

# Certifying Spoofing-Protection of Firewalls

Cornelius Diekmann, Lukas Schwaighofer, and Georg Carle

Technische Universität München

Email: {diekmann|schwaighofer|carle}@net.in.tum.de

**Abstract**—We present an algorithm to certify IP spoofing protection of firewall rulesets. The algorithm is machine-verifyably proven sound and its use is demonstrated in real-world scenarios.

## I. INTRODUCTION

In firewalls, it is good practice and sometimes an essential security feature to prevent IP address spoofing attacks [1]–[6]. The Linux kernel offers reverse path filtering [7], which can conveniently prevent such attacks. However, in many scenarios (e.g. asymmetric routing [8], failover [3], zone-spanning interfaces, or multihoming [2]), reverse path filtering must be disabled or is too coarse-grained (e.g. for filtering special purpose addresses [9]). In these cases, spoofing protection has to be provided by the firewall which is configured by the administrator. Unfortunately, writing firewall rules is prone to human error, a fact known for over a decade [10].

For Linux netfilter/iptables firewalls [11], we present an algorithm to certify spoofing protection of a ruleset. It provides the following contributions; a unique set of features in their combination:

- It is formally and machine-verifyably proven sound with Isabelle/HOL.
- It supports the largest subset of iptables features compared to any other firewall analysis systems.
- It was tested on the largest (publicly-available) firewall which was ever analyzed in academia.
- It terminates within a second for thousands of rules.

We chose iptables because it provides one of the most complex firewall semantics widely deployed [12], [13]. Of course, our algorithm is also applicable to similar or less complex (e.g. Cisco PIX) firewall systems.

## II. RELATED WORK

There are several popular static firewall analysis tools. The Firewall Policy Advisor [14] discovers inconsistencies between pairs of rules (e.g. one rule completely overshadowing the other) in distributed firewall setups. A similar tool is FIREMAN [12]. It can discover inconsistencies within rulesets or between firewalls and verify setups against administrator-defined policies. To represent sets of packets, Binary Decision Diagrams (BDDs) are used. It does not support matching on interfaces in a rule. Since interfaces can be strings of arbitrary length, they may not be ideal for the encoding in BDDs and adding support to FIREMAN might be complicated or deteriorate its performance.

Margrave [15] can be used to query (distributed) firewalls. It is well suited to troubleshoot and to debug firewall configurations or to show the impact of ruleset edits. For certain queries,

it can also find scenarios, e.g. it can show concrete packets which violate a security policy. This scenario finding is sound but not complete. Its query language as well as Margrave’s internal implementation is based on first-order logic. A similar tool (relying on BDDs), with a different query language and focus on complete networks, is ConfigChecker [16], [17]. IT-Val [5] can also be used to query firewalls. It can also describe the firewall in terms of equivalent IP address spaces [18], which does not require the administrator to pose specific queries.

None of these tools can directly verify spoofing protection. Nor are the tools themselves formally verified, which limits confidence in their results. In addition, the tools only support a limited subset of real-world firewalls [19]. If the firewall under analysis exceeds this feature set, the tools either produce erroneous results or cannot continue [19].

Jeffrey and Samak [20] analyze the complexity of firewall analysis and conclude that most questions (for variable packet models) are NP-complete. They show that SAT solvers usually outperform BDD-based implementations.

## III. MATHEMATICAL BACKGROUND

This work was done completely in the Isabelle theorem prover [21]. Isabelle is an LCF-style theorem prover. This means, a fact can only be proven if it is accepted by a mathematical inference kernel. This kernel is very small and thoroughly inspected by the formal methods community. This makes errors very unlikely, which has been demonstrated by Isabelle’s success over the past 20 years.

All facts presented in this work are formally verified with Isabelle and the formalization is publicly available (cf. Section Availability). An interested reader can replay the proofs and results of the evaluation on her system. For brevity, we only outline a proof’s core idea in this paper and refer the interested reader to our proof document for further mathematical details.

Our notation is close to Isabelle, Standard ML, or Haskell: Function application is written without parentheses, e.g.  $f a$  denotes function  $f$  applied to parameter  $a$ . For lists, we denote *cons* and *append* by ‘::’ and ‘:::’, e.g. ‘ $x :: [y, z] ::: [a]$ ’. Linux shell commands are set in `typewriter` font.

*Iptables Semantics:* To define and prove correctness of our algorithm, one must first define the semantics (i.e. behavior) of an iptables firewall. We rely on the semantics specified by Diekmann et al. [19]. The semantics defines the following common actions (also called “targets” in iptables terminology): *Accept*, *Drop*, *Reject*, *Log*, *calling to* and *Returning from user-defined chains*, as well as the “empty” action. *Matching a packet against match conditions* (e.g. IP

source or destination addresses) is mathematically defined with a “magic oracle” which understands *all* possible matches.

Obviously, semantics with a “magic oracle” cannot be expressed in terms of executable code. Nevertheless, the algorithm we present in this paper is both executable and proven sound w.r.t. these semantics.

#### IV. SPOOFING PROTECTION – MATHEMATICALLY

To define spoofing protection, two data sets are required: The firewall ruleset  $rs$  and the IP addresses assignment  $ipassmt$ .  $ipassmt$  is a mapping from interfaces to an IP address range. Usually, it can be obtained by `ip route`. We will write  $ipassmt[i]$  to get the corresponding IP range of interface  $i$ . For the following examples, we assume

$$ipassmt = [eth0 \mapsto \{192.168.0.0/24\}]$$

**Definition 1** (Spoofing Protection). A firewall ruleset provides *spoofing protection* if for all interfaces  $i$  specified in  $ipassmt$ , all packets from  $i$  which are accepted by the ruleset have a source IP address contained in  $ipassmt[i]$ .

For example, using pseudo iptables syntax, the following ruleset implements spoofing protection:

```
-i eth0 --src !192.168.0.0/24 Drop
any Accept _____ firewall _____
```

*Spoofing Protection with Unknowns:* iptables supports numerous match conditions and new ones may be added in future. It is practically infeasible to support all match conditions in a tool [19]. As we don’t want our algorithm to abort if an unknown match occurs, we will refine Def. 1 to incorporate unknown matches. This is motivated by the following examples.

We assume that `--foo` is a match condition which is unknown to us. Therefore, we cannot guarantee that

```
-i eth0 --src !192.168.0.0/24 --foo Drop
any Accept _____ firewall _____
```

implements spoofing protections since `--foo` could prevent certain spoofed packets from being dropped. Also, the following ruleset might neither implement spoofing protection since `--foo` might allow spoofed packets:

```
--foo Allow
-i eth0 --src !192.168.0.0/24 Drop
any Accept _____ firewall _____
```

However, the following ruleset definitely implements spoofing protection; Independently of the meaning of `--foo` and `--bar`, it is guaranteed that no spoofed packets are allowed:

```
--foo Drop
-i eth0 --src !192.168.0.0/24 Drop
--bar Accept _____ firewall _____
```

This motivates Def. 2.

**Definition 2** (Certifiable Spoofing Protection). A firewall ruleset provides *spoofing protection* if for all interfaces  $i$  specified in  $ipassmt$ , all packets from  $i$  which are *potentially* accepted by the ruleset have a source IP address in the IP range of  $i$ .

This new definition is stricter than the original one: Def. 2 implies Def. 1.<sup>1</sup> Therefore the new definition is *sound* and can be used to prove that the last example implements spoofing protection.

However, depending on the meaning of `--foo`, some of the previous examples might also implement spoofing protection. This cannot be shown with Def. 2, so the new definition is not *complete*. However, as long as we anticipate unknown matches to occur, it is impossible to obtain completeness.

#### V. SPOOFING PROTECTION – EXECUTABLE

A straight forward spoofing protection proof of a firewall ruleset using Def. 2 would require iterating over all packets, which is obviously infeasible. We present an efficient executable algorithm to certify spoofing protection.

We assume the ruleset to be certified is preprocessed. For this, we rely on the semantics-preserving ruleset simplification [19], which rewrites a ruleset to a semantically equivalent ruleset where only *Accept* and *Drop* actions occur.<sup>2</sup> A preprocessed ruleset always has an explicit deny-all or allow-all rule at the end; it can never be empty. This rule corresponds to a chain’s default policy.

We call our algorithm  $sp$  (“spoofing protection”). It certifies spoofing protection for one interface  $i$  in  $ipassmt$ . Using  $sp$  to certify all  $i \in ipassmt$  for a ruleset implies spoofing protection according to Def. 2.

We assume the global, static, and fixed parameters of  $sp$  are an interface  $i$  and the  $ipassmt$ . Then,  $sp$  has the following type signature:

$$rule\ list \times ipaddr\ set \times ipaddr\ set \rightarrow \mathbb{B}$$

The first parameter (*rule list*) is the preprocessed firewall ruleset. A rule is a tuple  $(m, a)$ , where  $m$  is the match condition and  $a$  the action. The action is either *Accept* or *Drop*. For a packet  $p$ , there is a predicate *matches*  $m\ p$  which tells whether the packet  $p$  matches the match condition  $m$ .

The second parameter (*ipaddr set*) is the set of potentially allowed source IP addresses for  $i$ . The third parameter (*ipaddr set*) is the set of definitely denied source IP addresses for  $i$ .

The last parameter ( $\mathbb{B}$ ) is a Boolean, which is true if spoofing protection could be certified.

Before we present the algorithm, we first present its correctness theorem (which will be proven later).

**Theorem 1** ( $sp$  sound). For any ruleset  $rs$ , if

$$\forall i \in ipassmt. sp\ rs\ \{\}\ \{\}$$

then  $rs$  provides spoofing protection according to Def. 2.

<sup>1</sup>Formally proven; It follows from [19, Thm 3]

<sup>2</sup>The correctness of this preprocessing is also verified with Isabelle.

$$\begin{aligned}
\text{sp } [] A D &= (A \setminus D) \subseteq \bigcup ipassmt[i] \\
\text{sp } ((m, Accept) :: rs) A D &= \text{sp } rs (A \cup \{ip \mid \exists p \text{ from interface } i \text{ with src address } ip. \text{ matches } m p\}) D \\
\text{sp } ((m, Drop) :: rs) A D &= \text{sp } rs A (D \cup (\{ip \mid \forall p \text{ from interface } i \text{ with src address } ip. \text{ matches } m p\} \setminus A))
\end{aligned}$$

Figure 1. An algorithm to certify spoofing protection.

The algorithm  $\text{sp}$ , presented in Fig. 1, is implemented recursively. It iterates over the firewall ruleset.

The base case is for an empty ruleset. Here,  $A$  and  $D$  are the set of allowed/denied source IP addresses. The firewall provides spoofing protection if the set of potentially allowed sources minus the set of definitely denied sources is a subset of the allowed IP range.

The two recursive calls collect these sets  $A$  and  $D$ . If the rule is an *Accept* rule, the set  $A$  is extended with the set of sources possibly accepted in this rule. If the rule is a *Drop* rule, the set  $D$  is extended with the set of sources definitely denied in this rule, excluding any sources which were potentially accepted earlier.

As Theorem 1 already states,  $\text{sp}$  can be started with any ruleset and the empty set for  $A$  and  $D$ .

We will now describe how the *ipaddr set* operations are implemented. In general, we symbolically represent a set of IP addresses as a set of IP range intervals. Since IP ranges are commonly expressed in CIDR notation (e.g. *a.b.c.d/n*), the interval datatype proves to be very efficient.

Next, the set  $\{ip \mid \exists p \text{ from interface } i \text{ with src address } ip. \text{ matches } m p\}$  requires executable code. Obviously, a straightforward implementation which tests the existence of any packet is infeasible. We provide an over-approximation for this set, the correctness proof confirms that this approach is sound. First we check that  $i$  matches all input interfaces specified in the match expression  $m$ . If this is not the case, the set is obviously empty. Otherwise, we collect the intersection of all matches on source IPs in  $m$ . If no source IPs are specified in  $m$ , then  $m$  matches any source IP and we return the universe.

The set  $\{ip \mid \forall p \text{ from interface } i \text{ with src address } ip. \text{ matches } m p\}$  can be computed similarly. However, we need to return an under-approximation here. First, we check that  $i$  matches, otherwise the set is empty. Next, we remove all matches on input interfaces and source IPs from  $m$ . If the remaining match expression is not unconditionally true, then we return the empty set. Otherwise, we return the intersection of all source IP addresses specified in  $m$ , or the universe if  $m$  does not restrict source IPs.

Note that after preprocessing we always have an explicit allow-all or deny-all rule at the end of the firewall ruleset. Thus,  $A \cup D$  will always hold the universe after consuming the last rule.

## VI. EVALUATION – MATHEMATICALLY

We outline the main idea of the correctness proof. Since  $\text{sp}$  operates on a fixed interface, we define certifiable spoofing protection for a fixed interface  $i$ . Showing Def. 3 for all interfaces is equivalent to Def. 2.

### Definition 3.

$$\begin{aligned}
\forall p \in \{p \mid p \text{ from } i \text{ and potentially accepted by the firewall}\}. \\
p.\text{src-ip} \in \bigcup ipassmt[i]
\end{aligned}$$

The correctness proof of  $\text{sp}$  is done by induction over the firewall ruleset. Theorem 1 does not lend itself to induction, since it features two empty sets which would generate unusable induction hypotheses. To obtain a strong induction hypothesis, we generalize. The ruleset is split into two parts:  $rs_1$  and  $rs_2$ . We assume that the algorithm correctly iterated over  $rs_1$ . For this lemma, we use the following notation:

$$\begin{aligned}
A_{exact} &= \{ip \mid \exists p. p \text{ from } i \text{ with src } ip \text{ and accepted by } rs_1\} \\
D_{exact} &= \{ip \mid \forall p. p \text{ from } i \text{ with src } ip \text{ and denied by } rs_1\}
\end{aligned}$$

**Lemma 1.** If  $A_{exact} \subseteq A$  and  $D \subseteq D_{exact}$  and  $\text{sp } rs_2 A D$  then Def. 3 holds for  $rs_1 :: rs_2$

*Proof:* The proof is done by induction over  $rs_2$  for arbitrary  $rs_1$ ,  $A$ , and  $D$ .

Base case (i.e.  $rs_2 = []$ ): From  $\text{sp } [] A D$  we conclude  $(A \setminus D) \subseteq \bigcup ipassmt[i]$ . Since  $A$  is an over-approximation and  $D$  an under-approximation:  $A_{exact} \setminus D_{exact} \subseteq A \setminus D$ . Since no IP address can be both accepted and denied we get  $A_{exact} \setminus D_{exact} = A_{exact}$ . From transitivity we conclude  $A_{exact} \subseteq \bigcup ipassmt[i]$ , which implies spoofing protection for that interface according to Def. 3.

The two induction steps (one for *Accept* and one for *Drop* rules) follow from the induction hypothesis. The over- and under- approximations were carefully constructed, such that the subset relations continue to hold. The executable implementations of these sets also respect the subset relation; hence, the induction hypothesis solves these cases. ■

*Proof of Theorem 1:* Lemma 1 can be instantiated where  $rs_1$  is the empty ruleset and  $A$  and  $D$  are the the empty set. For this particular choice, it is easy to see that the preconditions hold. Thus, for any  $rs$ , we conclude  $\text{sp } rs \{\} \{\}$  implies Def. 3. Since this holds for arbitrary interfaces, we conclude Def. 2. ■

Thus, our algorithm is proven *sound* according to Def. 2. This means, if the algorithm certifies a ruleset, then this ruleset is guaranteed to implement spoofing protection.

Note that our algorithm only certifies; debugging a ruleset in case of a certification failure remains manual. To debug, the proof of Lemma 1 suggest to consider the first rule where  $(A \setminus D) \subseteq \bigcup ipassmt[i]$  is violated.

Standards such as Common Criteria [22] require formal verification for their highest *Evaluation Assurance Level* (EAL7), for example with Isabelle [22, §A.5]. Therefore, any ruleset certified by Isabelle to provide spoofing protection

could also be certified by Common Criteria EAL7. Since we provide an executable algorithm, this can be done *automatically* – without the need for a user with formal background or manual proof.

The algorithm is *not complete*. This means, there may be rulesets which implement spoofing protection but cannot be certified by the algorithm. This is bought by the approximations and by the support for unknown match conditions. For example, the following ruleset cannot be certified:

```
-i eth0 --src !192.168.0.0/24 --foo Drop
-i eth0 --src !192.168.0.0/24 --!foo Drop
any Accept
_____ firewall _____
```

This is a reasonable decision for a completely unknown `--foo`, since it might update an internal state and the mathematical equation “`foo ∨ !foo = True`” may not hold. However, if `foo` is replaced by the known and stateless match condition `--protocol tcp`, the ruleset can be shown to correctly implement spoofing protection. The algorithm, however, cannot certify it, since it does not track this match condition. However, this is a made-up and bad-practice example, and we never encountered such special cases in any real-world ruleset. The evaluation—in which vast amounts of unknowns occurred—shows that the algorithm certified all rulesets which included spoofing protection and correctly failed only for those rulesets which did not (correctly) implement it. Thus, the incompleteness is primarily a theoretical limitation.

## VII. EVALUATION – EMPIRICALLY

Often, firewalls start with an `ESTABLISHED` rule. A packet can only match this rule if it belongs to a connection which has been accepted by the firewall previously. Hence, the `ESTABLISHED` rule does not contribute to the access control policy for connection setup enforced by the firewall [12]. Likewise, spoofed packets can only be allowed by the `ESTABLISHED` rule if they are allowed by any of the subsequent `ACL` rules. Therefore, as done in previous work [19, §6.4], we either exclude this rule from our analysis or only consider packets of state `NEW`.

We tested the algorithm on several real-world rulesets (c.f. Section Availability). Most of them either did not provide spoofing protection or had an obvious spoofing protection and could thus be certified. In this Section, we present the results of certifying the largest and most interesting ruleset.

First of all, for all rulesets, our algorithm was extremely fast: Once the ruleset is preprocessed (few seconds) the certification algorithm only takes fractions of seconds for rulesets with several thousand rules. We omit a detailed performance evaluation since these orders of magnitude are sufficient for a static/offline analysis system.<sup>3</sup>

We present the certification of a firewall with about 4800 rules, connecting about 20 VLANs. Every VLAN has its own interface.

Trying the certification, it immediately fails. Responsible was a work-around rule which should only have existed

temporarily but was forgotten. This rule is now on the administrator’s “things to do the right way” list and we exclude it for further evaluation.

Certifying spoofing protection for the first VLAN interface succeeds instantly. However, trying to certify all other VLANs fails. The reason is an error in the ruleset. For every VLAN  $n$ , the firewall defines three custom chains: `mac_n`, `ranges_n`, and `filter_n`. The `mac_n` chain verifies that for hosts with registered MAC addresses and static IP addresses, nobody (with a different MAC address) steals the IP address. This chain is primarily to avoid manual IP assignment errors. Next, the `ranges_n` chain should prevent outgoing spoofing. Finally, the `filter_n` chain allows packets for certain registered services. The main error was that a spoofed packet from VLAN  $m$  could be accepted by `filter_n` before it had to pass the `ranges_m` check. The discovery of this error also discovered that the `mac_m` chains were not working reliably. We verified these findings by sending and receiving such spoofed packet via the real firewall.

Finally, we fixed the firewall by moving all `mac_n` and `ranges_n` chains before any `filter_n` chains. The certification for all but one<sup>4</sup> internal VLAN interfaces succeeds.

Next, the interfaces attached to the Internet are certified. The IP address range was defined as the universe of all IPs, excluding the IPs owned by the institute. Here, certification failed in a first run. Responsible were some `ssh` rate limiting rules. These rules were originally designed to prevent too many outgoing `ssh` connections. However, since spoofing protection did not apply to them, an attacker could exploit them for a DOS attack against the internal network: The attacker floods the firewall with `ssh` TCP SYN packets with spoofed internal addresses. This exhausts the `ssh` limit of the internal hosts and it is no longer possible for them to establish new `ssh` connections. This flaw was fixed and certification subsequently succeeds.

The improved and certified firewall ruleset is now in production use.

After this, another administrator got interested and wanted to implement spoofing protection for his firewall. To complicate matters, he was in Japan and the firewall in Germany. It was a key requirement that he would not lock himself out. Our tool could certify both: His proposed changes to the firewall correctly enforce spoofing protection and he will not lose `ssh` access. To provide a *sound* guarantee for the latter, we applied the same idea as in our algorithm, but in reverse: the ruleset is abstracted to a stricter version (i.e. a version that blocks more packets) and we consequently certify that it still allows `NEW` and `ESTABLISHED` `ssh` packets from the Internet.

## VIII. CONCLUSION

We present an easy-to-use algorithm. It is fast enough to be run on every ruleset update. It discovered real problems in a large, production-use firewall. Both, the theoretical algorithm as well as the executable code are proven sound, hence if the algorithm certifies a firewall, the ruleset is *proven* to implement spoofing protection correctly.

<sup>3</sup>Certification runs of our algorithm were usually faster than reloading the ruleset on the firewall system itself.

<sup>4</sup>This one VLAN where the certification fails is only for internal testing purposes and deliberately features no spoofing protection

## REFERENCES

- [1] P. Ferguson and D. Senie, “Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing,” RFC 2827 (Best Current Practice), Internet Engineering Task Force, May 2000, updated by RFC 3704. [Online]. Available: <http://www.ietf.org/rfc/rfc2827.txt>
- [2] F. Baker and P. Savola, “Ingress Filtering for Multihomed Networks,” RFC 3704 (Best Current Practice), Internet Engineering Task Force, Mar. 2004. [Online]. Available: <http://www.ietf.org/rfc/rfc3704.txt>
- [3] T. J. Wagner, “Disabling reverse-path filtering in complex networks,” Jul. 2009, retrieved Mai 2015. [Online]. Available: <https://www.tolaris.com/2009/07/13/disabling-reverse-path-filtering-in-complex-networks/>
- [4] Y. Bartal, A. Mayer, K. Nissim, and A. Wool, “Firmato: A novel firewall management toolkit,” in *IEEE Symposium on Security and Privacy*. IEEE, 1999, pp. 17–31.
- [5] R. M. Marmorstein and P. Kearns, “A tool for automated iptables firewall analysis.” in *USENIX Annual Technical Conference, FREENIX Track*, 2005, pp. 71–81.
- [6] A. Wool, “The use and usability of direction-based filtering in firewalls,” *Computers & Security*, vol. 23, no. 6, pp. 459–468, 2004.
- [7] “ip-sysctl,” Linux Kernel 4.0 Documentation, 2015. [Online]. Available: [\\$Linux/Documentation/networking/ip-sysctl.txt](#)
- [8] J. Engelhardt, “Towards the perfect ruleset,” May 2011. [Online]. Available: [http://inai.de/documents/Perfect\\_Ruleset.pdf](http://inai.de/documents/Perfect_Ruleset.pdf)
- [9] M. Cotton, L. Vegoda, R. Bonica, and B. Haberman, “Special-Purpose IP Address Registries,” RFC 6890 (Best Current Practice), Internet Engineering Task Force, Apr. 2013. [Online]. Available: <http://www.ietf.org/rfc/rfc6890.txt>
- [10] A. Wool, “A quantitative study of firewall configuration errors,” *Computer*, IEEE, vol. 37, no. 6, pp. 62 – 67, 6 2004.
- [11] T. netfilter.org project, “netfilter/iptables project.” [Online]. Available: <http://www.netfilter.org/>
- [12] L. Yuan, H. Chen, J. Mai, C.-N. Chuah, Z. Su, and P. Mohapatra, “FIREMAN: a toolkit for firewall modeling and analysis,” in *IEEE Symposium on Security and Privacy*, May 2006, pp. 199–213.
- [13] S. Pozo, R. Ceballos, and R. M. Gasca, “Model-based development of firewall rule sets: Diagnosing model inconsistencies,” *Information and Software Technology*, vol. 51, no. 5, pp. 894–915, 2009.
- [14] E. Al-Shaer and H. Hamed, “Discovery of policy anomalies in distributed firewalls,” in *INFOCOM 2004. Twenty-third Annual Joint Conference of the IEEE Computer and Communications Societies*, vol. 4, March 2004, pp. 2605–2616 vol.4.
- [15] T. Nelson, C. Barratt, D. J. Dougherty, K. Fisler, and S. Krishnamurthi, “The margrave tool for firewall analysis.” in *24th USENIX Large Installation System Administration Conference (LISA)*, 2010.
- [16] E. Al-Shaer and M. Alsaleh, “ConfigChecker: A tool for comprehensive security configuration analytics,” in *Configuration Analytics and Automation (SAFECONFIG)*, 2011 4th Symposium on, Oct 2011, pp. 1–2.
- [17] E. Al-Shaer, W. Marrero, A. El-Atawy, and K. Elbadawi, “Network configuration in a box: towards end-to-end verification of network reachability and security,” in *17th IEEE International Conference on Network Protocols, (ICNP)*, Oct 2009, pp. 123–132.
- [18] R. M. Marmorstein and P. Kearns, “Firewall analysis with policy-based host classification.” in *Large Installation System Administration Conference (LISA)*, vol. 6. USENIX, Dec. 2006, pp. 41–51.
- [19] C. Diekmann, L. Hupel, and G. Carle, “Semantics-Preserving Simplification of Real-World Firewall Rule Sets,” in *20th International Symposium on Formal Methods (FM)*, Jun. 2015.
- [20] A. Jeffrey and T. Samak, “Model checking firewall policy configurations,” in *Policies for Distributed Systems and Networks*. IEEE, Jul. 2009, pp. 60–67.
- [21] T. Nipkow, L. C. Paulson, and M. Wenzel, *Isabelle/HOL: A Proof Assistant for Higher-Order Logic*, ser. LNCS. Springer, 2002, last updated 2015, vol. 2283.
- [22] Common Criteria, “Security assurance components,” *Common Criteria for Information Technology Security Evaluation*, vol. CCMB-2012-09-003, Sep. 2012. [Online]. Available: <http://www.commoncriteriaportal.org/files/ccfiles/CCPART3V3.1R4.pdf>

## AVAILABILITY

The analyzed firewall rulesets can be found at

<https://github.com/diekmann/net-network>

Our Isabelle formalization can be obtained from

[https://github.com/diekmann/Iptables\\_Semantics](https://github.com/diekmann/Iptables_Semantics)

## ACKNOWLEDGMENTS

Andreas Korsten deserves our sincerest thanks for keeping our infrastructure running. More than 20 VLANs, transfer networks, an AS, several testbeds, Internet-wide scanning, hundreds of publicly accessible VMs randomly spawned on the fly, and probably thousands of special cases and exceptions. This leads to an extremely complex infrastructure, all managed by one person; nevertheless he offers the time, commitment, and responsibility to deploy our newest research innovations on the live system.

Julius Michaelis implemented and proved large parts of the datatype to efficiently handle IP address ranges as intervals.

This work has been supported by the German Federal Ministry of Education, EUREKA project SASER, grant 16BP12304, and project SURF, grant 16KIS0145, and by the European Commission, project SafeCloud, grant 653884.