Paper Discussion: Policy Advisor and FIREMAN

Claas Lorenz
Institute for Computational Science, Potsdam University, Potsdam, Germany
email: cllorenz@cs.uni-potsdam.de

Berichte Institut für Informatik und Computational Science
UP, ISSN 0946-7580, TR-2014-2

Paper Discussion: Policy Advisor and FIREMAN

Claas Lorenz

Institute for Computational Science, University of Potsdam, Potsdam, Germany
email: cllorenz@cs.uni-potsdam.de

1 Introduction

Firewalls are an integral part of the defense strategy for most organizations and security aware users. They provide fine grained access control mechanisms that allow the implementation of a sophisticated security policy. The richness of options enables a variety of misconfigurations and incosistencies which are reffered to as anomalies. The first systematic approach for the detection of anomalies was the Policy Builder provided by Al-Shaer et. al. (see [3], [4], [5], [6], [2]). Later Yuan et. al. (see [9]) introduced the FIREMAN algorithm which used model checking techniques for analysis. This idea was affiliated by Jeffrey and Samak (see [8]) who followed another model checking paradigm and whose algorithm challenged FIREMAN experimentally. This paper focuses on Policy Advisor and FIREMAN which will be introduced in chapter 2 and 3. Both sections give a detailed analysis of the main issues related to the different approaches and allow a well founded comparison. Most prominent these are:

- the expressiveness of the model,
- the detected anomalies and
- the complexity of the algorithm.

In section 4 an overall comparison of Policy Advisor, FIREMAN and the approach of Jeffrey and Samak will be given, followed by a prospect for future work.

2 Policy Advisor

The Policy Builder (see [3], [4], [5], [6], [2]) is a top-down modeling and analyzing approach for distributed firewalls. The main idea is to model Access Control Lists (ACL, also referred to as firewalls) and networks with an abstract description language and compile the results into real firewall rulesets like iptables or pf. In the following sections there will be a description of the model (in 2.1) detected anomalies (in 2.2) and the algorithm (in 2.3). The latter concludes with a brief discussion of the complexity.

2.1 The Model

Models for Policy Advisor are simple lists of rules with intermediate actions. Rules consist of fields that can be either integers or don't cares and are used to describe whether a rule matches. Rules have the canonical form <order>,<protocol>,<s_ip>,<s_port>,<d_ip>,<d_port>,<action> where

- <order>: index of the rule within the ACL

- <protocol>: protocol field, may be either tcp or udp
- <s_ip>,<d_ip>: source and destination IPv4 address
- <s_port>,<d_port>: source and destination port number
- <action>: action of the rule, may be either accept or deny

Fields of rules can be compared in terms of set relations. These are $\subset$, $\supset$, $=$ and $\neq$. For example the more specific port 80 is a subset of the more general don't care * or formally <d_port=80>$\subset$<d_port=*>. To evaluate two canonical rules their fields are compared pairwise. To transform a rule into canonical form all fields that are not stated explicitly are handled as don't cares. The possible relations between two rules are called completely disjoint, exactly matching, inclusively matching and partial disjoint. See [3] for a formal definition.

2.2 Detected Anomalies

Certain combinations of relations paired with filtering patterns result in anomalies that are regarded as errors or warnings. For internal rules of a firewall these are shadowing, correlation, generalization and redundancy. The intuitions are as follows (a formal definition of all anomalies is given in [3]):

Shadowing A rule is not reachable due to rules handling all packets that it matches differently. An example would be the following ACL:

1, tcp, 1.2.3.4, *, *, * deny

2, tcp, 1.2.3.4, 80, *, *, accept

The first rule blocks all packets from 1.2.3.4 regardless of the port number. This leaves no traffic for the second rule to match. Shadowing is considered to be an error.

Correlation Correlated rules have different filter actions and overlapping sets of matching packets. The first rule matches traffic that the second matches and vice versa. For example in:

1, tcp, 140.192.37.20, *, *, * deny

2, tcp, *, *, 161.120.33.40, 80, accept

Both rules match tcp traffic while the first rule blocks some packets and the second one allows some packets. If both rules were switched exactly the traffic that was blocked orignally would be allowed and the traffic allowed would be blocked. This circumstance might not be intentioned by the administrator and is therefore labelled as warning.

Generalization Generalization takes place if a previous more specific rule makes an exception of a later and more general rule. Even though this might be a very sound intention of an administrator to allow exceptions from a general policy rule (i.e. deny-all) it is considered a warning since it may also occur unintentioned. The following example shows such a case where the more general rule does not seem to be a policy rule:

1, tcp, 140.192.37.20, *, *, * deny

2, tcp, 140.192.37.*, *, *, accept

Redundancy A rule is redundant to a later rule if they have the same filtering action and the second rule filters the same traffic as the first one. This makes the first rule irrelevant for the overall filtering properties of the ACL. The first rule may be removed to reduce the amount of rules and therefore save space. But the first rule might be intended as a shortcut for traffic that is likely to be handled very often. This rule would reduce the amount of rules to be checked before a match and thus increase the overall performance. As both scenarios could be intended redundancies are regarded as warnings. For an example see:

1, tcp, 140.192.37.*, *, 161.120.33.40, 53 accept

2, tcp, *.*.*.*, *, 161.120.33.40, 53, accept

Networks Within networks rules of so called upstream and downstream firewalls are compared. On a path through a network an upstream firewall is located closer to the internet while a downstream firewall is closer to the internal network. To find inter firewall anomalies two rules on a network path are compared to find shadowing, redundancy, correlation and spuriousness anomalies. The first three anomalies are the same ones as for the intra firewall detection. This prerequisites that there are no such anomalies withing any firewall on the path. The latter anomaly, spuriousness, occurs if an upstream firewall allows traffic that is blocked downstream. This scenario might also be intended by the administrator to separate a part of the network (i.e. an internal network with workstations) from a part with lower security standards like a DMZ. Therefore, this anomaly is considered to be a warning. The following example gives the intuition:

1, tcp, 140.192.*.*, *, *.*.*.*, 25 accept

2, tcp, 140.192.*.*, *, *.*.*.*, *, deny

2.3 The Algorithm and its Complexity

The detection algorithm is a straight forward application of the set relation analysis. It is done by using automata for the inter firewall detection (see figure 1) and its pendant on the intra firewall level (see figure 2). The complexity to compare two rules is in $O(1)$ since the rules have a fixed size which forces every run of the automata to have the same duration. Since a firewall is a simple list which represents exactly one path where rules are compared pairwise the complexity is in $O(n^2)$ with n rules. The complexity for the checking of networks highly depends on the topology. Every path is checkable in $O(n^2)$ but networks tend to have much more paths. If the network is regarded as a tree the complexity grows to $O(max(m,n)^3)$ with m paths and n rules on the longest path. If the network has an arbitrary graph topology the complexity is exponential in space since loops must be detected to avoid paths of infinite length. Another issue is the comparison of rule fields. The model does not allow to define rules with unusual CIDR addresses like 192.255.0.0/9. The encoding for this rule would result in 128 single rules with 192.128.*.*, 192.129.*.*, 192.130.*.*, ..., 192.255.*.*. Therefore, the encoding of the complete address range would result in an exponential space complexity in the number of rules. This reduces the applicability of the approach significantly.

3 FIREMAN

The FIREMAN algorithm (see [9]) is a different approach to detect anomalies in distributed firewalls. It has another focus than the Policy Builder since it analyzes existing firewalls instead of modeling them in the first place. This bottom-up view enforces a more complex model to handle features of real world firewalls like jumps and returns to and from other ACL. Section 3.1 gives an overview of the model used by FIREMAN. Afterwards in 3.2 detected anomalies and their severity are described. In 3.3 follows a brief description of the FIREMAN algorithm and its complexity is discussed. To improve the understanding of the algorithm in 3.4 a short example is given.

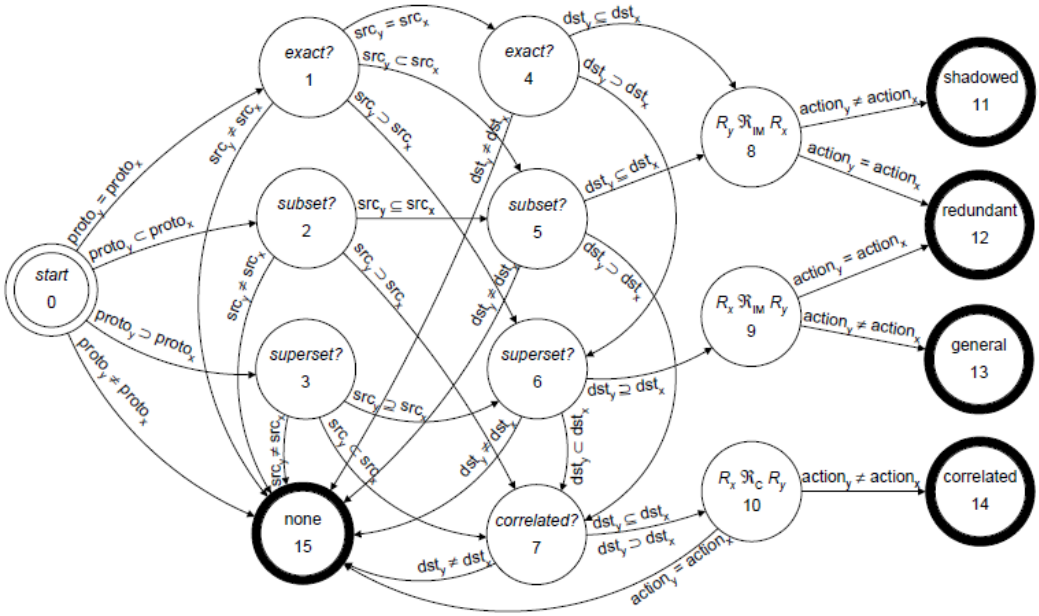


Figure 1: Automaton for the intra firewall anomaly detection (obtained from [5]).

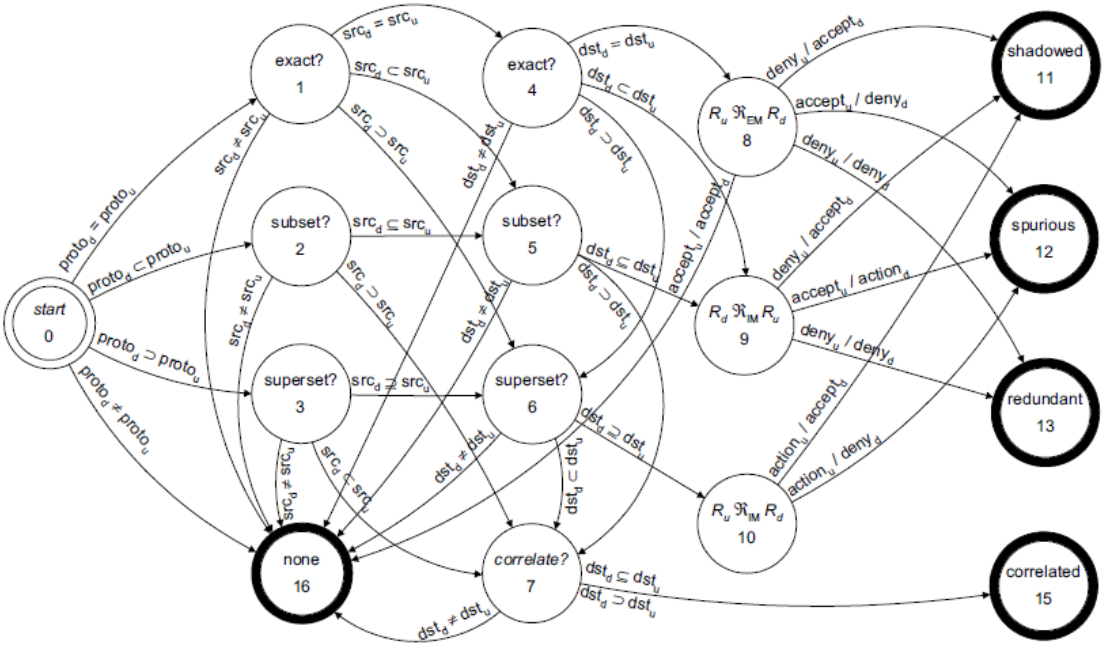
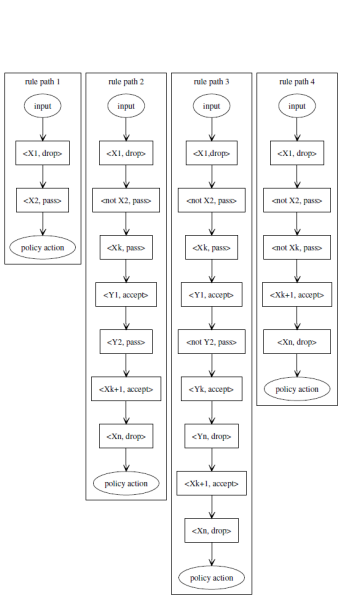


Figure 2: Automaton for the inter firewall anomaly detection (obtained from [5]).

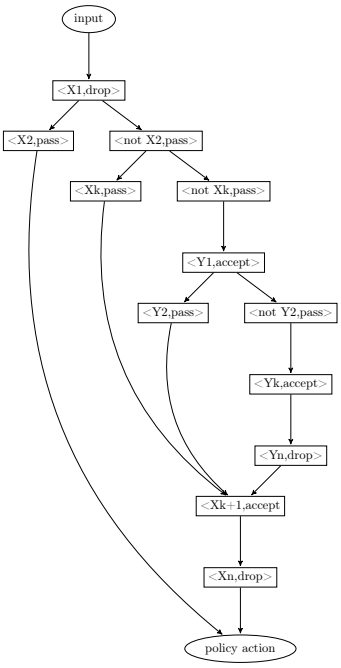
3.1 The Model

In FIREMAN models are gained by parsing existing firewalls and translate them into an intermediate format. Rules have the format <P, action> with a matching predicate P and a resulting action∈{accept, deny, chain, return}. The latter two actions are used to handle complex ACL

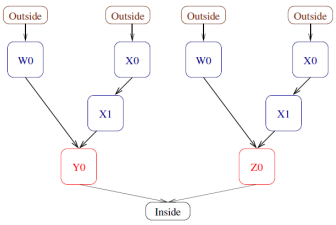
that can branch to other ACL and return after checking. The anomaly detection algorithm runs on paths which are represented by simple lists. To map branching onto lists the ”pass” action is introduced and used whenever a branch occurs. For every branch a list is created. One with <P, pass> followed by the targeted ACL and another with <¬P, pass> indicating that no jump was performed. See figure 3a for an example. To limit the amount of paths it is necessary to prevent loops which are detected by allowing every rule to occur at most once on a path. If this constraint is violated FIREMAN stops with an error. To save space all paths of a (complex) ACL can be summarized into a directed tree with shared pre- and postfixes which is also shown in figure 3b. To model networks (also referred to as distributed firewalls) ACL-models are arranged as a tree corresponding to the network topology (see figure 3c). To reduce the amount of ACL to be checked the authors of FIREMAN suggest to check network gateways (i.e. Internet-to-DMZ or DMZ-to-Internal) only. This is sound if no Man-in-the-Middle occurs within the checked network. If such an attacker existed he would be able to manipulate the packet flow at will and make the anomaly detection obsolete. Therefore, requiring his nonexistence is a necessity. Another argument for this point of view lies in the nature of model checking which founds the basis of FIREMAN. It is a static verification method and therefore, meant to be executed before runtime to prevent possible violations. Checks during runtime are out of focus and should be done by other security entities like IDS. The FIREMAN algorithm relies heavily on the usage of sets and their operations like



(a) Paths of a firewall (obtained from [9]).



(b) Firewall as directed tree with space compression.



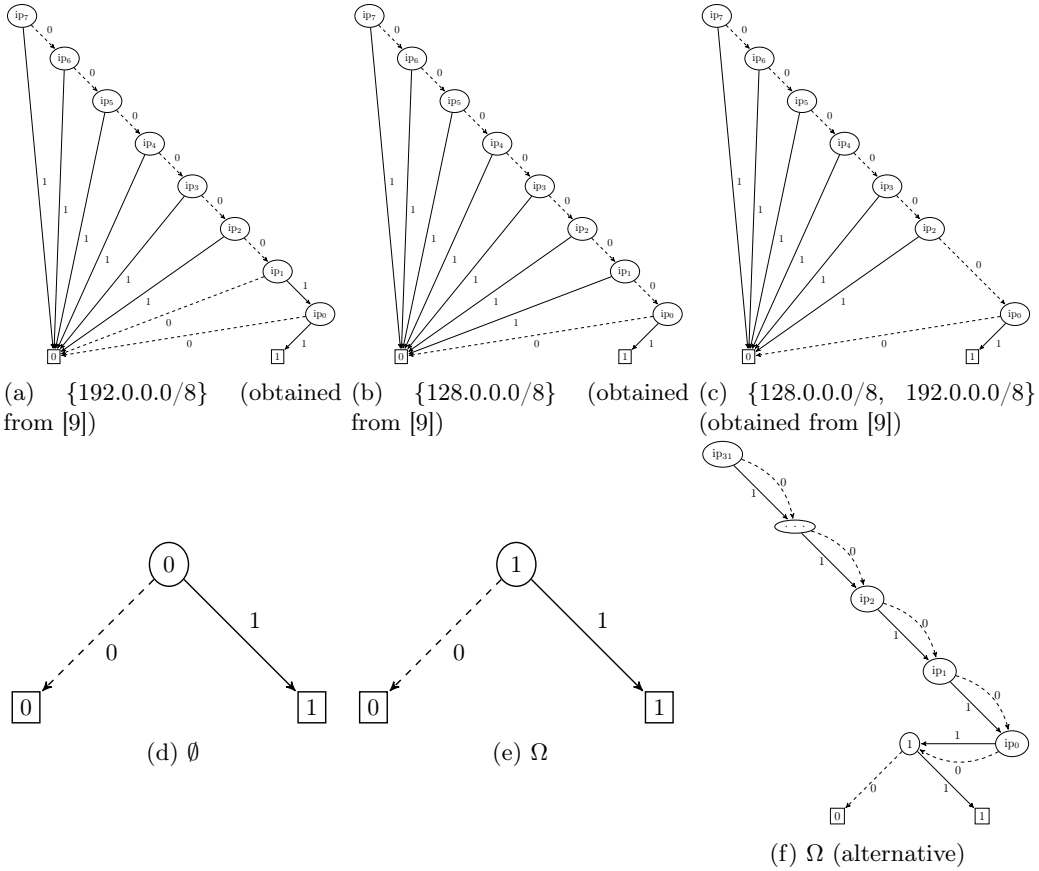
(c) Network of firewalls (obtained from [9]).

Figure 3: Different representations of models in FIREMAN.

union, intersection, etc. To handle these needs efficiently Binary Decision Diagrams (BDD, see [1] for details) are utilized. A BDD is defined as an:

- acyclic, directed, connected and rooted Graph $G = (V, E)$.
- It has exactly two leafs named 0 and 1 respectively.
- Inner nodes have exactly two leaving edges labelled with 0 and 1 respectively. Alternatively the 0-edge is drawn dashed while the 1-edge remains solid.

Following this definition every node is either a leaf or an inner node. BDD provide a way to express sets in a very compact way while allowing very fast operations that run in $O(1)$. Initially coming from the sphere of representing boolean functions traversing a BDD allows to evaluate those functions. A set of bits is represented by variables (inner nodes of the BDD) and the edges between them show boolean correlations. Figure 4a shows the set of the IPv4 range 192.0.0.0/8 (or binary: 10000000_2). Beginning at the root all nodes evaluate to zero except for ip_0 that results in a one. If one of the least significant nodes evaluates to one the boolean function returns zero showing that this bit combination does not belong to the set. The same applies if ip_0 evaluates to zero. Figures 4b and 4c show examples for the set {128.0.0.0/8} and {128.0.0.0/8,192.0.0.0/8}. An important issue is the representation of the empty set and the full set. Figure 4d shows the empty set. It needs an auxiliary variable permanently set to 0 to meet the definition of a BDD. The same applies for the full set in figure 4e. 4f shows an alternative approach for the full set. The complete IPv4 address range can be represented by a linear structure with an auxiliary node constantly set to 1. By giving the variable order explicitly some performance risks for BDD might be avoided. Even though these examples show very little overhead in space while representing large sets of information their worst case space complexity is exponential leaving its usage at risk.



(a) {192.0.0.0/8} (obtained from [9])

(b) {128.0.0.0/8} (obtained from [9])

(c) {128.0.0.0/8, 192.0.0.0/8} (obtained from [9])

(d) $\emptyset$

(e) Ω

(f) Ω (alternative)

Figure 4: Sets represented by BDD.

3.2 Detected Anomalies

Like Policy Advisor FIREMAN supports a broad variety of anomalies that can be detected. These are divided into policy violations and inconsistencies. The first are regarded as errors while the latter raise warnings. On the intra firewall level the only detected violations is shadowing.

Inconsistencies are generalization, correlation, redundancy and verbosity. This new anomaly shows rules, that can be summed up to a single rule and provide a gain of efficiency for the firewall. On the network level policy violations consist of shadowing and checking against black- and whitelists. There is just a single inconsistency named cross-path which means that on different paths through the network a certain packet is treated differently.

3.3 Algorithm and Complexity

The main idea of the FIREMAN algorithm is to incrementally traverse the model and consistently update some sets holding information about blocked and allowed packets. Since the model is represented by a tree the algorithm covers all possible paths through the network and holds in finite time. Before every increment the sets can be checked for anomalies. To check an ACL the following sets are utilized:

- The input I which holds all packets that may enter the ACL.
- The acceptance set A that represents all packets that are accepted by the ACL.
- The deniance set D which is a set of packets that are denied by the ACL.
- The divertance set F contains packets that are diverted to other data paths.

A state for the j-th rule of an ACL is defined as (A_j,D_j,F_j). The remainance set is $R_j = I \cap \neg(A_j \cup D_j \cup F_j)$ which shows a set of packets that can possibly arrive at rule j+1 because they were not filtered, accepted or diverted yet. State transitions can be done as follows:

$$(A, D, F), < P, accept > \vdash (A \cup (R \cap P), D, F)$$

$$(A, D, F), < P, deny > \vdash (A, D \cup (R \cap P), F)$$

$$(A, D, F), < P, pass > \vdash (A, D, F \cup (R \cap \neg P))$$

Before every transition the detection for anomalies is performed. For details have a look at table 2 in appendix A. When the ACL is completely traversed decisions can be made by calculating

$$A_{ACL} = \bigcup_{i \in paths} A_{path_i}$$

$$D_{ACL} = \bigcup_{i \in paths} D_{path_i}$$

over all paths of the ACL. Similarly the ideal input set for the ACL can be determined by $I_{ACL} = A_{ACL} \cup D_{ACL}$ (the initial input was the full set $I = \Omega$). By applying the default action of the ACL (default blocking) the remainance set is $R_{ACL} = \emptyset$.
The checking of a network is done by subsequently calculating the acceptance and deniance sets of the ACL on network trees' nodes. The output of an ACL forms the input of the next ACL. For this serial view the following operations are applied:

$$A = \bigcap_{acl \in n} A_{acl}$$

$$D = \bigcup_{acl \in n} D_{acl}$$

If two or more paths unite in a node its input can be calculated in a parallel manner as follows:

$$A = \bigcup_{acl \in n} A_{acl}$$

$$D = \bigcap_{acl \in n} D_{acl}$$

The initial input is the full set $I = \Omega$ and the intermediate sets are initialized by $A_0 = D_0 = F_0 = \emptyset$. During all operations the checking for anomalies can be performed by surveilling if certain set constellations occur. They are listed in table 2 in appendix A.
Since set operations with BDD run in $O(1)$ (also see [7]) the checking of a path is linear in the number of its rules. By using trees to represent a set of paths through a firewall the runtime complexity is $O(max(m,n)^2)$ where m is the number of paths and n is the number of rules on the longest path. This circumstance does not change when moving from single firewalls to networks of firewalls which are also represented by trees. The disadvantage of BDD lies in their exponential space complexity. The same is true for the loop detection which is necessary to build the trees.

3.4 Example

To improve the understanding of the FIREMAN algorithm a brief example is proposed. Given a simple firewall configuration

1. <s_ip=1.2.0.0/16, d_p=22, deny>
2. <s_ip=192.0.0.0/8, d_p=80, accept>
3. <s_ip=1.2.3.4, d_p=22, accept>

the following table shows the actions performed by FIREMAN including the intermediate results:

Index j	P_j,action	$P_j \cap R_{j-1} \stackrel{?}{=} \emptyset \wedge (P_j \stackrel{?}{\subseteq} A_{j-1} \vee P_j \stackrel{?}{\subseteq} D_{j-1})$	A_j	D_j	F_j	R_j
0	-	–	$\emptyset$	$\emptyset$	$\emptyset$	Ω
1	{s_ip=1.2.0.0/16, d_p=22},deny	$\bot$	$\emptyset$	$\emptyset \cup (\Omega \cap P_1)$ = {s_ip=1.2.0.0/16, d_p=22}	$\emptyset$	$\Omega \cap \neg(\emptyset \cup D_1 \cup \emptyset)$ = $\Omega \backslash$ {s_ip=1.2.0.0/16, d_p=22}
2	{s_ip=192.0.0.0/8, d_p=80}, accept	$\bot$	$\emptyset \cup (R_1 \cap P_2)$ = {s_ip=192.0.0.0/8, d_p=80}	{s_ip=1.2.0.0/16, d_p=22}	$\emptyset$	$\Omega \backslash$ {s_ip=1.2.0.0/16, s_ip=192.0.0.0/8, d_p=22, d_p=80}
3	{s_ip=1.2.3.4, d_p=22}, accept	$\top$	$A_2 \cup (R_2 \cap P_3)$ = {s_ip=192.0.0.0/8, d_p=80}	{s_ip=1.2.0.0/16, d_p=22}	$\emptyset$	$\Omega \backslash$ {s_ip=1.2.0.0/16, s_ip=192.0.0.0/8, d_p=22, d_p=80}

The third column is the check for a shadowing anomaly. The main points during the different phases are:

0. In the first step no rule is processed and thus, no anomaly detection is performed. Instead all sets are initialized with their default values.
1. The first rule is processed by checking for an anomaly. The first part {s_ip=1.2.0.0/16, d_p=22} $\cap \Omega \neq \emptyset$ fails which indicates that no shadowing anomaly is present so far. Since the rule's action is "deny" the deniance set D is updated with the rule body. Afterwards it is subtracted from the remainance set R, too.
2. Since the second rule matches different traffic than the rule before the check for shadowing fails again by evaluating the first part of the formula. This is followed by an update of the acceptance set and the removal of the rule's body from the remainance set.
3. Finally, checking the last rule leads to the detection of an anomaly. The first part of the formula holds as well as the second part of the braced expression which is {s_ip=1.2.3.4, d_p=22} $\subseteq$ {s_ip=1.2.0.0/16, d_p=22}.

4 Conclusion and Future Work

Both Policy Advisor and FIREMAN detect a broad variety of anomalies. While the first offers a complete classification the latter provides a more sophisticated view which seems to be more

	Policy Advisor	FIREMAN	Jeffrey, Samak [8]
Model (intra)	simple ACL (List)	complex ACL (Tree)	Kripke-Structure
Model (inter)	Tree	Tree	Kripke-Structure
Complexity	$O(n^2)$ for single paths, $O(n^3)$ for networks	$O(n^2)$, but exponential in space	$O(2^n)$
Anomalies (intra)	Shadowing, Correlation, Generalization, Redundancy, Irrelevance	Shadowing, Correlation, Generalization, Redundancy, Verbosity	Reachability, Cyclicity
Anomalies (inter)	Shadowing, Spuriousness, Redundancy, Correlation	Shadowing, Cross-Path, Blacklist, Whitelist	Reachability, Cyclicity

Table 1: Comparison of the different anomaly detection mechanisms.

realistic. Especially Cross-Path anomalies are serious security threats that are very challenging to find by hand. Also FIREMAN offers a better time complexity which is independent of the checking mode (intra or inter firewall). This advantage is bought by using a more complex model which creation may have exponential space complexity. Nonetheless, it is more expressive than the simple list model of Policy Advisor because it supports the full CIDR notation for addresses. To simulate this trait with Policy Advisor an exponential amount of rules would be necessary exposing its quadratic or even cubic complexity for the detection even more severly. With Kripke-Structures the approach of Jeffrey and Samak uses an even more complex but also the most natural model to represent firewall rules and networks. They have an exponential time complexity by proving the NP-completeness of their algorithm. Implicitly, this overall worst case complexity lays within Policy Advisor and FIREMAN as well but is hidden in the less expressive model and the exponential space complexity of BDD. Even though, Jeffrey and Samak have a rather limited amount of detected anomalies. Reachability is the counter part of shadowing. It shows that a certain rule is reachable within the network while shadowing proves the non-reachability. The second anomaly is cyclicity which is necessary to detect in the possibly cyclic environment. Also FIREMAN needs a cyclicity check to build its models. All relevant information about the algorithms is summarized in table 1. Jeffrey and Samak mention that the other approaches support more sophisticated anomalies and declare their integration into their algorithm as future work.

The expansion of the approach of Jeffrey and Samak with more interesting anomalies seems to be the most promising task. Especially since Policy Advisor is lacking a model that is expressive enough to meet real world issues. These are very likely to occur in bottom-up anomaly detection. FIREMAN on the other hand buys its speed and elegance with exponential space complexity. By moving the exponentiality into runtime and by using SAT-solvers as underlying technology Jeffrey and Samak theoretically gain benefit from the active development in that field of research. Also they provide the most natural model for firewalls and networks. FIREMAN uses some techniques to minimize the risks of exponential bloat. Finding similar optimizations for a SAT encoding will be the main challenge. Nevertheless there should be some synergies with research in the field of general model checking.

References

[1] Sheldon B. Akers. Binary decision diagrams. *IEEE Trans. Computers*, 27(6):509–516, 1978.

[2] Ehab Al-Shaer, Hazem H. Hamed, Raouf Boutaba, and M. Hasan. Conflict classification and analysis of distributed firewall policies. *IEEE Journal on Selected Areas in Communications*, 23(10):2069–2084, 2005.

[3] Ehab S. Al-Shaer and Hazem H. Hamed. Design and implementation of firewall policy advisor tools, 2002.

[4] Ehab S. Al-Shaer and Hazem H. Hamed. Firewall policy advisor for anomaly discovery and rule editing. In Germán S. Goldszmidt and Jürgen Schönwälder, editors, *Integrated Network Management*, volume 246 of *IFIP Conference Proceedings*, pages 17–30. Kluwer, 2003.

[5] Ehab S. Al-Shaer and Hazem H. Hamed. Discovery of policy anomalies in distributed firewalls. In *INFOCOM*, 2004.

[6] Ehab S. Al-Shaer and Hazem H. Hamed. Modeling and management of firewall policies. *IEEE Transactions on Network and Service Management*, 1(1):2–10, 2004.

[7] Henrik Reif Andersen. An Introduction to Binary Decision Diagrams - Lecture notes for Effictient Algorithms and Programs. Technical report, The IT University of Copenhagen, 1999.

[8] Alan Jeffrey and Taghrid Samak. Model checking firewall policy configurations. In *POLICY*, pages 60–67. IEEE Computer Society, 2009.

[9] Lihua Yuan, Jianning Mai, Zhendong Su, Hao Chen, Chen-Nee Chuah, and Prasant Mohapatra. Fireman: A toolkit for firewall modeling and analysis. In *IEEE Symposium on Security and Privacy*, pages 199–213. IEEE Computer Society, 2006.

A Appendix

	acceptance rules	
1.	$P_j \subseteq R_j$	good
2.	$P_j \cap R_j = \emptyset$	masked rule (error)
2a	$P_j \subseteq D_j$	shadowing
2b	$P_j \cap D_j = \emptyset$	redundancy
2c	else	redundancy and correlation
3.	$P_j \not\subseteq R_j$ and $P_j \cap R_j \neq \emptyset$	partially masked rule (warning)
3a	$P_j \cap D_j \neq \emptyset$	correlation
3b	$\forall x < j.\exists(P_x, deny).P_x \subseteq P_j$	generalization
3c	$P_j \cap A_j \neq \emptyset$ and $\forall x < j.\exists(P_x, accept).P_x \subseteq P_j$	redundancy
	deny rules	
4.	$P_j \subseteq R_j$	good
5.	$P_j \cap R_j = \emptyset$	masked rule (error)
5a	$P_j \subseteq A_j$	shadowing
5b	$P_j \cap A_j = \emptyset$	redundancy
5c	else	redundancy and correlation
6.	$P_j \not\subseteq R_j$ and $P_j \cap R_j \neq \emptyset$	partially masked rule (warning)
6a	$P_j \cap A_j \neq \emptyset$	correlation
6b	$\forall x < j.\exists(P_x, accept).P_x \subseteq P_j$	generalization
6c	$P_j \cap A_j \neq \emptyset$ and $\forall x < j.\exists(P_x, deny).P_x \subseteq P_j$	redundancy
	distributed acceptance rules	
	$P \subseteq I$	good
	$P \subseteq \neg I$	shadowing
	distributed deny rules	
	$P \subseteq I$	raised security level?
	$P \subseteq \neg I$	redundancy?
	$\forall j \in m.I_j = I$	cross-path consistency
	$I \cap blacklist \neq \emptyset$	policy violation
	$whitelist \not\subseteq I$	policy violation

Table 2: Formulae for the anomaly detection in FIREMAN. Note that formulae with letters are evaluated in conjunction with their base formula, which has the same index but no letter.