes the action of sequential firewalls.

Under the restrictions above, these are associative, commutative binary operators, with identities ϕ and $\mathcal{A}$ respectively, and the identity of the $\oplus$ operator is the annihilator of the $\otimes$ operator. Thus, they define a *commutative semiring* across the policy space $\Phi = \{p : \mathcal{A} \to \mathcal{A} \mid p \text{ is a monoid endomorphism}\}$.

B. Conduit policy comparison

A policy can be a single rule or as a set of rules and operators, *i.e.,* we can write a policy as a single mapping function $p^X \in \Phi$ or as its components $p^X = (p_1^X \oplus \cdots \oplus p_m^X)$. There are many ways to combine rules to achieve a given affect, so we define two policies to be *equivalent* as follows.

Definition 3 (Equivalence)**.** *Two policies p^X and p^Y are* equivalent *on $\mathcal{A}$ iff $p^X(s) = p^Y(s)$, $\forall s \in \mathcal{A}$. We denote this equivalence by $p^X \equiv p^Y$.*

When we compare policies on two networks, it is also useful to be able to partition the policies on the networks in *equivalence classes* [4]. We call this a *semantic partition* (SP), which is formally defined as follows.

Definition 4 (Semantic partition)**.** *The* semantic partition *SP of a set of policies P is given by $SP = \{e_m\}$ where $P = \cup_m e_m$ and the $e_m \subset P$ are the minimal number of equivalence classes,* i.e., *for all $p_i, p_j \in e_m$ we have $p_i \equiv p_j$.*

An *equivalence class* groups a set of conduits with identical policies. Large equivalence class sizes in a network imply many zones with identical reachability. This indicates a *badly designed* Zone-Conduit model, to overcome it, the said zones may need to be amalgamated (at least at a logical level). Semantic partitions also help to write better policies, for they allow to check if a collection of conduits have a given policy.

Two policies with different rule sets, can have the same underlying semantics (*i.e.,* they allow the same set of services between zones). For instance, Figures 1(a) and 1(b) illustrate the idea based on TCP port filtering of single packets. Each rectangle indicates the allowed packets of a single rule. Combined, the rules cover the same set of allowed packets.

We could compare policies by exhaustive comparison, but that would be highly inefficient. A more efficient approach would be to derive a unique, *canonical*, representation of each policy. We can represent canonicalisation of policies through a mapping $c : \Phi \to \Theta$, where Θ is the canonical space of policies, in which all equivalent policies of Φ map to a singleton. Given a canonicalisation mapping, we note the following (the proof being the direct result of the definitions).

Lemma 5. *Policies $p^X \equiv p^Y$ iff $c(p^X) = c(p^Y)$.*

Thus, comparison is eased by the canonicalisation of policies. Figure 1(c) illustrates the idea using TCP policy rules and dissect the polygon formed in our example policy into horizontal partitions, using a Rectilinear-Polygon to Rectangle conversion algorithm [9]. Each partition is chosen to provably guarantee its uniqueness. We derive canonical policy elements by translating each partition back to a rule and ordering the resulting rule-set uniquely in increasing IP protocol number and source, destination port numbers. We find a *unique partition quickly* rather than a guaranteed minimal partition. The result is a deterministic, ordered set of non-overlapping rules.

We need to evaluate our policies against industry-recommended guidelines. When two flow-policies are *equivalent*, they essentially accept and deny the same IP packets, but it will be common that some customisation of policies must be conducted locally. A policy *complies* with another if it is more restrictive. In that context we define *inclusion* as follows.

Definition 6 (Inclusion)**.** *A policy p^X is* included *in p^Y on $\mathcal{A}$ iff $p^X(s) \in \{p^Y(s), \phi\}$,* i.e., *$X$ has the same effect as Y on s, or denies s, for all $s \in \mathcal{A}$. We denote inclusion by $p^X \subset p^Y$.*

In order to make practical use of this definition (via canonicalisation), we restrict our attention to the accept rules considered above, where the following lemma applies.

Lemma 7. *If policy $p^X \subset p^Y$ then $p^X \oplus p^Y \equiv p^Y$.*

Proof. We can describe all accept rules by functions such as (2) and $p_A \subset p_B$ means that $A \subset B$. Moreover $p_A \oplus p_B = p_{A \cup B}$, and $A \subset B$ implies $A \cup B = B$. Hence the result. $\square$

We define the *semantic difference* of two policies as follows (again restricting to `accept rules` for practical use).

Definition 8 (Semantic Difference)**.** *The semantic difference between policies p^X and p^Y is given by $p^X - p^Y = (p^X \oplus p^Y) \otimes (p^X \otimes p^Y)^c$, where $(p_A)^c = p_{A^c}$ and A^c is A's complement.*

We present $c(p^X - p^Y)$ in our results. Clearly, when two policies are *equivalent*, their semantic difference yields ϕ. If $p^X \subset p^Y$, the result will contain elements from p^Y that are not in p^X. This notation is particularly useful in change impact analysis. It helps administrators understand in detail how a policy has evolved with time. *Equivalence* and *inclusion* notations on the other hand, allow fast, high-level checks.

IV. Comparison of network policies

We considered policy rules on a *single directed conduit* in § III. Now we generalise policies to a network, or rather the simplified Zone-Conduit model of the network.

When we apply a policy to a conduit, we explicitly label the address space of allowed sources and destinations. Thus, we now consider policy rules to act as part of a $(edge, rule)$ pair, where the $rule$ excludes details of sources and destinations, which are incorporated through the Zone-Conduit model.

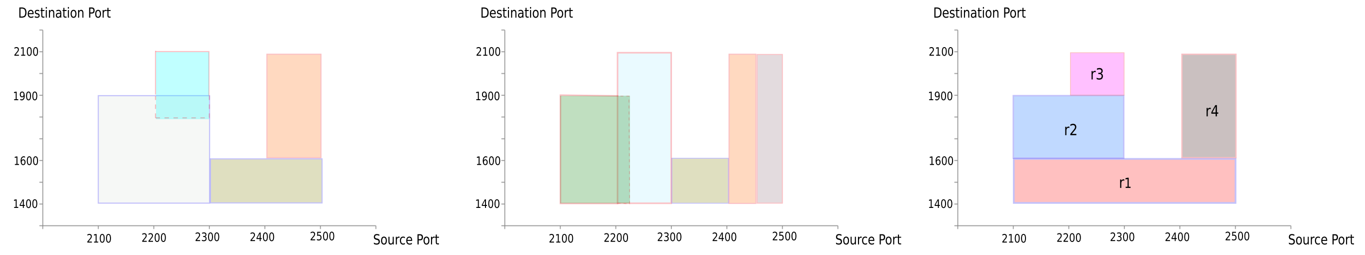
The Zone-Flow model, derived from the Zone-Conduit model (§ II), is a directed graph $G = (Z, C)$, where Z is the set of zones, and $C \subset Z \times Z$ is the set of *directed* conduits. Note that we include here indirect conduits that may be built up from multiple physical paths.

Definition 9 (Network Policy)**.** *A network policy $\mathcal{P} = (G, P)$ means a directed-graph $G(Z, C)$ with policy functions $p_{ij} \in \Phi$ for $(i, j) \in C$.*

The policy function p_{ij} here does not involve addresses, which are implicit in the directed-conduit label (i, j), and the mapping from zone to addresses.

We test if a policy complies with a larger ruleset, by extending flow-policy *equivalence*, *inclusion* and *difference* definitions to compare policies of an entire network.

Definition 10 (Equivalence)**.** *A policy $\mathcal{P}_1 = (G, P_1)$ is* equivalent *to $\mathcal{P}_2 = (G, P_2)$ iff $\forall e \in C$, $p_1^e \equiv p_2^e$. We denote this by $\mathcal{P}_1 \equiv \mathcal{P}_2$.*



(a) Four rules indicated by (overlapping) rectangles. (b) Five rules producing an equivalent policy to (a). (c) Horizontal partitions of polygon in (a) or (b).

Fig. 1: *Canonicalisation of distinct rule sets of the same policy. Rectangles indicate the packets allowed by a particular rule.*

Definition 11 (Inclusion)**.** *A policy $\mathcal{P}_2 = (G, P_2)$* includes *$\mathcal{P}_1 = (G, P_1)$ iff $\forall e \in C$, $p_1^e \subset p_2^e$. We denote this by $\mathcal{P}_1 \subset \mathcal{P}_2$.*

Definition 12 (Semantic Difference)**.** *The semantic difference between $\mathcal{P}_1 = (G, P_1)$ and $\mathcal{P}_2 = (G, P_2)$ is given by $\mathcal{P}_1 - \mathcal{P}_2 = \bigoplus\{(p_1^e - p_2^e); \forall e \in C\}$.*

Similarly, we also extend the idea of a SP to a network. An SP groups together the set of directed conduits with *equivalent policies*. These groups partition in the sense that they cover P with disjoint sets. We now extend the notion of comparison of two network policies to the comparison of their SPs.

Definition 13. *The semantic partitions SP_1 and SP_2 of policies P_1 and P_2, respectively, are* equivalent *iff $|SP_1| = |SP_2|$ and $\forall e_1 \in SP_1$, $\exists e_2 \in SP_2$ such that for any $p_1 \in e_1$ and $p_2 \in e_2$, we have $p_1 \equiv p_2$. We denote this by $SP_1 \equiv SP_2$.*

Semantic partition SP_1 includes *SP_2 iff $\forall e_2 \in SP_2$ $\exists e_1 \in SP_1$ s.t. $e_2 \subset e_1$. We denote this by $SP_2 \subset SP_1$.*

Definitions 10 to 13 assume that the underlying networks are the same. We may wish to make comparisons between different networks, or one network as it evolves over time. So we also extend the definitions to the case where the two graphs G_1 and G_2 are isomorphic, *i.e.,* $G_1 \simeq G_2$.

The essential property of our high-level language is that it has a 1:1 mapping between policies and conduits, so comparing policies of two networks is a matter of comparing policies of each conduit. The difficulty is that the zone labels are arbitrary. So, two networks might have completely different labels and policies cannot be compared directly. Ultimately, we must evaluate if the two (unlabelled) digraphs have the same graph structure and identify a mapping between the zones of the two networks. This is the *graph isomorphism* problem.

The complexity of the graph isomorphism problem is unknown, but there are no known polynomial-time algorithms [14] (the best being a recent quasi-polynomial time algorithm [5]), so there is a practical difficulty to overcome.

We also need to compare two different graphs (*i.e.,* $G_1 \not\simeq G_2$), for one, to see the effect of a network change. For another, to check if firewall policies are deployed consistently across an organisation's multiple SCADA sites. Here, the policies can't be strictly *equivalent*, but it would be confusing to talk of *inclusion*, so we introduce a new concept: *incorporation*.

Definition 14 (Incorporation)**.** *Policy $\mathcal{P}_2$* strictly incorporates *$\mathcal{P}_1$ iff G_1 is isomorphic to a subgraph of G_2 and $\forall e \in C_1$, $p_1^e \equiv p_2^e$. We denote this by $P_1 \equiv P_2(G_1)$. Policy $\mathcal{P}_2$* partially incorporates *$\mathcal{P}_1$ iff G_1 is a subgraph of G_2 and $\forall e \in C_1$, $p_1^e \subset p_2^e$. We denote this by $\mathcal{P}_1 \subset \mathcal{P}_2(G_1)$.*

A network policy P is compliant with recommended practice RP if $P \subset RP$, through either *inclusion* or *incorporation*.

To check compliance, we must now solve the *subgraph isomorphism* problem, which is NP-complete. The two isomorphism problems seem intractable, but the Zone-Conduit policy model is not a completely unlabelled digraph. We have edge labels in the form of policies.

We exploit this information to avoid the graph isomorphism problem. The mathematical tool we will work with is called the *line digraph* (directed graph), defined as follows [11]:

Definition 15 (Line Digraph)**.** *Given a digraph $D = (N, E)$, its line digraph $\mathcal{L}(D) = (P, L)$ is defined by the non-empty set of points $P = E$, and edges $(uv, vw) \in L$ wherever $(u, v) \in E$ and $(v, w) \in E$.*

The line-digraph is formed by taking one node for each edge in D, and creating edges between these new nodes wherever the original edges connected through the same node, *i.e.,* form part of a length-2 path. Finding an isomorphism in the line-digraphs allows to identify isomorphism in the original digraphs as per Theorem 16 (which generalises the Whitney graph isomorphism theorem to multi-digraphs) [11]:

Theorem 16. *Let D be a multi-digraph. If D' is a multi-digraph such that $\mathcal{L}(D) \simeq \mathcal{L}(D')$, then the digraphs formed by D and D' by deleting all* sources *and* sinks *are isomorphic.*

A multi-digraph is a digraph that allows multiple directed edges between graph nodes [11]. So, Theorem 16 applies to our model. Conduits in the Zone-Conduit model are always formed from a directed pair (to allow for different policies), so, our digraphs are *source* and *sink* free. We need not delete any nodes to apply Theorem 16: *i.e.,* if the line digraphs are isomorphic then the original policy digraphs are isomorphic.

In our case, the constructed line digraphs also have a (non-unique) node-labelling defined by the policies that labelled the original conduits (Figure 2). We can use this labelling to avoid the full complexity of the graph isomorphism problem. The resulting algorithms are described in detail in § V, but it essentially involves (i) converting to the labelled line digraph, and (ii) testing these for isomorphism or subgraph isomorphism, using the labels to reduce the potential search space.

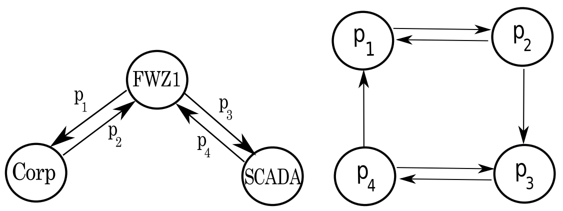


Fig. 2: *A policy digraph and its corresponding line digraph. Nodes in the line digraph are the edges of the original digraph.*

V. Implementation

Malachite is currently implemented in Python, and allows comparison of high-level firewall policies written in our own policy specification language. Throughout we use the rectilinear canonicalisation described earlier. We will make the system open source in the near future.

We now describe our implementation – *semantic partitioning method* – which groups equivalent policies. We compare it with an exhaustive method [17] used as a benchmark.

A. Check whether $\mathcal{P}_1 \equiv \mathcal{P}_2$

For policy *equivalence*, digraphs G_1 and G_2 need to be isomorphic, and we must derive all possible bijections between the two, and test for policy equivalence given the bijection. So, we derive the line digraphs $\mathcal{L}(G_1)$, $\mathcal{L}(G_2)$ and check if they can be isomorphic (*i.e.,* equal node and edge counts). We then find the possible bijections between the two graphs as follows.

We derive the semantic partitions of P_1, P_2 (*i.e.,* SP_1 and SP_2). If $SP_1 \equiv SP_2$, we find all feasible mappings between these partitions (need $O(m^2)$ comparisons where $m = |SP_1| = |SP_2|$). A feasible mapping only exists between two *equivalent classes* of SP_1 and SP_2 with equal cardinality (*e.g.,* between e_1 and e_1' in Table I). So, the search space is reduced compared to the exhaustive approach.

All Feasible Mappings (FMs) between P_1, P_2 stem from the cross product of the class-level mappings in Table I:

$$FMs = M_1 \times M_2 \times \ldots \times M_m. \tag{3}$$

Finally, we construct the adjacency matrices A_1, A_2 of $\mathcal{L}(G_1)$ and $\mathcal{L}(G_2)$ for each mapping in FMs. If there exists a mapping for which $A_1 = A_2$, then $\mathcal{P}_1 \equiv \mathcal{P}_2$.

The time complexity of the algorithm components are given in Table II. The dominant component of the algorithm in the worst case is the evaluation of FMs, with time complexity $O(\prod_{i=1}^{m} c_i!)$. In the worst case this is no better than that of the exhaustive algorithm ($O(n!)$ [17]), but in the best case where the equivalence classes are singular, then the algorithm performance is $O(n^2)$. We will describe below why, in real networks, this is the typical real performance.

B. Check whether $\mathcal{P}_1 \subset \mathcal{P}_2$

Here, we derive SP_1, SP_2 as before, but check $SP_1 \subset SP_2$ (Definition 13). If so, we determine feasible mappings between the two networks, however, the process is now complicated because we need to find *inclusive* policy mappings, not just *equivalences*. Table III illustrates the process with an example.

TABLE I: *Generate mappings between equivalent equivalence-classes of semantic-partitions SP_1 and SP_2.*

SP_1 class	equivalent SP_2 class	class size	number of mappings	mappings-set
e_1	e_1'	c_1	$c_1!$	M_1
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
e_m	e_m'	c_m	$c_m!$	M_m

TABLE II: *Time complexity of the semantic-partitioning algorithm components used to check policy equivalence.*

algorithm component	time complexity	comments
cannonicalise policy	$O(n)$	$n = \#$ of flow-policies
construct line digraph	$O(n^2)$	
derive semantic partitions	$O(n^2)$	
check partitions are equal	$O(m^2)$	$m = \#$ of partitions
evaluate mappings in FMs	$O(\prod_{i=1}^{m} c_i!)$	$c_i = \|e_i\|$

We determine the cardinality of each M_i's in the table, by taking equal number of elements from the corresponding classes and then considering all permutations. For instance,

$$|M_1| = (c_1)! \times \binom{c_1' + c_2'}{c_1}, \tag{4}$$

where we know that $c_1' + c_2' \geq c_1$ in order for the cardinality constraints to be satisfied. More generally

$$|M_i| = (c_i)! \times \binom{c_i''}{c_i}, \tag{5}$$

where $c_i'' \geq c_i$ is *total size* of all matching classes.

We determine FMs as in § V-A, construct adjacency matrices for each mapping and check equality.

The dominant component of the algorithm in the worst case again has time complexity $O(\prod_{i=1}^{m} c_i''!)$. But as before, this worst case is unlikely, and the typical case is much better.

C. Check whether $\mathcal{P}_1 \equiv \mathcal{P}_2(G_1)$

For *strict incorporation*, the node-count (NC) and edge-count (EC) must be such that ($NC_{G_2} > NC_{G_1}$ and $EC_{G_2} \geq EC_{G_1}$) OR ($NC_{G_2} = NC_{G_1}$ and $EC_{G_2} > EC_{G_1}$).

We derive SP_1 and SP_2 and check if every SP_1 class has an equivalent SP_2 class (with equal or higher class size). $\mathcal{P}_2$ can *strictly incorporate* $\mathcal{P}_1$, only if this requirements is met. We then generate FMs by examining all of the permutations of policies within equivalent partitions. For each FM, we construct and check their adjacency matrices for equality.

D. Check whether $\mathcal{P}_1 \subset \mathcal{P}_2(\mathbf{G_1})$

Again, we follow the approach in §V-C, but check if $SP_1 \subset SP_2$. We derive Table III and use (3) to get FMs. Finally, we determine the adjacency matrices for each mapping in FMs and check for equality. If equal, then $\mathcal{P}_1 \subset \mathcal{P}_2(G_1)$.

The algorithms in § V-C and § V-D have the same time complexity of § V-B, with similar best and worst-case values.

TABLE IV: *High-level summary of SCADA case studies adapted from [19].*

SUC	Configuration date	Firewall type	Firewalls*	Zones	Conduits	Flow-policies	Equivalence classes	Minimum class size	Maximum class size	Policy
1	Sep 2011	Cisco IOS	3	7	11	22	12	1	7	$\mathcal{P}_1 = (G_1, P_1)$
2	Aug 2011	Cisco ASA	6	21	81	162	87	1	8	$\mathcal{P}_2 = (G_2, P_2)$
3	Oct 2011	Cisco PIX	4	10	17	34	15	1	8	$\mathcal{P}_3 = (G_3, P_3)$
4	Mar 2011	Cisco ASA	3	9	16	32	16	1	5	$\mathcal{P}_4 = (G_4, P_4)$

TABLE III: *Mappings between inclusive equivalence-classes of semantic partitions SP_1 and SP_2,* e.g., *the first line of the table implies that $e_1 \subset e'_1$ and $e_1 \subset e'_2$.*

SP_1 class	class size	included in SP_2 class(es)	total size of SP_2 class(es)	mappings-set
e_1	c_1	e'_1, e'_2	$c'_1 + c'_2$	M_1
e_2	c_2	e'_2	c'_2	M_2
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
e_m	c_m	e'_1, e'_3, e'_m	$c'_1 + c'_3 + c'_m$	M_m

VI. A SERIES OF CASE STUDIES

We now demonstrate the use of our proposed algebras, through four real case studies summarised in Table IV. The data was provided by the authors of [19], and comes from real firewall-network configuration case studies.

There are four Systems Under Consideration (SUCs), involving various firewall architectures and models. We use them to demonstrate several properties, most notably that the computational complexities of our algorithms are tractable.

The most important feature is the *size of the equivalence classes* in the semantic partitions of the firewall policies of each network. Table IV lists the *minimum* and *maximum* class sizes. We see that the maximum equivalence class size of a semantic partition is *small* relative to the flow-policy count.

This is to be expected. An *equivalence class* groups a set of conduits with identical policies. It is easy to argue that if many conduits have identical policies then the zone-conduit diagram of a network is *badly designed*. Many equivalent policies leads to zones with identical reachability, which might be amalgamated. The subsequent reduction in complexity makes policy specification easier and less error prone.

As a result, in the real networks we studied, the maximum equivalence class size was 8. Moreover, most of the class sizes are small (the majority have only *one member*). That makes the algorithms presented above computationally feasible. We describe such networks as close to being *semantically disjoint*.

We demonstrate this in detail below by comparing the policies of $\mathcal{P}_1$ and $\mathcal{P}_4$ whose zone-flow models are shown in Figure 3. We have performed similar tests on the other pairs.

The two policies cannot be *equivalent* or *inclusive* as the node and edge counts don't match, so we immediately proceed to test incorporation, *i.e.,* $\mathcal{P}_1 \equiv \mathcal{P}_4(G_1)$ and $\mathcal{P}_1 \subset \mathcal{P}_4(G_1)$.

The node and edge counts (NC and EC) of $\mathcal{L}(G_1)$ and $\mathcal{L}(G_2)$ meet the *incorporation* criteria in §V-C. So, we proceed to derive and compare the equivalence classes of SP_1 with those of SP_4 and find that not every SP_1 class has an *equivalent* SP_4 class. So, $\mathcal{P}_4$ *does not strictly incorporate* $\mathcal{P}_1$. But as Table V illustrates, $SP_1 \subset SP_4$.

Since $|e_{12}| > |e'_{17}| = 5$, every policy in e_{12} cannot be *included* in a policy in e'_{17}. So, P_1 policy tuple $(p_1, p_2, \dots, p_{22})$ cannot be *included* in any P_4 tuple (with 22 elements). We can conclude that $\mathcal{P}_4$ *does not partially incorporate* $\mathcal{P}_1$, without further evaluation. So, the policy between, for instance, SCADA and Corporate zones across both networks enable *non-equivalent* and *non-inclusive* sets of traffic services.

Malachite automates all of these comparisons, and in this case takes 24 seconds to run on a standard desktop computer (*e.g.,* Intel Core CPU 2.7-GHz computer with 8GB of RAM running Mac OS X). We sought to compare *Malachite's* performance to previous work [15] that enabled less comprehensive firewall policy comparisons, but have been unable to obtain their software, to date.

We checked each policy in Table IV for compliance with the industry-recommended policy ($\mathcal{RP}$) [3] through inclusion. *Malachite* generates the semantic-partitions SP_i and SP_{RP} and checks whether $SP_i \subset SP_{RP}$. Alarmingly *none* of the above policies were included by $\mathcal{RP}$, indicating potential *non-compliance*. Hence, *every* real-world SCADA network we studied was significantly vulnerable to cyber attack!

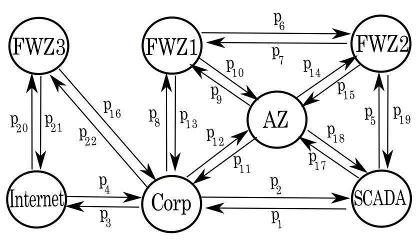
We used *Malachite* to compute the semantic difference between each policy and $\mathcal{RP}$. We could then identify the recommended-practice violations in detail and rectify them to reduce each SUC's vulnerability to cyber attack (see Listing 1).

At the time of our analysis, we were unable to obtain time evolved versions of the policies in Table IV. So, we used synthetic policy versions instead and derived the *semantic difference* for each case to observe how easily interpretable our output results are (see [17] for details).

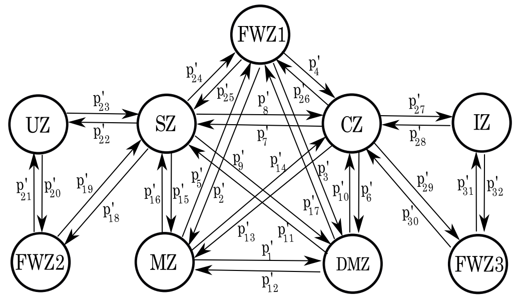
Malachite can also accurately compute the functional dis-

TABLE V: *Mappings between inclusive equivalence-classes of semantic partitions SP_1 and SP_4.*

SP_1 class	$\subset SP_4$ class(es)	total size	mappings
$e_1 = \{p_1\}$	e'_1	1	M_1
$e_2 = \{p_2\}$	e'_2	1	M_2
$e_3 = \{p_4\}$	e'_3	1	M_3
$e_4 = \{p_5, p_6\}$	e'_4	4	M_4
$e_5 = \{p_7\}$	$e'_4 - e'_7,$	15	M_5
$e_6 = \{p_8, p_{22}\}$	$e'_7 - e'_9$	7	M_6
$e_7 = \{p_{11}, p_{13}\}$	e'_1, e'_{10}	2	M_7
$e_8 = \{p_{16}\}$	$e'_2, e'_3, e'_5, e'_{11} - e'_{14}$	9	M_8
$e_9 = \{p_{17}\}$	e'_1, e'_{15}	2	M_9
$e_{10} = \{p_{19}\}$	e'_1, e'_{12}, e'_{14}	3	M_{10}
$e_{11} = \{p_{12}, p_{20}\}$	e'_1, e'_{16}	2	M_{11}
$e_{12} = \{p_3, p_9, p_{10}, p_{14}, p_{15}, p_{18}, p_{21}\}$	e'_{17}	5	M_{12}



(a) Zone-Flow model of SUC1.



(b) Zone-Flow model of SUC4.

Fig. 3: *Zone-Flow models of two real SCADA case studies.*

Listing 1: *Malachite output of the recommended-practice violations of SUC1 firewall policy (partial listing).*

```
WARNING suc_policy1.policyml violates ISA practices
WARNING policy violations::
corp−>scada: {prot=17; udp.dst_port=53; udp.src_port=1024−65535;}
corp−>scada: {prot=6; tcp.dst_port=53; tcp.src_port=1024−65535;}
corp−>scada: {prot=6; tcp.dst_port=21; tcp.src_port=1024−65535;}
corp−>scada: {prot=6; tcp.dst_port=20; tcp.src_port=1024−65535;}
corp−>scada: {prot=6; tcp.dst_port=80; tcp.src_port=1024−65535;}
```

crepancies between multiple policy designs from users in distinct policy sub-domains (*e.g.,* SCADA engineers, Corporate admins). By doing so, these users can clearly identify the design differences and negotiate on what variations are acceptable and so on.

However, Malachite also has several limitations, for one, checking a policy against a best-practice policy relies on the best-practices being correct to begin with. Also, when a policy is a subset of a best-practice policy that may not always mean the policy is compliant. For instance, the best-practices recommend enabling TCP port 443 (*i.e.,* HTTPS) inbound to the SCADA-Zone as it's deemed as a safe protocol. But allowing so, doesn't guarantee compliance as non-HTTPS traffic can still be tunnelled through this port. The final mapping of subsets also need to be manually examined for correctness, so, some human intervention is still required.

VII. Conclusion

There are various obstacles that hinder the meaningful comparison of firewall policies. Most prominent is the lack of standards for firewall policy specification which makes policy semantics rule-order and firewall-implementation dependent. To compound the problem, a firewall policy with a given semantic can also be constructed using different rule sets.

Malachite addresses these challenges, allowing direct comparisons of firewall policies. We employ it to check that firewall policies are recommended-practices compliant and to analyse their change impact.

Acknowledgment

The authors would like to thank input and suggestions from Tim Griffin. This project was supported by an Australian Postgraduate Award, Australian Research Council Linkage Grant LP100200493, and CQR Consulting.

References

[1] E. S. Al-Shaer and H. H. Hamed. Discovery of policy anomalies in distributed firewalls. In *Annual Joint Conference of the IEEE CCS*, pages 2605–2616. INFOCOM, 2004.

[2] C. J. Anderson, N. Foster, A. Guha, J.-B. Jeannin, D. Kozen, C. Schlesinger, and D. Walker. Netkat: Semantic foundations for networks. *ACM SIGPLAN Notices*, 49(1):113–126, 2014.

[3] ANSI/ISA-62443-1-1. Security for industrial automation and control systems part 1-1: Terminology, concepts, and models, 2007.

[4] C. Avelsgaard. *Foundations for Advanced Mathematics*. Scott, Foresman/Little, Brown Higher Education, 1990.

[5] L. Babai. Graph isomorphism in quasipolynomial time. *arXiv preprint arXiv:1512.03547*, 2015.

[6] Y. Bartal, A. Mayer, K. Nissim, and A. Wool. Firmato: A novel firewall management toolkit. *ACM TOCS*, 22(4):381–420, 2004.

[7] P. Billingsley. Probability and measure. *A Wiley-Interscience Publication, John Wiley & Sons, New York*, 1995.

[8] E. Byres, J. Karsch, and J. Carter. NISCC good practice guide on firewall deployment for SCADA and process control networks. *NISCC*, 2005.

[9] K. D. Gourley and D. M. Green. Polygon-to-rectangle conversion algorithm. *IEEE CGA*, pages 31–32, 1983.

[10] J. D. Guttman and A. L. Herzog. Rigorous automated network security management. *IJIS*, 4(1-2):29–48, 2005.

[11] F. Harary and R. Z. Norman. Some properties of line digraphs. *Rendiconti del Circolo Matematico di Palermo*, 9(2):161–168, 1960.

[12] C. D. Howe. *What's Beyond Firewalls?* Forrester Research, Incorporated, 1996.

[13] H. Hu, W. Han, G.-J. Ahn, and Z. Zhao. Flowguard: Building robust firewalls for software-defined networks. In *Proceedings of the third workshop on Hot topics in software defined networking*, pages 97–102. ACM, 2014.

[14] D. S. Johnson. The NP-completeness column. *ACM Transactions on Algorithms*, 1(1):160–176, July 2005.

[15] A. X. Liu and M. G. Gouda. Diverse firewall design. *Parallel and Distributed Systems, IEEE Transactions on*, 19(9):1237–1251, 2008.

[16] A. Mayer, A. Wool, and E. Ziskind. Fang: A firewall analysis engine. In *IEEE Symposium on Security and Privacy*, pages 177–187, 2000.

[17] D. Ranathunga, M. Roughan, P. Kernick, and N. Falkner. Malachite: Firewall policy comparison, http://tinyurl.com/o2ke3py. *Technical Report*, 2015.

[18] D. Ranathunga, M. Roughan, P. Kernick, and N. Falkner. Towards standardising firewall reporting. In *1st Workshop on the Security of Cyber Physical Systems (WOS-CPS)*. LNCS, 2015.

[19] D. Ranathunga, M. Roughan, P. Kernick, N. Falkner, and H. Nguyen. Identifying the missing aspects of the ANSI/ISA best practices for security policy. In *1st ACM Workshop on Cyber-Physical System Security (CPSS)*, pages 37–48. ACM, 2015.

[20] A. Tongaonkar, N. Inamdar, and R. Sekar. Inferring higher level policies from firewall rules. *LISA*, 7:1–10, 2007.

[21] A. Wool. Architecting the Lumeta firewall analyzer. In *USENIX Security Symposium*, pages 85–97, 2001.

[22] L. Yuan, H. Chen, J. Mai, C.-N. Chuah, Z. Su, and P. Mohapatra. FIREMAN: A toolkit for firewall modeling and analysis. In *IEEE SSP*, pages 15–213, 2006.