IEEE International Symposium on Policies for Distributed Systems and Networks

Model Checking Firewall Policy Configurations

Alan Jeffrey
Security Research Department
Bell Labs, Alcatel-Lucent
2701 Lucent Lane, Lisle, IL 60532
Email: ajeffrey@bell-labs.com

Taghrid Samak
School of Computing
DePaul University
243 South Wabash Ave, Chicago, IL 60604
Email: taghrid@cdm.depaul.edu

***Abstract*—The use of firewalls to enforce access control policies can result in extremely complex networks. Each individual firewall may have hundreds or thousands of rules, and when combined in a network, they may result in unexpected combined behavior. To mitigate this problem, there has been recent interest in the use of model checking techniques for analyzing the behavior of firewall policy configurations, and reporting anomalies. Existing techniques for firewall policy analysis are based on decision diagrams, most normally reduced ordered Binary Decision Diagrams (BDDs). BDDs are a rich data structure, supporting more logical operations than just solving boolean formulae. Typically, search algorithms for boolean satisfiability (so-called SAT-solvers) outperform BDDs. In this paper, we show that the extra structure provided by BDDs is not necessary for firewall policy analysis, and that SAT solvers are sufficient. This argument is supported both by theoretical analysis and by experimental data.**

I. Introduction

Firewall configuration is a crucial element of implementing a network security policy. Such configurations typically take the form of filter rulesets or policies specifying properties of packets to be dropped or accepted. Each filter rule is given as a condition predicate over packet headers, together with an action to be performed on matching packets.

Individual firewall configurations are often complex, with hundreds or thousands of rules, and the behavior of a network with many firewalls may be difficult to establish without mechanical assistance. As a result, there has been increasing interest in tools to analyze network configurations, such as the work of Al-Shaer and Hamed [1], Yuan et al. [2], and Matoušek et al. [3].

At the heart of any tool for firewall policy analysis is a model of condition predicates. Existing tools make use of data structures based on decision diagrams, such as Binary Decision Diagrams (BDDs) [4], which provide mechanisms for checking satisfiability of quantified boolean formulae[1].

Satisfiability of quantified boolean formulae (QSAT) is the canonical PSPACE-complete problem, in the same way that satisfiability of boolean formulae (SAT) is the canonical NP-complete problem. Heuristics for solving NP-complete problems are known as SAT-solvers, often based on the DLL [5] search procedure. Modern SAT-solvers include zChaff [6] and MiniSAT [6], and are surveyed in [7].

The disadvantage of using a SAT-solver compared with BDDs is expressivity: SAT-solvers only solve SAT, not QSAT, so tools that use SAT-solvers cannot handle quantifiers in the way that BDD-based tools can. The advantage is that SAT-solvers are often more efficient than BDDs.

The observation that SAT-solvers can often replace BDDs in analyzing systems was made in Biere et al.'s seminal paper on Bounded Model Checking [8], where system properties (such as assertions in C code failing) are translated into instances of SAT and passed to a SAT-solver. This translation requires a bound on size of the problem (such as the number of lines of C code executed), so does not support complete analysis of cyclic systems, even in cases when model checking succeeds.

In this paper, we discuss making use of the techniques of Bounded Model Checking in analyzing firewall policy configurations. In particular, we observe that cyclicity in a firewall configuration is an error (a packet will cycle forever without being accepted or dropped) and so we can reject any cyclic policy without further analysis. As a result, model checking is NP-complete rather than PSPACE-complete, and so is a suitable target for a SAT-solver.

Our new contribution on firewall policy analysis is:

- a formal model of firewall policy (based on IPtables/Netfilter [9]),
- a proof that reachability and cyclicity of policy configurations are both NP-complete[2],
- a translation of networks of firewalls into a single firewall instance, and
- experimental evidence for comparing the performance of BDDs with SAT-solvers.

In this paper, we will investigate two properties of firewall policy configurations:

- *Reachability*: for each rule r of the policy, there is a

[1]Quantified boolean formulae extend boolean formulae (expressions written from boolean variables, conjunction, disjunction and negation) with universal and existential quantification over boolean variables. $x \wedge \neg y$ is a boolean formula, while $\forall x \,.\, \exists y \,.\, (x \wedge \neg y)$ is a quantified boolean formula.

[2]At first sight, this appears to contradict prior work [2] which shows reachability to be $O(n)$, but that result assumed $O(1)$ operations on packet sets. This is discussed further in Section II-B.

978-0-7695-3742-9/09 $25.00 © 2009 IEEE
DOI 10.1109/POLICY.2009.32

```
iptables -N FWDIN
iptables -A FWDIN -i eth0 -s 192.168.1.0/24 -j FWDOUT
iptables -A FWDIN -i eth1 -s 192.0.2.0/24 -j FWDOUT
iptables -A FWDIN -i eth2 -s ! 192.168.1.0/24,192.0.2.0/24 -j FWDOUT
iptables -A FWDIN -j DROP

iptables -N FWDOUT
iptables -A FWDOUT -o eth0 -i ! eth2 -d 192.168.1.0/24 -j ACCEPT
iptables -A FWDOUT -o eth1 -d 192.0.2.0/24 -j ACCEPT
iptables -A FWDOUT -o eth2 -j ACCEPT
iptables -A FWDOUT -j DROP

iptables -A FORWARD -j FWDIN
```

Figure 1. Example iptables configuration script

packet p which causes rule r to fire (a firewall containing unreachable rules is probably misconfigured)

- *Cyclicity*: there is a packet p which causes the firewall to enter an infinite loop, without accepting or rejecting the packet (a cyclic firewall is certainly misconfigured).

Extending those properties to the case of multiple firewalls and network configurations makes them even more interesting, where reachability of rules is related to host reachability. Our analysis is not limited to these properties, and we expect would scale to a language for specifying firewall properties.

II. The Model

In this section we will start by modeling single firewall policy. We will then discuss the complexity of both reachability and cyclicity checking. We will then present the model for network configuration as a single firewall instance.

A. Firewall Configuration

The following model of firewall policies is based on IPtables/Netfilter [9]. An example IPtables configuration script is given in Figure 1. An *atomic model* $\mathcal{M} = (\Sigma, \Phi, \sigma)$ consists of a finite set of *atoms* Σ, a finite set of *atomic propositions* Φ and a *valuation function* $\sigma : \Sigma \times \Phi \to \mathsf{Bool}$. We will often write $\mathcal{M}$ for the set Σ, for example writing $\mathcal{M} \to X$ for the set of functions $\Sigma \to X$.

We shall use an atomic model of records where:

- atoms are records of the form $\{f_1{=}v_1, \ldots, f_n{=}v_n\}$,
- atomic properties are equalities of the form $f = v$, and
- valuation $\sigma(r, f = v)$ is true whenever $(f{=}v) \in r$.

Our examples include atomic models for packets such as:

$\{\mathsf{dst}_0{=}192, \mathsf{dst}_1{=}168, \mathsf{dst}_2{=}1, \mathsf{dst}_3{=}1, \mathsf{dport}{=}80, \ldots\}$

and atomic models for firewall states such as:

$\{\mathsf{chain}{=}\mathsf{forward}, \mathsf{rule}{=}0, \mathsf{oiface}{=}\mathsf{eth0}, \ldots\}$

We can form the product of atomic models $\mathcal{M} = \mathcal{M}_1 \times \mathcal{M}_2$, where $\Sigma = \Sigma_1 \times \Sigma_2$, $\Phi = \Phi_1 + \Phi_2$, and $\sigma(x_1, x_2, \phi) = \sigma_i(x_i)$ when $\phi \in \Phi_i$.

Let $\mathcal{B}(\mathcal{M})$ be the boolean formulae over atomic propositions drawn from $\mathcal{M}$, that is formulae given by the grammar:

$A, B, C ::= \mathsf{true} \mid \mathsf{false} \mid A \wedge B \mid A \vee B \mid \neg A \mid \phi$

where ϕ is an atomic proposition from $\mathcal{M}$. We extend the valuation function to $\sigma : \Sigma \times \mathcal{B}(\mathcal{M}) \to \mathsf{Bool}$, and write $x \vDash A$ whenever $\sigma(x, A) = \mathsf{true}$.

A *firewall policy configuration* $\mathcal{C} = (\mathcal{P}, \mathcal{S}, \gamma, \delta)$ consists of an atomic model of *packets* $\mathcal{P}$, an atomic model of *states* $\mathcal{S}$, a *decision function* $\gamma : \mathcal{S} \to \mathcal{B}(\mathcal{P} \times \mathcal{S})$ and a *transition relation* $\delta \subseteq \mathcal{S} \times \mathsf{Bool} \times \mathcal{S}$. We write $p \vDash s \to t$ whenever $(s, \sigma(p, s, \gamma(s)), t) \in \delta$. A firewall policy configuration is *deterministic* when the transition relation is a function $\delta : \mathcal{S} \times \mathsf{Bool} \to \mathcal{S}$.

Such firewall configurations can be viewed graphically as decision graphs, for example in Figure 2 we give the decision graph for the IPtables configuration script in Figure 1.

For readers familiar with temporal logic (see, for example [10] and [11]) a firewall policy configuration is essentially a Kripke structure indexed by a packet model.

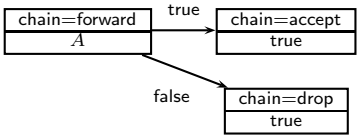
B. NP-hardness

An atomic model $\mathcal{M} = (\Sigma, \Phi, \sigma)$ is *boolean-valued* whenever $\Sigma \subseteq \mathsf{Bool}^k$ and and Φ is a set of boolean formulae in k chosen variables. A *rooted* atomic model is an atomic model $\mathcal{M}$ together with a boolean formula $\mathsf{init} \in \mathcal{B}(\mathcal{M})$. In our examples, init is $(\mathsf{chain} = \mathsf{forward})$. For the remainder of this section, we assume packet models to be boolean-valued, and state models to be rooted.

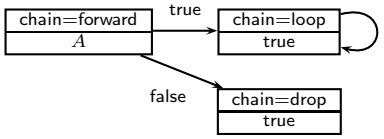
A state $t \in \mathcal{S}$ is *reachable* in a firewall configuration whenever there exists a packet p and state s such that $p \vDash s \to \cdots \to t$ and $s \vDash \mathsf{init}$.

A firewall policy configuration is *cyclic* whenever there exists a packet p and states s and t such that $p \vDash s \to \cdots \to t \to \cdots \to t$ and $s \vDash \mathsf{init}$.

It is direct to see that checking reachability and cylicity are NP-hard. For reachability, given any boolean formula A in k variables, we let the packet model be Bool^k, and the firewall configuration be given by:



The state $\{\mathsf{chain} = \mathsf{accept}\}$ is reachable in this firewall configuration precisely when A is satisfiable. Cyclicity is similar, except that the configuration is given by:



Note that these constructions rely on an arbitrary packet model $\mathcal{P}$. If $\mathcal{P}$ is fixed, then we can exhaustively search the packet space in fixed time. So reachability and cyclicity in firewall configurations collapse to reachability and cyclicity in graphs, which can be performed in linear time.

The difference between a fixed and variable packet model is the reason why previous work [2] provides a linear time algorithm for reachability: they rely on $O(1)$ set operations on packets. In practice, the number of bits may be fixed

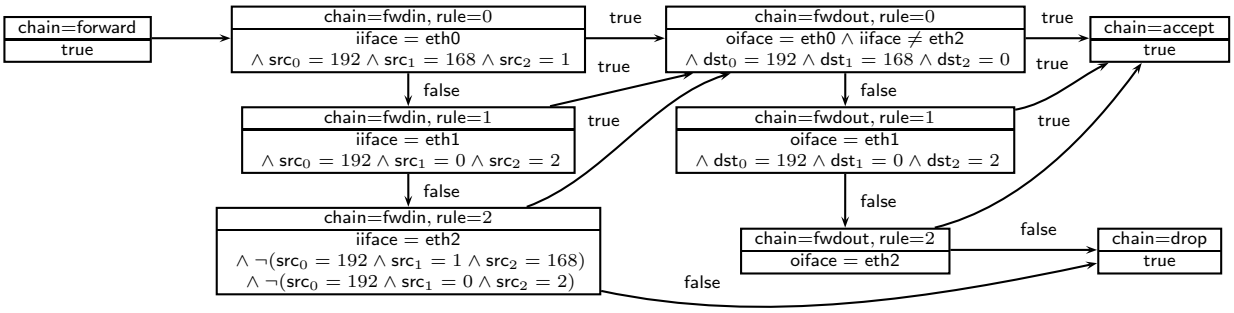


Figure 2. Example decision graph

(e.g. the 104 bits of IP packet headers), so the linear time result is justified. We note, however, that a constant factor of 2^{104} is quite high (for example, there have been $\sim 2^{80}$ 1GHz CPU cycles since the big bang). Moreover, extensions to the model, for example to support IPSec tunneling will require a variable packet model.

Theorem 1: Checking reachability and cyclicity of firewall policy configurations are both NP-hard.

C. NP-completeness

We now show that checking reachability and acyclicity of firewall configurations is in NP, by reduction to SAT. This reduction forms the heart of our model-checker, as discussed in Section III-A.

For a given firewall configuration $\mathcal{C} = (\mathcal{P}, \mathcal{S}, \gamma, \delta)$, we introduce boolean variables:

- x_i for each $i \leq k$, the chosen variables for the packet model $\mathcal{P} \subseteq \mathsf{Bool}^k$, and
- $y_{(s,b,t)}$ for each $(s,b,t) \in \delta$, which is true in any run which includes the transition s, b, t.

and define boolean formula $\mathsf{trans}(\mathcal{C})$ over those variables as:

$$\begin{aligned}\mathsf{trans}(\mathcal{C}) = \forall (t,c,u) \in \delta \,.\, y_{(t,c,u)} \rightarrow \\ \mathsf{trans}(t, \gamma(t)) \leftrightarrow c \\ \wedge\, \mathsf{trans}(t, \mathsf{init}) \vee \exists (s,b,t) \in \delta \,.\, y_{(s,b,t)}\end{aligned}$$

where $\mathsf{trans}(s, A)$ is defined to be:

$$\begin{aligned}\mathsf{trans}(s, \mathsf{true}) &= \mathsf{true} \\ \mathsf{trans}(s, \mathsf{false}) &= \mathsf{false} \\ \mathsf{trans}(s, B \wedge C) &= \mathsf{trans}(s, B) \wedge \mathsf{trans}(s, C) \\ \mathsf{trans}(s, B \vee C) &= \mathsf{trans}(s, B) \vee \mathsf{trans}(s, C) \\ \mathsf{trans}(s, \neg B) &= \neg \mathsf{trans}(s, B) \\ \mathsf{trans}(s, \phi) &= \begin{cases} \phi & \text{when } \phi \in \mathcal{P} \\ \sigma(s, \phi) & \text{when } \phi \in \mathcal{S} \end{cases}\end{aligned}$$

A satisfying assignment for $\mathsf{trans}(\mathcal{C})$ corresponds to a set of runs of $\mathcal{C}$. We can then define a boolean formula for reachability of a state t as:

$$\mathsf{reach}(\mathcal{C}, t) = \mathsf{trans}(\mathcal{C}) \wedge \exists (s,b,t) \in \delta \,.\, y_{(s,b,t)}$$

and cyclicity as:

$$\begin{aligned}\mathsf{cyclic}(\mathcal{C}) = \mathsf{trans}(\mathcal{C}) \\ \wedge \forall (s,b,t) \in \delta \,.\, y_{(s,b,t)} \rightarrow \exists (t,c,u) \in \delta \,.\, y_{(t,c,u)}\end{aligned}$$

A satisfying assignment for $\mathsf{reach}(\mathcal{C}, t)$ corresponds to a set of runs of $\mathcal{C}$, one of which contains state t, and hence to reachability of t. A satisfying assignment for $\mathsf{cyclic}(\mathcal{C})$ corresponds to a set of runs of $\mathcal{C}$ where every reachable state has a successor, and hence to cyclicity of $\mathcal{C}$. Together, these translations give an algorithm for translating reachability and cyclicity of firewall configurations into SAT, and so (together with the previous NP-hardness) are NP-complete. This makes them good candidates for implementing using a SAT-solver.

Theorem 2: Checking reachability and cyclicity of firewall configurations are both NP-complete.

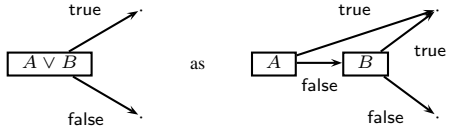
D. Conjunctive Firewall Configurations

A potential argument against our model of firewall configurations is that it is more expressive than is seen in practice, as it allows an arbitrary boolean formula $\gamma(s)$ to be used as the condition in state s, whereas firewall implementations such as IPTables [9] have limited support for disjunction. In this section we shall show that firewall policies with arbitrary conditions can be reduced in linear time to ones with no uses of disjunction, and so our model is just as succinct as the model without disjunction.

Define a boolean formula over atomic model $\mathcal{M}$ to be *conjunctive* when it does not use disjunction, and negation is only allowed on atoms, that is given by the grammar:

$A, B, C ::= \mathsf{true} \mid \mathsf{false} \mid A \wedge B \mid \phi \mid \neg\phi$

A firewall configuration is conjunctive whenever $\gamma(s)$ is, for every s. To reduce an arbitrary firewall configuration to a conjunctive one, we first use De Morgan duality to push negation to atoms, then rewrite:



This reduction gives us that conjunctive firewall configurations are equally expressive and succinct as arbitrary firewall configurations.

Theorem 3: Checking reachability and cyclicity for firewall configurations reduces in linear time to checking conjunctive firewall configurations.

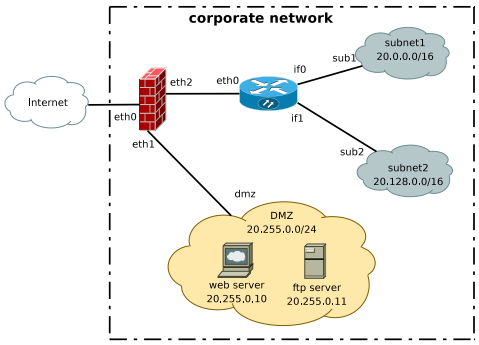


Figure 3. Sample network configuration

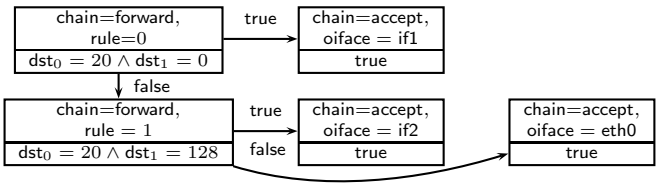


Figure 4. Router policy configuration

E. Network Configurations

Another potential argument against our model of firewall policies is that it only deals with a single firewall, and ignores the network topology. In this section we give a model of network topology with multiple firewalls, and show that it can be reduced to a single firewall configuration. We will consider the sample network configuration given in Figure 3.

We first observe that our model of firewall configuration supports packets which terminate at the firewall, and which can make routing decisions based on packet headers. We can therefore model subnets and routers as firewalls, and can focus on networks where every node in the network acts as a firewall.

A *network element* $\mathcal{E} = (\mathcal{C}, \mathcal{I}, \mathcal{O}, \iota, \alpha)$ is a firewall configuration $\mathcal{C} = (\mathcal{P}, \mathcal{S}, \gamma, \delta)$ together with a finite set of *input interfaces* $\mathcal{I}$, a finite set of *output interfaces* $\mathcal{O}$, an *initial state* function $\iota : \mathcal{I} \to \mathcal{S}$, and an *accepting state* function $\alpha : \mathcal{O} \to \mathcal{S}$.

A *network configuration* $\mathcal{N} = (\mathcal{V}, \mathcal{E}, \rho)$ is an atomic model $\mathcal{V}$ of *vertices*, a network element $\mathcal{E}_v$ for each vertex $v \in \mathcal{V}$, and a *routing function* $\rho : \mathcal{O} \to \mathcal{I}$ (where we define $\mathcal{I} = \sum\{\mathcal{I}_v \mid v \in \mathcal{V}\}$ and $\mathcal{O} = \sum\{\mathcal{O}_v \mid v \in \mathcal{V}\}$).

For example, the network in Figure 3 has six network elements: internet, fw, router, subnet1, subnet2, and dmz. Each has appropriate interfaces, for example $\mathcal{I}_{\mathsf{router}} = \mathcal{O}_{\mathsf{router}} = \{\mathsf{eth0}, \mathsf{if0}, \mathsf{if1}\}$, and a routing function given by the network topology, for example $\rho(\mathsf{eth0}) = \mathsf{eth2}$, $\rho(\mathsf{if0}) = \mathsf{sub1}$ and $\rho(\mathsf{if1}) = \mathsf{sub2}$. Each node has a configuration, for example $\mathcal{C}_{\mathsf{router}}$ is given in figure 4.

A state of such a network configuration is given as coming from $\mathcal{S} = \sum\{\mathcal{S}_v \mid v \in \mathcal{V}\}$. We write $p \vDash s \to t$ for the state transition relation given by either:

- an *intra-firewall* transition where $s, t \in \mathcal{S}_v$ and $p \vDash s \to t$ in $\mathcal{C}_v$, or
- an *inter-firewall* transition where $s = \alpha_v(O)$ and $t = \iota_w(\rho(O))$, for some $O \in \mathcal{O}_v$ and $\rho(O) \in \mathcal{I}_w$.

We can then define the reachable states and cyclic network configurations as before.

We note that this model is considerably more complex than the model for a single firewall configuration, and that configurations may display misconfigurations that arise due to interactions of apparently well-configured network elements. For example, in Figure 3, if fw is configured to route packets with destination $20.*.*.*$ to output interface eth2, then a packet with destination 20.1.1.1 will cause a cycle, with fw forwarding the packet to router, which forwards it back to fw. Such misconfigurations which arise due to network element interaction are difficult to detect other than by analysis tools.

We will now show that analysis of network configurations can be reduced to analysis of single firewalls in linear time.

Given a network configuration $\mathcal{N}$, define the firewall configuration $\mathcal{C}(\mathcal{N}) = (\mathcal{P}, \mathcal{S}, \gamma, \delta)$ to be one which inherits $\mathcal{S}$ and γ from $\mathcal{N}$, and where δ is defined to be:

$$\delta \stackrel{\text{def}}{=} \{(s,b,t) \in \delta_v \mid s,t \in \mathcal{S}_v\} \cup \{(\alpha_v(O), b, \iota_w(\rho(O))) \mid O \in \mathcal{O}_v, \rho(O) \in \mathcal{I}_w\}$$

A run for $\mathcal{C}(\mathcal{N})$ corresponds to a run of $\mathcal{N}$, which gives us the desired reduction.

Theorem 4: Checking reachability and cyclicity for network configurations reduces in linear time to checking firewall configurations.

III. Experimental Methodology

A. Implementation

The implementation of the model is a realization of the reduction described in section II-C to generate a SAT instance, which is then handed to a SAT-solver for solution. The core component of the system maps a network configuration to a boolean expression in conjunctive normal form. Reachability and cyclicity properties can be checked by finding a satisfying assignment to the model with certain additional constraints.

Figure 5 shows a high level view of the implementation. Following is a functional summary for each component.

- The network configuration file describes the network topology, IP addresses and subnet masks. The policy files give firewall configurations for each node.
- The solver maps the configuration to a SAT instance, with boolean variables x_i for packet header bits (104 variables for the basic five tuples), and transition variables $y_{(s,b,t)}$ (2 variables per rule).
- The model is solved first for testing cyclicity in the configuration. If the configuration is acyclic, reachability for each rule is tested.

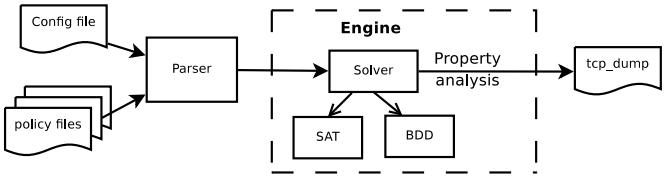


Figure 5. Implementation Components

Topology variables	$2-28$	interfaces (in/out) and policies
Transition variables	$29-60$	two variable per rule
Packet variables	61-164	

Table I
VARIABLE INDEXES GENERATED BY MINISAT

- The satisfying assignment of the model gives a complete run for a packet entering the network. This includes the possible packet reaching the rule, all rules in the configuration that have been fired, and rules tested but not fired.
- The packet header fields are then converted to readable format for future inspection.

Our implementation reads configurations in IPTables format, and writes packets in tcp-dump format.

The SAT and BDD components are both generic classes that use any SAT-solver or any BDD package. For our experiments, an incremental solver (MiniSAT [12]) was used, and BuDDy [13] was used for the BDD class.

We also implemented the *Fireman* algorithm [2] for comparison. Experimental results are presented in Section IV.

Example: We will show here a sample SAT instance generated for the small network from figure 3. Policies are defined for the firewall, router and each subnet. The overall number of rules is 16. Subnet policies have only two rules; accept packet if the destination is in the subnet, otherwise forward packet to the router. Table I shows the indexes generated by MiniSat for each model variable. Clauses generated for the SAT instance are shown in table II.
The first group of clauses describes the connectivity of interfaces in the network. The second corresponds to the transitions between rules. A transition from the first routing rule in figure 4 to the first rule in the first subnet will happen when a packet satisfies $\mathsf{dst}_0 = 20 \wedge \mathsf{dst}_1 = 0$. The final group models the transition when the condition is not satisfied.

B. Datasets

Extensive evaluation of the model was performed on single firewall policy configuration. Policies were generated with different sizes ($10-3000$ filtering rules). Three classes of synthetic policies were considered:

1) *Random policies*: are policies generated with random assignment to packet header fields for each policy rule. The fields are defined using masks for IP addresses, and ranges for port numbers.

Interface connection variables	-2 13 2 -13 ...	$v_2 \Rightarrow v_{13}$ $v_{13} \Rightarrow v_2$
True transition clauses	-29 -61 -29 62 -29 63 ...	$v_{29} \Rightarrow \neg v_{61}$ $v_{29} \Rightarrow \neg v_{62}$ $v_{29} \Rightarrow \neg v_{63}$
False transition	-45 61 -62 -63	$v_{45} \Rightarrow v_{61} \wedge \neg v_{62} \ldots$

Table II
GENERATED CLAUSES FOR THE NETWORK IN FIGURE 3

2) *Structured policies*: are more representative of actual policies defined on firewalls. The policies are structured so that specific rules have more priority and reflect exceptions to more general rules with lower priority and opposite action. For example a rule that accepts traffic to host $10.0.0.20$ has higher priority than a rule that denies $10.0.0.0/24$. Those policies also span a limited portion of the domain of all packet fields, so they are more representative of a single domain-specific firewall configuration.
3) *Structured with don't care*: This is a more realistic class of policies where some of the packet header fields might not be mentioned in the filtering decision (don't care). For example, the source port number is rarely if ever used in rule specifications, and specific port values are more common than general port ranges.

Random policies may be helpful in testing a wide range of configurations, as there is no attempt to model the structure of real-world firewalls. In particular, they may uncover corner cases which structured policies do not. However, they may not model real-world performance. Moreover, all algorithms for SAT-solving (including BDDs) perform extremely poorly on random input, and rely on regularity in the SAT instance to perform optimization, so we expect to see significantly higher performance on structured datasets.

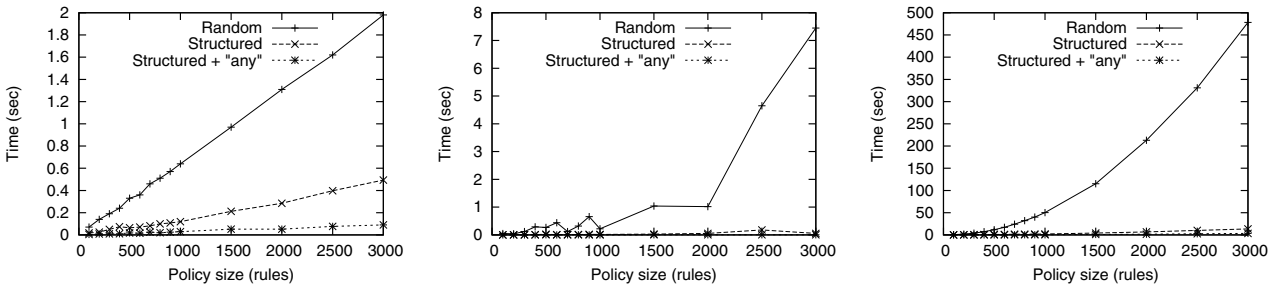
IV. EXPERIMENTAL RESULTS

We compare four algorithms:

- our translation of firewall policy to SAT, solved using BDDs,
- our translation of firewall policy to SAT, solved using a SAT-solver,
- Yuan et al.'s translation of firewall policy to SAT, solved using BDDs, and
- Yuan et al.'s translation of firewall configurations to SAT, solved using a SAT-solver.

For each experiment, we considered the run times for translating the problem into SAT, for solving for cyclicity (which is not covered by Yuan et al.) and for solving for reachability for each policy rule. Each experiment was carried out with random, structured and structured-with-don't-care datasets.

The results here are presented considering a single policy configuration. As discussed earlier, network configurations



(a) Initialization (b) Cyclicity (c) Reachability

Figure 6. Performance of our algorithm using different policy classes

can be reduced to single firewall configurations in linear time, so the analysis here is sufficient.

A. Dataset selection

Figure 6 shows the comparison between each policy class when applying our algorithm. As predicted, random policies always take more time than structured policies. For all cases, the model initialization time is linear. The reachability and cyclicity tests are very efficient with realistic policies.

We will proceed now by evaluating SAT implementation versus BDD implementation using structured policies.

B. Results for our algorithm in SAT vs in BDDs

We will first investigate the implementation of our model using a SAT-solver (MiniSat) and BDDs (Buddy). Our initial experiment used a set of policies with small number of rules (1–100). The BDD performed very poorly on most of the cases, where the run time for 100-rule policy would take 2 minutes as opposed to (0.01 seconds) for MiniSat. Figure 7 shows the evaluation results for small policies. Although the reachability time for BDD-based implementation is comparable to the SAT one, the initialization time dominates the overall performance of BDD.

C. Results for our algorithm in SAT vs Fireman in BDDs

Reachability in our model can be mapped to two of the anomalies detected by *Fireman*: shadowing and redundancy. For both anomalies, a rule may not be reached since there is a previous rule covering its space either with the same or different action. In this section we compare the policy anomaly detection algorithm performed by *Fireman* with our test for rule reachability.

We implemented the *Fireman* algorithm [2] using BDDs. Figure 8(a) shows the comparison between SAT implementation of our model, and the BDD implementation of the *Fireman* framework. It is clear that their algorithm outperforms ours when processing all rules in the configuration.

This result is surprising, as we expected the BDD model to be less efficient, but we hypothesize that this is due to the number of variables. Our model introduces one variable per packet bit, plus two variables per rule, where Fireman only uses packet bits. When policies become large, the number of packet bits is swamped by the number of rule bits, resulting in higher performance for Fireman.

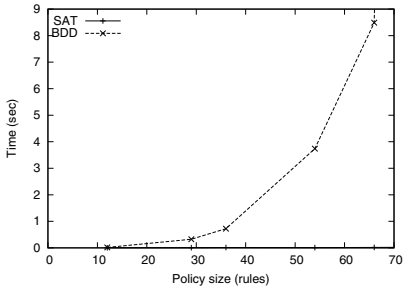
Furthermore, we note that the Fireman algorithm uses a linear pass through the firewall configuration: testing reachability of rule n requires testing reachability of rules $1, \ldots, n-1$. As a result, Fireman is significantly more efficient than our algorithm when testing reachability of all rules, but not for testing reachability of just one rule. Reachability for single rule is more useful when analyzing network configuration with multiple firewalls. In Figure 8(b) we show comparative timings for checking reachability of a single randomly chosen rule, as the size of rulesets grows.

D. Results for Fireman in BDDs vs in SAT

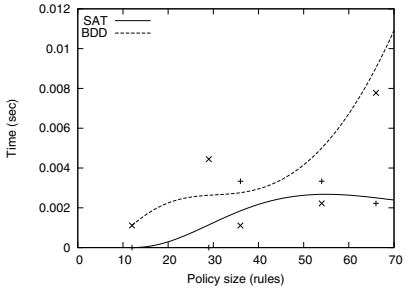
To test our hypothesis that the performance advantage for Fireman is due to the translation of policies into SAT rather than due to BDDs, we reimplemented Fireman using a SAT-solver rather than a BDD package. In doing so, we note that Fireman makes no use of BDD features which are outside of SAT (there are no uses of quantifiers, and no equality comparisons between predicates in their algorithm).

The reimplementation of Fireman using a SAT-solver proceeds through the firewall rules in order, maintaining a SAT instance S. At rule n, the SAT instance matches the set of packets which have not been handled by rules $1, \ldots, n-1$. To check reachability of rule n we solve the SAT instance $S \wedge C_n$ where C_n is the condition predicate for rule n. We then update $S := (S \wedge \neg C_n)$. In the case of a condition C_n which is a conjunction of literals (that is, one using bitmasks, with no uses of negation or ranges) we note that S is in CNF with no uses of auxiliary variables, and so the Fireman algorithm is a good match to an incremental CNF SAT-solver such as MiniSAT.

As seen in Figure 9, the performance of both implementations is comparable for small policies (< 1500 rules). As policy size increases (which is the case for large domains), the SAT-based implementation outperforms the BDD. The BDD expression built while solving anomalies using *Fireman* becomes very complicated with large number of rules.

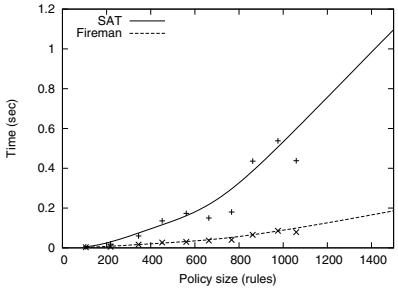


(a) Initialization

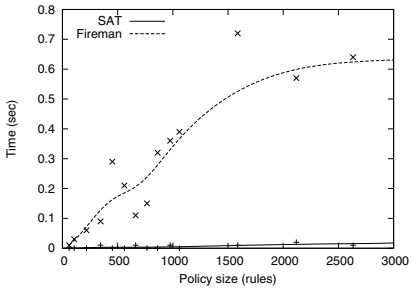


(b) Reachability

Figure 7. Performance of our algorithm using SAT versus BDD implementations



(a) Reachability of all rules



(b) Reachability of a randomly chosen rule

Figure 8. Performance of our algorithm (SAT) versus Fireman (BDD)

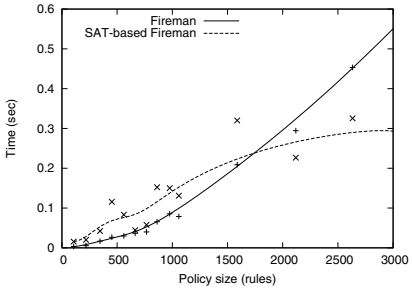


Figure 9. Evaluation of the SAT-based implementation of the Fireman algorithm

For each incoming rule, more minterms are added, and the BDD building time causes performance degradation. SAT-solvers are very efficient when dealing with large instances.

V. Related Work

Previous work of interest related to model checking firewall configuration can be grouped into two categories; configuration analysis tools, and model checking frameworks.

A. Configuration Analysis Tools

Several firewall analysis tools have been developed, such as the work of Al-Shaer and Hamed [1] and Yuan at al. [2]. These tools aim to discover configuration inconsistencies, which result in undesired behavior for the firewall. Specific categories of conflicts as discussed in both [1] and [2] are *shadowing*, *generalization*, and *correlation*.

While we focus here on reachability (the inverse of shadowing) and cyclicity, we expect that SAT-based techniques would apply to generalization and correlation as well.

In the recent work by Matoušek et al. [3], a dynamic network analysis is performed to verify security properties in networks with changing topologies. First order logic was used to represent packet filtering elements in the network and generalize the regular rule set representation. The proposed model provides general property analysis, although the time complexity and experimental results were not included.

Tools focused on firewall design, architecture and performance include Gouda and Liu [14], Mayer, Wool and Ziskind [15] and Wool [16] (which includes automated conflict detection). Other firewall analysis tools, such as [17] and [18], focus on the performance of firewalls in terms of implementation and filtering delays.

For general network configuration analysis, the most related work to our approach is the work by Xie et al. [19]. This work addressed the reachability of hosts in a network including routers, firewalls and NAT devices. The complexity of this algorithm is linear in the number of network nodes, assuming a fixed packet model.

B. Model Checking Frameworks

The closest work related to model checking policies are the work of Hamed et al. [20] and Yuan et al. [2]. Both of the frameworks perform analysis over multiple paths for the packet. Paths are calculated before applying model checking, and BDDs are used to solve for anomalies over each path.

This approach can be characterized by the *non-symbolic* representation of paths, compared to the *symbolic* representation of packets. This requires any algorithm performing search over paths to perform exhaustive search, compared to search over packets, which is performed by the BDD representation. In comparison, our approach represents packets and paths symbolically, so (for example) the search for a cyclic path can be carried out by the SAT-solver rather than by an explicit search through all possible paths.

Hamed et al. [20] also utilized BDD in model checking firewalls, VPN and IPSec policies. They used a similar approach to *Fireman* where model checking is performed

at the lower level of the operations after calculating paths and expanding the search tree.

In [3] a formal model for security analysis was introduced. The model was implemented using interval decision diagrams (IDD, rather than BDDs). The approach is similar to the *Fireman* model, since it also expands all possible packet paths and then property checking is performed.

VI. Conclusion and Future Directions

In this paper we presented a model for firewall configuration. We investigated the use of SAT solvers in the model analysis. Two properties of interest in policy definition were analyzed; reachability and cyclicity. The problem of analyzing a firewall configuration with respect to those properties was proved to be an NP-complete problem, hence the use of a SAT solver is justified. A model for network configuration is also presented based on single firewall model. Experimental evaluation was conducted to compare our model with recent BDD-based configuration analysis approaches. Using SAT solvers in firewall policy analysis was shown to be efficient over BDD analysis especially when the problem size increases.

As future directions the following will be investigated:

- Extending our model to cover more general properties using W3C XPath notation. Experimental validation will be performed on the general model.
- Possible applications for the model and its implementation will be investigated. Generating packets as test cases for firewall behavior is one of them.
- As shown in the results, one of the most performance degradation points in the BDD is the building and initialization time. This limits the applicability of BDD-based analysis in dynamic environments where configuration modifications and policy edits are performed frequently. Analysis on the efficiency of SAT versus BDD in changing domains will be investigated.

References

[1] E. S. Al-Shaer and H. H. Hamed, "Discovery of policy anomalies in distributed firewalls," in *Proc. IEEE Conf. Computer Communications*, 2004, pp. 2605–2616.

[2] L. Yuan, J. Mai, Z. Su, H. Chen, C.-N. Chuah, and P. Mohapatra, "FIREMAN: A toolkit for firewall modeling and analysis," in *Proc. IEEE Symp. Security and Privacy*.

[3] P. Matoušek, J. Ráb, O. Ryšavy, and M. Svéda, "A formal model for network-wide security analysis," in *Proc. IEEE Int. Conf. Engineering of Computer Based Systems*, 2008.

[4] R. E. Bryant, "Graph-based algorithms for boolean function manipulation," *IEEE Trans. Computers*, vol. C-35, no. 8, pp. 677–691, 1986.

[5] M. Davis, G. Logemann, and D. Loveland, "A machine program for theorem proving," *Communications of the ACM*, vol. 5, no. 7, pp. 394–397, 1962.

[6] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik, "Chaff: Engineering an efficient SAT solver," in *Proc. 39th Design Automation Conference*, 2001.

[7] J. Gu, P. W. Purdom, J. Franco, and B. W. Wah, "Algorithms for the satisfiability (SAT) problem: A survey," in *Satisfiability Problem: Theory and Applications*, ser. DIMACS Series in Discrete Mathematics and Theoretical Computer Science. American Mathematical Society, 1997, pp. 19–152.

[8] A. Biere, A. Cimatti, E. Clarke, and Y. Zhu, "Symbolic model checking without BDDs," in *Proc. Tools and Algorithms for the Construction and Analysis of Systems*, ser. Lecture Notes in Computer Science. Springer-Verlag, 1999, pp. 193–207.

[9] "The netfilter.org project," http://www.netfilter.org/.

[10] W. Thomas, "Automata on infinite objects," in *Handbook of Theoretical Computer Science*, J. van Leeuwen, Ed. MIT Press, 1990, vol. B, ch. 4, pp. 133–192.

[11] E. A. Emerson, "Temporal and modal logic," in *Handbook of Theoretical Computer Science*, J. van Leeuwen, Ed. MIT Press, 1990, vol. B, ch. 16, pp. 955–1072.

[12] N. Eén and N. Sörensson, "An extensible SAT-solver," in *Proc. Theory and Applications of Satisfiability Testing*, ser. Lecture Notes in Computer Science, vol. 2919. Springer, 2003, pp. 502–518.

[13] J. Lind-Nielsen, "Buddy - a binary decision diagram package," http://buddy.sourceforge.net/, 2004.

[14] M. G. Gouda and A. X. Liu, "Firewall design: Consistency, completeness, and compactness." in *Proc. Int. Conf. Distributed Computing Systems*, 2004, pp. 320–327.

[15] A. Mayer, A. Wool, and E. Ziskind, "Fang: A firewall analysis engine," *Proc. IEEE Symp. Security and Privacy*, 2000.

[16] A. Wool, "Architecting the Lumeta firewall analyzer," in *Proc. USENIX Security Symposium*, 2001, p. 7.

[17] M. R. Lyu and L. K. Y. Lau, "Firewall security: Policies, testing and performance evaluation," in *Proc. Computer Software and Applications Conference*, 2000, pp. 116–121.

[18] B. Hickman, D. Newman, S. Tadjudin, and T. Martin, "Benchmarking methodology for firewall performance," RFC 3511, 2003.

[19] G. G. Xie, J. Zhan, D. A. Maltz, H. Zhang, A. G. Greenberg, G. Hjálmtýsson, and J. Rexford, "On static reachability analysis of IP networks," in *Proc. IEEE Computer and Communications Societies*, 2005, pp. 2170–2183.

[20] H. H. Hamed, E. S. Al-Shaer, and W. Marrero, "Modeling and verification of IPSec and VPN security policies," in *Proc. IEEE Int. Conf. Network Protocols*, 2005, pp. 259–278.