Checking the Expressivity of Firewall Languages

Lorenzo Ceragioli[1](✉), Pierpaolo Degano[1], and Letterio Galletta[2]

[1] Dipartimento di Informatica, Università di Pisa, Pisa, Italy
lorenzo.ceragioli@phd.unipi.it, degano@di.unipi.it
[2] IMT School for Advanced Studies, Lucca, Italy
letterio.galletta@imtlucca.it

Abstract. Designing and maintaining firewall configurations is hard, also for expert system administrators. Indeed, policies are made of a large number of rules and are written in low-level configuration languages that are specific to the firewall system in use. As part of a larger group, we have addressed these issues and have proposed a semantic-based transcompilation pipeline. It is supported by FWS, a tool that analyses a real configuration and ports it from a firewall system to another. To our surprise, we discovered that some configurations expressed in a real firewall system cannot be ported to another system, preserving the semantics. Here we outline the main reasons for the detected differences between the firewall languages, and describe F2F, a tool that checks if a given configuration in a system can be ported to another system, and reports its user on which parts cause problems and why.

1 Introduction

Firewalls are the basic mechanisms for the protection of computer networks. Their effectiveness heavily depends on the correctness of configurations, since even a small flaw may break up completely the security or the functionality of an entire network.

Policies are typically written in low-level configuration languages that are specific to the firewall system in use and support non-trivial control flow constructs, such as *call* and *goto*. A configuration usually consists of a large number of rules interacting with each other, often in some obscure manner. Indeed, the way rules are used depends on the firewall system in hand, and also on contextual information, e.g. the order in which rules appear in the configuration. Also, the way in which these systems work has to be inferred from manuals and by experiencing with them, since almost always they have no formal semantics. It is therefore hard to understand the firewall behaviour, also because of the different ways in which packets are processed by the network stack of the operating system

The first two authors have been partially supported by project PRA_2018_66 *DECLware: Declarative methodologies for designing and deploying applications* of the Università di Pisa; the third author by IMT project *PAI VeriOSS*.

running on the firewall. An additional source of mess is Network Address Translation (NAT) that translates IP addresses and performs port redirection while packets traverse the firewall. In cooperation with a larger group of researchers, we have proposed a semantic-based transcompilation pipeline [3,4] to support system administrators. The pipeline is implemented by the tool FWS, described in [4]. It is available online[1] and it provides several utilities. Its first stages support a system administrator in analysing the current firewall configuration, by first extracting a formal description of the firewall behaviour. Then, the user can port a configuration from a system to another, preserving its meaning. Crucial to all the pipeline is the use of the Intermediate Firewall Configuration Language (IFCL) that has a formal semantics. This intermediate language has been used in [3,4] as a common framework to encode firewalls written in `iptables` [12], `ipfw` [13] and `pf` [11], which are the most used languages in Unix and Linux.

To our surprise, while implementing the pipeline, we discovered that some configurations could not be compiled to a language different from the source one. To investigate on these corner cases, we used the new denotational semantics of IFCL and the new algorithm for porting configurations given in [6]. The expressivity of real firewall systems is then formally compared, and we proved that their languages originate a partial order [5]. In particular there exists a firewall expressible in `iptables`, but neither in `ipfw` nor in `pf`; also, `iptables` is incomparable and `ipfw` dominates `pf`.

Here, we describe F2F, a tool that given a configuration c for a source system $\mathcal{L}$ and a target system $\mathcal{L}'$, checks if there exists a configuration c' in $\mathcal{L}'$ such that the behaviour is the same. If instead there is none, it reports to the user on which packets cause problems and why. The information provided by F2F can guide the system administrator in the choice of another target system, but could also be used to make an informed decision on how to change the configuration in order to make it expressible by the target system. We also present a simple yet realistic use case for F2F, that is available online[2] and takes a few seconds to perform such a check on medium size configurations, although it is in a preliminary version.

Our purpose here is to complement the papers [3–6] that have the full details about the theoretical basis of the tool; in particular, we refer the interested reader to [5,6] for the new denotational semantics of IFCL and for an analysis on the expressive power of various firewall languages.

Related Work. To the best of our knowledge the expressive power of firewall systems is formally investigate only in [5], which provides us with firm guidelines for the development of F2F.

In the literature there are many proposals for modeling and analyzing firewall configurations. Some of them, e.g. see [7,9,10], do not rely on a formal semantics and typically either compile from a high level to a low level language or check if configurations comply with given specifications.

[1] https://github.com/secgroup/fws.
[2] https://github.com/lceragioli/F2F.

The following instead provide firewall systems with a formal semantics. Diekmann et al. [8] consider a subset of `iptables` without packet transformations, and mechanize it using Isabelle/HOL. Furthermore, they define and prove correct a simplification procedure that aims to make configurations easier to be analyzed by automatic tools. Adão et al. [1] introduce Mignis, a high level firewall language for specifying abstract filtering policies, and a compiler from it into `iptables`, which is formally proved correct. The paper formalizes the semantics of Mignis and gives an operational semantics of `iptables`, including packet filtering and translations. Successively, Adão et al. [2] propose a denotational semantics for Mignis that maps a configuration into a packet transformer, representing all the accepted packets with the corresponding translations, similarly to the one in [5], which is the basis of our tool.

Plan of the Paper. Section 2 surveys the firewall system we consider in this paper, the IFCL semantics and the transcompilation pipeline of [4].

Section 3 presents an `iptables` configuration, implementing a policy to be applied in a simple yet realistic scenario, and shows how our tool detects that neither `ipfw` nor `pf` can express this configuration. The internals of the tool F2F are described in Sect. 4 with the help of some examples that illustrate the main reasons for the differences between the expressive power of the considered firewall languages. In Sect. 5 we conclude and discuss some future work.

2 Background

In the following we survey the most used firewall languages, the intermediate language IFCL and the way it is used to encode them. Finally, we briefly present the stages of the transcompilation pipeline.

2.1 `iptables`

It is the default tool for packet filtering in Linux and it operates on top of Netfilter, the framework for packets processing of the Linux kernel [12].

An `iptables` configuration is built on *tables* and *chains*. Intuitively, each table has a specific purpose and is made of a collection of chains. The most commonly used tables are: `Filter` for packet filtering; `Nat` for network address translation; `Mangle` for packet alteration. Chains are actually rulesets, and are ordered lists of policy rules that are inspected to find the first one matching the packet under evaluation. There are five predefined chains, and the user can extend them defining its own. Chains are inspected by the Linux kernel at specific moments of the packet life cycle [14]: `PreRouting`, when a packet reaches the host; `Forward`, when a packet is routed through the host; `PostRouting`, when a packet is about to leave the host; `Input`, when a packet is routed to the host; `Output`, when a packet is generated by the host.

Each rule specifies a *condition* and an action to apply to the matching packet, called *target*. The most commonly used targets are: ACCEPT, to accept the packet;

DROP, to discard the packet; DNAT, to perform destination NAT, i.e., a translation of the destination address; SNAT, to perform source NAT, i.e., a translation of the source address. There are also targets that allow implementing mechanisms similar to jumps and procedure calls, but since they can be macro-expanded (see below), we ignore them and refer the reader to the technical documentation [12]. Built-in chains have a user-configurable default policy (ACCEPT or DROP) to be applied when the evaluation reaches the end of a built-in chain.

2.2 ipfw

It is the standard firewall for FreeBSD [13]. A configuration consists of a single list of rules (called ruleset) that is inspected twice, when the packet enters the firewall and when it exits. It is possible to specify when a certain rule has only to be applied in either case by using the keywords `in` and `out`. Similarly to `iptables`, rules are inspected sequentially until the first match occurs and the corresponding action is taken. The packet is dropped if there is no matching rule. The sequential order of inspection is altered by special targets that jump to a rule that follows the current one.

2.3 pf

This is the standard firewall of OpenBSD [11] and MacOS since version 10.7. A `pf` configuration consists of a single ruleset, inspected when the packet enters and exits the host. Similarly to the other systems, each rule consists of a condition and a target that specifies how to process the packets matching the condition. The most common targets are: `pass` and `block` to accept and reject packets, respectively; `rdr` and `nat` to perform destination and source NAT, respectively. The rule to apply is the *last matched rule* (unless otherwise specified). Moreover, when a packet enters the host, DNAT rules are examined first and filtering is performed after the address translation. Similarly, when a packet leaves the host, its source address is translated by the relevant SNAT rules, and then the resulting packet is filtered.

2.4 The Intermediate Language IFCL

We use the Intermediate Firewall Configuration Language (IFCL) [3,4] as a common setting in which we encode the firewall systems above. It has been originally equipped with an operational semantics [3].

A firewall configuration in IFCL consists of a set of rulesets and a control diagram. As usual, a ruleset is a list of rules that are inspected sequentially to find the first rule matching a packet. A *control diagram* is a graph C that deterministically describes the order in which rulesets are applied by the network stack of the operating system. Every node q represents thus a processing phase and it is associated with the ruleset to apply when the control reaches q. Arcs are labeled with predicates that encode routing decisions taken by the firewall, e.g., depending on whether a packet comes from the external world. Intuitively, a packet p is accepted if there exists a path π from the initial to the final node of C such that p passes the checks of (and is possibly transformed by) the rulesets associated with the nodes of π.

A firewall rule consists of a predicate ϕ over packets and an action t, called *target*, defining how packets matching ϕ are processed. For our purposes, it suffices to consider the following subset of targets considered in [4] (note that we can safely neglect targets altering the control, like *call*, *goto* and *return*, because they can suitably be macro-expanded)

ACCEPT	the packet is accepted
DROP	the packet is discarded
$\texttt{NAT}(n_d, n_s)$	apply NAT

In the NAT action, n_d and n_s specify how to translate the destination and source addresses/ports of a packet and, as done before, we use $-$ to denote the identity translation. For instance, $n_d = n : -$ means that the destination address of a packet is translated according to n, while the port is kept unchanged.

Formally, an IFCL firewall is defined as follows:

Definition 1 (Firewall). *An IFCL* firewall *is a pair (C, Σ) where C is a control diagram and Σ is a configuration, defined below.*

Let $\mathcal{P} \subseteq \mathbb{P}$ be a set of predicates over packets p. A deterministic control diagram *C is a tuple (Q, A, q_i, q_f), where*

- *Q is the set of nodes;*
- *$A \subseteq Q \times \mathcal{P} \times Q$ is the set of arcs, such that whenever $(q, \psi, q'), (q, \psi', q'') \in A$ and $q' \neq q''$ then $\neg(\psi \wedge \psi')$;*
- *q_i, $q_f \in Q$ are special nodes denoting the start of elaboration of an incoming packet p and the end if p is accepted.*

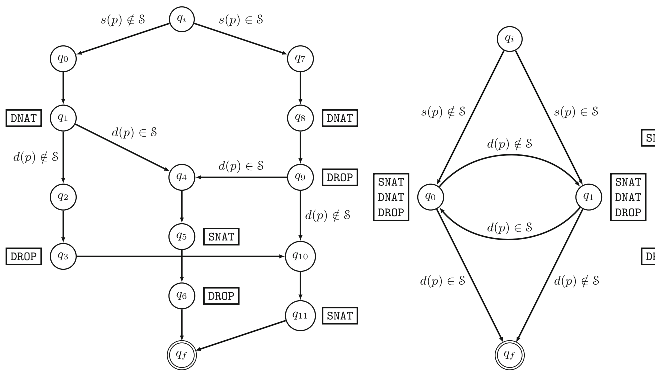
A configuration *is a pair $\Sigma = (\rho, c)$, where*

- *ρ a set of rulesets;*
- *$c\colon Q \to \rho$ is a function assigning a ruleset to each node $q \in Q$.*

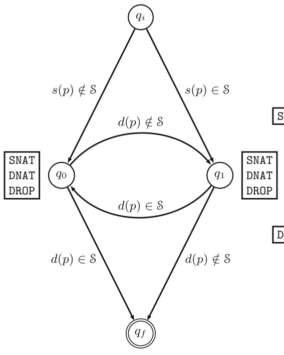
2.5 Modeling `iptables`, `ipfw` and `pf` in IFCL

We now survey the encoding of the firewall systems mentioned above into IFCL. The full definitions are in [3,4], and we only report here the relevant information for our treatment. We remark that the behaviour of those firewall systems is only informally defined by manuals, except for `iptables` for which [1] introduced a formal semantics. However they all inherit the formal semantics of IFCL, via the encodings below (for `iptables` the inherited semantics and that of [1] coincide).

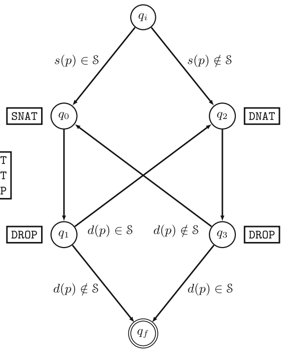
Hereafter let $\mathcal{S}$ be the set of the addresses of the firewall interfaces; let $p \in \mathbb{P}$ be a packet; let $d(p)$ and $s(p)$ denote the destination address and source address



(a) `iptables`



(b) `ipfw`



(c) `pf`

Fig. 1. Control diagrams of `iptables`, `ipfw` and `pf`.

of p, respectively; and let $d(p) \in \mathbb{S}$ ($s(p) \in \mathbb{S}$, respectively) specify that p is for (comes from, respectively) the firewall. In the control diagrams of Fig. 1 we label arcs with predicates expressing constraints on the header of packets according to $\mathbb{S}$; arcs with no label implicitly carry "*true*".

iptables. Figure 1a shows the control diagram $C_{\texttt{iptables}}$ of `iptables` (recall that its tables contain predefined chains). The encoding into IFCL associates these predefined chains with the nodes of the control diagram. For example, the table `Nat` contains the `PostRouting` chain that is associated with q_{11}. It is important to note that in `iptables` a `DNAT` is only performed in nodes q_1, q_8, whereas `SNAT` only in nodes q_5, q_{11}. Similarly, `DROP` can only be applied when the control is in either nodes q_3, q_6, q_9. As we will see later on, these capabilities are represented by the labels in the boxes.

ipfw. The control diagram of `ipfw` is in Fig. 1b. The encoding of [4] splits the single `ipfw` ruleset in two rulesets containing each the rules annotated with the keyword `in` and `out`, respectively (if not annotated, the rule goes in both). Both rulesets can filter packets and transform them. The node q_0 is associated with the ruleset applied when an IP packet reaches the host from the net, whereas, q_1 is for when the packet leaves the host. Note that the loop between the nodes q_0 and q_1 causes no problem, because firewalls detects cycles and drop the packet causing it.

pf. Figure 1c displays the control diagram of `pf`, where the nodes q_2 and q_3 are associated with the rulesets applied to packets that reach the firewall, while q_0 and q_1 are for when they leave the firewall. Also in this case, the single ruleset of `pf` is split in different rulesets. A source NAT can only be applied to packets leaving the firewall, and destination NAT only those reaching it. Importantly NAT rules (in nodes q_0 and q_2) are evaluated *before* filtering (in nodes q_1 and q_3). To represent the *last matching-rule* policy of `pf`, it suffices to reverse the order of rules inside rulesets.

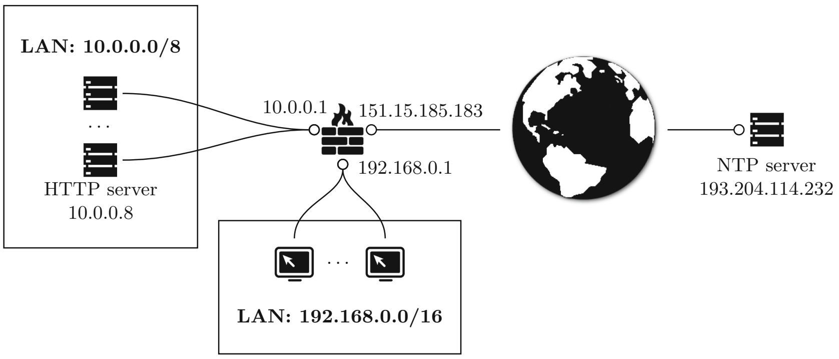


Fig. 2. A simple scenario.

2.6 Transcompilation Pipeline

The transcompilation pipeline of [3,4] supports system administrators in (i) *decompiling* real configurations into abstract specifications representing the set of the permitted connections; (ii) performing *policy refactoring* by supporting configuration updates and elimination of redundant rules, thus obtaining minimal and clean configurations; (iii) automatically *porting* a configuration written for a system into the language of another system.

The proposed transcompiling pipeline is made of the following stages:

1. decompile a policy from the source language to IFCL;
2. extract the meaning of the policy as a table describing how the accepted packets are translated;
3. compile the semantic table into the target language.

All the steps above are supported by the tool FWS, described in [4] and available online.[3] Experiments have been made on te languages mentioned above.

3 An Example Illustrating the Expressivity Problem

We present below a specific network with a firewall, and a specific configuration, expressed in `iptables`. Then, using F2F we illustrate the reasons why this configuration can neither be ported in `ipfw` nor in `pf`, without resorting to ad hoc extensions of these languages.

Consider the firewall connected to the Internet using the IP 151.15.185.183, in the network in Fig. 2. It protects two LANs with addresses ranging in 10.0.0.0/8 and 192.168.0.0/16,[4] respectively, because all the connections from and to the Internet pass through the firewall. The first LAN hosts various servers, and the

[3] https://github.com/secgroup/fws.
[4] We use the standard CIDR notation to denote the range of IP addresses.

```
*nat
:PREROUTING ACCEPT [0:0]
:INPUT ACCEPT [0:0]
:OUTPUT ACCEPT [0:0]
:POSTROUTING ACCEPT [0:0]

-A PREROUTING -p udp --dport 123 -j DNAT --to 193.204.114.232
-A OUTPUT -p udp --dport 123 -j DNAT --to 193.204.114.232
-A PREROUTING -p tcp -d 151.15.185.183 --dport 80 -j DNAT --to 10.0.0.8
-A OUTPUT -p tcp -d 151.15.185.183 --dport 80 -j DNAT --to 10.0.0.8

-A POSTROUTING -d 192.168.0.0/16 -j ACCEPT
-A INPUT -d 192.168.0.0/16 -j ACCEPT
-A POSTROUTING -d 10.0.0.0/8 -j ACCEPT
-A INPUT -d 10.0.0.0/8 -j ACCEPT
-A POSTROUTING -j SNAT --to 151.15.185.183
-A INPUT -j SNAT --to 151.15.185.183

COMMIT

*filter
:INPUT DROP [0:0]
:FORWARD DROP [0:0]
:OUTPUT DROP [0:0]

-A INPUT -m state --state ESTABLISHED -j ACCEPT
-A INPUT -p tcp -d 10.0.0.8 --dport 80 -j ACCEPT
-A INPUT -s 10.0.0.0/8 -d 10.0.0.0/8 -j ACCEPT
-A INPUT -s 192.168.0.0/16 ! -d 10.0.0.0/8 -j ACCEPT
-A INPUT -p udp -d 193.204.114.232 --dport 123 -j ACCEPT

-A FORWARD -m state --state ESTABLISHED -j ACCEPT
-A FORWARD -p tcp -d 10.0.0.8 --dport 80 -j ACCEPT
-A FORWARD -s 10.0.0.0/8 -d 10.0.0.0/8 -j ACCEPT
-A FORWARD -s 192.168.0.0/16 ! -d 10.0.0.0/8 -j ACCEPT
-A FORWARD -p udp -d 193.204.114.232 --dport 123 -j ACCEPT

-A OUTPUT -m state --state ESTABLISHED -j ACCEPT
-A OUTPUT -p tcp -d 10.0.0.8 --dport 80 -j ACCEPT
-A OUTPUT -s 10.0.0.0/8 -d 10.0.0.0/8 -j ACCEPT
-A OUTPUT -s 192.168.0.0/16 ! -d 10.0.0.0/8 -j ACCEPT
-A OUTPUT -p udp -d 193.204.114.232 --dport 123 -j ACCEPT

COMMIT
```

Fig. 3. A simple configuration in `iptables`.

second one common users. The firewall interacts with the LANs through the interfaces with the addresses 10.0.0.1 and 192.168.0.1, respectively.

We assume that hosts in the LANs have private addresses that cannot be routed on the Internet, hence the source address of outgoing packets and the destination address of incoming packets are translated using NAT. Also these hosts are not allowed to directly communicate with each other, and thus their messages always pass through the firewall.

The configuration in Fig. 3 enforces the following behaviour. The hosts in the LANs can freely communicate each other, but servers cannot start a connection towards a common user. When the protocol is NTP (Network Time Protocol) on port 123, the connection is redirected to the remote host 193.204.114.232, via a destination NAT (`DNAT`). The connections from the Internet towards the hosts in

```
(venv) user@here:~/$ fwp iptables ~/interfaces ~/iptables.conf ipfw
Solving: [##################################################] (   36/   36) 100.00%

PROBLEM FOUND!
In ipfw the following rule schema is not expressible!
==============================================================================
|   sIp   |   dIp   ||    tr_sIp     :  tr_sPort |    tr_dIp     :  tr_dPort |
==============================================================================
|  Self   |  Self   || SNAT ( Self)  : id        | DNAT (~Self)  : id        |
==============================================================================
Hence the following is impossible to achieve:
===================================================================================================
||    sIp     | sPort |      dIp       | dPort | prot ||      tr_src       |        tr_dst       ||
===================================================================================================
|| 192.168.0.1 |   *   |   127.0.0.1    |  123  | udp  || 151.15.185.183 : - | 193.204.114.232 : - ||
||            |       | 151.15.185.183 |       |      ||                   |                     ||
||            |       |    10.0.0.1    |       |      ||                   |                     ||
||            |       |  192.168.0.1   |       |      ||                   |                     ||
===================================================================================================

PROBLEM FOUND!
In ipfw the following rule schema is not expressible!
==============================================================================
|   sIp   |   dIp   ||    tr_sIp    :  tr_sPort  |    tr_dIp    :  tr_dPort  |
==============================================================================
|  Self   | ~Self   || SNAT ( Self) : id         | DNAT (~Self) : id         |
==============================================================================
Hence the following is impossible to achieve:
==================================================================================================================
||    sIp     | sPort |              dIp              | dPort | prot ||      tr_src       |        tr_dst       ||
==================================================================================================================
|| 192.168.0.1 |   *   |       0.0.0.0 - 10.0.0.0       |  123  | udp  || 151.15.185.183 : - | 193.204.114.232 : - ||
||            |       |      10.0.0.2 - 127.0.0.0      |       |      ||                   |                     ||
||            |       |   127.0.0.2 - 151.15.185.182  |       |      ||                   |                     ||
||            |       |  151.15.185.184 - 192.168.0.0 |       |      ||                   |                     ||
||            |       | 192.168.0.2 - 255.255.255.255 |       |      ||                   |                     ||
==================================================================================================================
```

Fig. 4. Checking the portability of the configuration in `ipfw`.

the LANs are blocked, except than those to the external address 151.15.185.183 on port 80 (written 151.15.185.183 : 80), that are redirected to the HTTP server at 10.0.0.8, also via `DNAT`. Every host can connect to 193.204.114.232, while only common users can connect to other Internet addresses. The source address of these outgoing packets get their source address translated to the external address of the firewall, via a `SNAT`.

Assume we want to port this configuration into `ipfw`, and for that we invoke F2F to check if this is possible. Intuitively, the configuration is internally represented as a table whose rows represent the accepted packets and how they are translated. For example, consider a UDP packet with destination 151.15.185.184 : 123 and source 192.168.0.1 : $*$ ($*$ stands for any port). It is accepted when its destination is translated to 193.204.114.232 : 123 and its source to 151.15.185.183 : $-$ ($-$ stands for "the port remains the same"), because its destination is first translated by rule 10, then it passes through the `OUTPUT` chain in the *filter* table and is forwarded by rule 42, finally it is subject to `SNAT` in rule 16 that is the first matching rule of the `POSTROUTING` chain in *nat* table. Roughly, the relevant portion of the corresponding row will look as follows:

`sIp`	`sPort`	`dIp`	`dPort`	`prot`	`+tr_scr+`	`+tr_dst+`
192.168.0.1	$*$	151.15.185.184	123	`udp`	151.15.185.183 : $-$	193.204.114.232:123

```
(venv) user@here:~/$ fwp iptables ~/interfaces ~/iptables.conf pf

PROBLEM FOUND!
In pf the following rule schema is not expressible!
============================================================================
|  sIp   |  dIp   ||    tr_sIp    :  tr_sPort  |    tr_dIp    :  tr_dPort  |
============================================================================
|  Self  |  Self  || id           : id         | DNAT (~Self) : id         |
============================================================================
Hence the following is impossible to achieve:
=====================================================================================
||      sIp       | sPort |      dIp       | dPort | prot || tr_src |    tr_dst    ||
=====================================================================================
||   127.0.0.1    |   *   | 151.15.185.183 |   80  | tcp  || - : -  | 10.0.0.8 : - ||
|| 151.15.185.183 |       |                |       |      ||        |              ||
||    10.0.0.1    |       |                |       |      ||        |              ||
||  192.168.0.1   |       |                |       |      ||        |              ||
=====================================================================================

PROBLEM FOUND!
In pf the following rule schema is not expressible!
============================================================================
|  sIp   |  dIp   ||    tr_sIp    :  tr_sPort  |    tr_dIp    :  tr_dPort  |
============================================================================
|  Self  |  Self  || SNAT ( Self) : id         | DNAT (~Self) : id         |
============================================================================
Hence the following is impossible to achieve:
=====================================================================================================
||     sIp     | sPort |      dIp       | dPort | prot ||       tr_src       |        tr_dst       ||
=====================================================================================================
|| 192.168.0.1 |   *   |   127.0.0.1    |  123  | udp  || 151.15.185.183 : - | 193.204.114.232 : - ||
||             |       | 151.15.185.183 |       |      ||                    |                     ||
||             |       |    10.0.0.1    |       |      ||                    |                     ||
||             |       |  192.168.0.1   |       |      ||                    |                     ||
=====================================================================================================

PROBLEM FOUND!
In pf the following rule schema is not expressible!
============================================================================
|  sIp   |  dIp   ||    tr_sIp    :  tr_sPort  |    tr_dIp    :  tr_dPort  |
============================================================================
|  Self  | ~Self  || SNAT ( Self) : id         | DNAT (~Self) : id         |
============================================================================
Hence the following is impossible to achieve:
====================================================================================================================
||     sIp     | sPort |              dIp              | dPort | prot ||       tr_src       |        tr_dst       ||
====================================================================================================================
|| 192.168.0.1 |   *   |       0.0.0.0 - 10.0.0.0      |  123  | udp  || 151.15.185.183 : - | 193.204.114.232 : - ||
||             |       |      10.0.0.2 - 127.0.0.0     |       |      ||                    |                     ||
||             |       |   127.0.0.2 - 151.15.185.182  |       |      ||                    |                     ||
||             |       |  151.15.185.184 - 192.168.0.0 |       |      ||                    |                     ||
||             |       | 192.168.0.2 - 255.255.255.255 |       |      ||                    |                     ||
====================================================================================================================
```

Fig. 5. Checking the portability of the configuration in `pf`.

where `sIp` and `dIp` stand for source and destination IP address, `sPort` and `dPort` for their ports, `prot` for the protocol, `tr_scr`, `tr_dst` for the transformations applied (for brevity the IP address and the port are separated by a ":").

Starting from this tabular representation, F2F checks if the same transformations can be obtained by a configuration of the target system. Roughly, to do that, it visits the target control diagram and verifies if it is possible to apply the relevant transformations in the nodes of the paths followed by a packet. If this is not possible, then the counterexamples are reported in tabular form.

Back to our example, Figs. 4 and 5 display the output of F2F on the configuration in Fig. 3 when the targets are `ipfw` and `pf`, respectively. The parameters of the tool are the source firewall system (here `iptables`); a file with the firewall interfaces and their addresses (`interfaces`); the input configuration (`iptables.conf`); and the target system. Some problems are detected because of a different expressivity power of `iptables` and `ipfw`, and thus porting this configuration cannot preserve the semantics. For each of them the tool displays the reason why, and the rules of the source configuration that cannot be ported, in tabular form.

The first problem with `ipfw` arises when (i) a packet has source and destination addresses of the firewall; (ii) the source is transformed by `SNAT` to an address of the firewall itself (`Self`); (iii) the destination is transformed `DNAT` to an address not of the firewall (`~Self`).The reason is that `ipfw` cannot apply both transformations in this case.The second table lists those packets that cannot be correctly handled. For example, the packet with source IP 192.168.0.1 and any source port, with destination IP 10.0.0.1 and port 123, and protocol UDP cannot be translated to a packet with source 151.15.185.183 and destination 193.204.114.232, with no changes to the ports.

Also the second problem with `ipfw` arises because of a pair `SNAT` and `DNAT` on a packet generated by the firewall but this time with another host as destination. E.g., the UDP packet with source 192.168.0.1 : $*$ and destination 151.15.185.184 : 123 shown in Fig. 3 cannot be transformed in the packet with source 151.15.185.183 : $*$ and destination 193.204.114.232 : 123.

Figure 5 shows that the configuration in Fig. 3 cannot be ported to `pf`, as well. Besides the same problems already discussed for `ipfw`, the tool reports in the first subtable that a packet generated by the firewall and directed to one of its interfaces cannot be redirected to another external host, e.g., the remote NTP, by applying a `DNAT`. Note that this case implicitly shows that there is a packet expressible in `ipfw`, but not in `pf`.

4 Why and How It Works

This section describes the internals of our tool that works in two steps. The first extracts the meaning of a configuration in a given language $\mathcal{L}$ as a function from packets to transformations, which has been intuitively represented as a table in Sect. 3. The second step uses the semantics just obtained and checks if the configuration can be expressed in a target language $\mathcal{L}'$.

4.1 The Denotational Semantics of Configurations in a Nutshell

Given a configuration in $\mathcal{L}$, FWS decompiles it into a set of rules in IFCL, associated with the relevant control diagram of $\mathcal{L}$ [4].

This intermediate representation of the configuration is then associated with a function $\mathcal{F}$ that maps packets into (the sets of) transformations they are subject to while the kernel of the operating system processes them. (Recall that packets belonging to established connections are not treated here, since they are accepted by default and usually never translated[5]). A *transformation* can be the discard

[5] Actually, some translations may occur, typically `SNAT`, but these are performed by other components of the operating system at run-time.

of a packet (represented by $\lambda_\perp$) or a specification of what happens to each of its fields: either it is left unchanged (id) or it is transformed by NAT to a specific value a (λ_a). The semantics of a single ruleset and then of an entire IFCL firewall are defined in [5] as the composition of those functions, collected while packets traverse the paths of the control diagram. In Sect. 2 the meaning of a firewall configuration is expressed as a table, which is a succinct representation of its denotational semantics, where each row corresponds to an equivalence class of packets associated to the same transformation.

For simplifying the treatment of the expressivity of firewall languages it is convenient to annotate each node in a control diagram with the kind of transformations that are allowed in that node. This is because in the translation to IFCL some types of rules cannot be associated with all the nodes. For example, in `pf` rules containing the DROP target can only be associated with nodes q_1 and q_3. Graphically, in Fig. 1, we include these annotations, or *capability labels*, in boxes, and when the only allowed transformation is the identity `ID`, we omit the box. We dispense the actual reasons of the exact association of nodes with labels in the three control diagrams,[6] because not necessary for the present treatment; the interest reader can find the complete presentation of them in [5].

4.2 Checking Portability of Configurations

By exploiting the semantic function $\mathcal{F}$ and the control diagram of the target language $\mathcal{L}'$, we check if a configuration having the same semantics is expressible in $\mathcal{L}'$. The underlying idea of the checking algorithm follows. Given a packet p, we follow the path in the control diagram of $\mathcal{L}'$ from the initial node to the final one, if p is accepted, or to the node that drops p. Along the path, we collect the list of the capabilities T_j associated with each node q_j, i.e. the transformations that can be applied when the control reaches q and graphically included in boxes. Now, let $\mathcal{F}(p)$ be the sequence of transformations $t_1, \cdots, t_n$ that transform p to $\tilde{p}$. If for all the packets and for all j it is $t_j \in T_j$, then the configuration can be ported from $\mathcal{L}$ to $\mathcal{L}'$ (recall that t_j can be the identity).

Consider again the UDP packet p with source IP $192.168.0.1 : *$ and destination IP $10.0.0.1 : 123$. As discussed in Sect. 3, it is accepted by the configuration in Fig. 3, traversing the following path in the control diagram of `iptables`:

$$q_i \rightarrow q_7 \rightarrow q_8 \rightarrow q_9 \rightarrow q_{10} \rightarrow q_{11} \rightarrow q_f$$

where node q_8 transforms 10.0.0.1:123 to 193.204.114.232: 123 by a `DNAT`, and the node q_{11} transforms $192.168.0.1 : *$ to $151.15.185.183 : *$, the interface towards the Internet. Indeed, `DNAT` is a capability of q_8 and `SNAT` of q_{11}.

Instead, p can only follow the two following paths in the control diagram of `ipfw`, attempting to obtain the same behavior of `iptables`:

[6] As a matter of fact, it is not `ipfw` that actually translates address, but it demands this task to other lower level components, possibly to the operating system kernel itself. For the sake of generality, we have only modelled such calls, because the actual translations heavily depend on the specific setting of the system hosting the firewall.

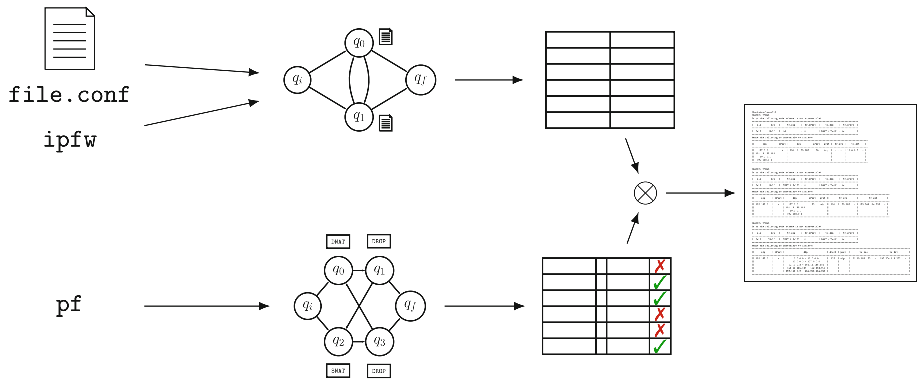


Fig. 6. How F2F checks the portability of an `ipfw` configuration (`file.conf` in the figure) into `pf`.

$$q_i \to q_1 \to q_f \qquad\qquad q_i \to q_1 \to q_0 \to q_f$$

We have the following two cases

1. in the leftmost path a `DNAT` occurs in q_1 and p is accepted with no source address transformation applied, violating the intended semantics;
2. in the rightmost path an `SNAT` applies, then in q_0 we do not apply the `DNAT` because otherwise the predicate of the arc leading to q_f would be falsified. If instead we apply `DNAT` in q_0, the control moves back to q_1 and a loop will be detected and the packet will be dropped.

Consider now p' with with source IP 192.168.0.1 : $*$ and destination IP 151.15.185.182 : 123 that is transformed to one with source 151.15.185.183 : $*$ and destination 193.204.114.232 : 123 (of the NTP server) along the following accepting path in `iptables` (the same just seen above):

$$q_i \to q_7 \to q_8 \to q_9 \to q_{10} \to q_{11} \to q_f$$

In the control diagram of `pf` there only is one path that p' can follow:

$$q_i \to q_0 \to q_1 \to q_f$$

Note that while the source address can be transformed by node q_0, it is not possible to perform `DNAT` because the only node with capability `DNAT` is q_2, which is not reachable since the predicate of the arc requires the packet destination to be an interface of the firewall. Hence when the packet is accepted by the final node its fields do not contain the expected values.

Given a control diagram with its capability labels, the algorithm of [5] returns a table in which each row represents both the expressible and the inexpressible transformations for a given class of packets. The procedure for building these table is similar to the one sketched above for checking a single packet p. Rather than on single packets, our algorithm actually works on (a representative of the) equivalence classes of packets that cover every possible behaviour of the firewall in hand. The tool F2F matches this expressivity table with the one representing $\mathcal{F}$ and computed by FWS, looking for clashes. In Fig. 6 we recapitulate the steps needed for evaluating the portability of a configuration: (i) first translate the source policy into IFCL; use the control diagram of the source language (with its capability labels) and perform a syntactic translation; and then compute the semantic function $\mathcal{F}$ in a tabular form (top of the figure); (ii) similarly, take the control diagram of the target language with its capability labels, and derive the table representing its expressive power (bottom of the figure); (iii) finally compare the two tables to produce the final result (right part of the figure).

5 Conclusions

We have briefly introduced the pipeline of [3,4] that supports system administrators in better understanding, in analysing and maintaining a firewall configuration, and in porting it from one firewall configuration language to another. It is folklore that firewall configuration languages only differ in pragmatic aspects and in syntactic sugar. To our surprise, we discovered that this is not the case, and we proved in [5] that the most used ones, `iptables`, `ipfw` and `pf`, form a hierarchy of expressivity.

We have described F2F, a tool available online[7] that checks if a configuration is expressible in a given language. We have also discussed some revealing cases in which this is not possible, by inspecting the outputs of F2F. This results have the form of a table showing the shape of the packets that can be filtered and redirected in one language, but not in another. Here, we have mainly concentrated on the form of the source and destination addresses before and after possible translations. Although in a preliminary version, our tool can be used efficiently, because it is a matter of seconds checking the expressivity of a medium size configuration.

Network administrators can use F2F to choose the firewall system more appropriate to the current situation, or at least to evaluate if their preferred one, as well as which of the available tools are the more suited for implementing the needed configuration. Also, a system administrator can overcome the detected limitation of the firewall systems in use and patch the configuration in hand in the very troubling points, by resorting to calls to procedures in some programming language.

Future work will consider different firewall systems, like Cisco-IOS, which is particularly challenging because the control diagram is also affected by routing choices. We plan to involve systems administrators in experimenting F2F on real configurations, and get feedback from them. This of course requires to improve the interface of F2F.

[7] https://github.com/lceragioli/F2F.

References

1. Adão, P., Bozzato, C., Rossi, G.D., Focardi, R., Luccio, F.L.: Mignis: a semantic based tool for firewall configuration. In: IEEE 27th Computer Security Foundations Symposium, CSF 2014, pp. 351–365 (2014)
2. Adão, P., Focardi, R., Guttman, J.D., Luccio, F.L.: Localizing firewall security policies. In: Proceedings of the 29th IEEE CSF, Lisbon, Portugal, 27 June–1 July 2016, pp. 194–209 (2016)
3. Bodei, C., Degano, P., Focardi, R., Galletta, L., Tempesta, M.: Transcompiling firewalls. In: Bauer, L., Küsters, R. (eds.) POST 2018. LNCS, vol. 10804, pp. 303–324. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-89722-6_13
4. Bodei, C., Degano, P., Focardi, R., Galletta, L., Tempesta, M., Veronese, L.: Language-independent synthesis of firewall policies. In: Proceedings of the 3rd IEEE European Symposium on Security and Privacy (2018)
5. Ceragioli, L., Degano, P., Galletta, L.: Are All Firewall Systems Equally Powerful? Submitted for publication. https://sysma.imtlucca.it/wp-content/uploads/2019/03/firewall-expressivity.pdf
6. Ceragioli, L., Galletta, L., Tempesta, M.: From firewalls to functions and back. In: Italian Conference on Cybersecurity ITASEC 2019. CEUR Proceedings, vol. 2315 (2019). http://ceur-ws.org/Vol-2315/paper04.pdf
7. Cuppens, F., Cuppens-Boulahia, N., Sans, T., Miège, A.: A formal approach to specify and deploy a network security policy. In: Dimitrakos, T., Martinelli, F. (eds.) Formal Aspects in Security and Trust. IIFIP, vol. 173, pp. 203–218. Springer, Boston, MA (2005). https://doi.org/10.1007/0-387-24098-5_15
8. Diekmann, C., Hupel, L., Michaelis, J., Haslbeck, M.P.L., Carle, G.: Verified iptables firewall analysis and verification. J. Autom. Reason. **61**(1–4), 191–242 (2018)
9. Foley, S.N., Neville, U.: A firewall algebra for openstack. In: 2015 IEEE Conference on Communications and Network Security, CNS 2015, pp. 541–549 (2015)
10. Martínez, S., Cabot, J., Garcia-Alfaro, J., Cuppens, F., Cuppens-Boulahia, N.: A model-driven approach for the extraction of network access-control policies. In: Proceedings of the MDSec 2012, pp. 5:1–5:6. ACM (2012)
11. Packet Filter (PF). https://www.openbsd.org/faq/pf/
12. Russell, R.: Linux 2.4 packet filtering HOWTO (2002). http://www.netfilter.org/documentation/HOWTO/packet-filtering-HOWTO.html
13. The IPFW Firewall. https://www.freebsd.org/doc/handbook/firewalls-ipfw.html
14. The Netfilter Project. https://www.netfilter.org/