TÉCNICO
LISBOA

A declarative based tool for reasoning about CISCO IOS firewall configurations

Shams Karim Valibhai

Thesis to obtain the Master of Science Degree in

Computer and Electrical Engineering

July 2019

Declaration

I declare that this document is an original work of my own authorship and that it fulfills all the requirements of the Code of Conduct and Good Practices of the Universidade de Lisboa.

Acknowledgments

I would like to thank my parents. My mom, for all the support given during these past few years. And my dad who was always extremely patient and understanding.

A thank you to all my friends who helped me, in any way, during the course of this thesis. Special thank yous to Francisco Pereira and Miguel Rodrigues dos Santos for all the encouragement they gave me.

I would also like to acknowledge my supervisor, Prof. Pedro Adão, for all the insight and support.

To everyone who had a role during this thesis – Thank you.

Abstract

One of the mandatory tasks assigned to computer network administrators is to keep unwanted or malicious traffic outside the network. Because of this, firewalls became a vital component in any computer network connected to the Internet. However, configuring and maintaining a firewall is a complicated and error-prone process mostly due to the design model used in conventional firewalls, where the ordering between firewall rules matters.

To simplify these tasks, we propose a low-level abstraction of a router, that is a simplification of a CISCO IOS device, that contains the concepts of access-lists, rules and policies existent in these devices. This is accompanied by a semantic allowing both packet filtering and address translation. We then capitalize on the MIGNIS firewall specification language proposed by Adão et al [1] that is simple and powerful enough to specify firewall configurations and its semantic is immune to the relative ordering of rules, and prove a sound translation from this model to our low-level abstraction thus entailing both simple specification and easy verification of firewall policies. We also provide conditions over the policies for this translation to be complete. Finally, we developed a tool that translates MIGNIS configurations into real CISCO IOS configurations.

Keywords

Firewall, Cisco IOS, MIGNIS, Network Security, Firewall semantics, Network Address Translation (NAT)

Resumo

Uma das tarefas fundamentais atribuídas aos administradores de redes de computadores é que qualquer tráfego indesejado ou malicioso seja bloqueado da rede. Devido a isto, as *firewalls* tornaram-se um componente imprescindível em qualquer rede de computadores ligada à Internet. A configuração e manutenção de uma *firewall* é um processo complicado e propenso a erros. Isto deve-se, em grande parte, ao modelo no qual as *firewalls* convencionais são desenhadas, onde a ordem entre regras de *firewall* é relevante.

De modo a simplificar estas tarefas, propomos um abstracção de um router, que é uma simplificação de um dispositivo CISCO IOS, contendo os conceitos de *access-list*, regras e políticas presentes nestes dispositivos. Esta abstração é acompanhada de uma semântica que permite a filtragem de pacotes e tradução de endereços (NAT). Reintroduzimos a linguagem de configuração de *firewalls* MIGNIS proposta por Adão *et al* [1], que é simples e poderosa o suficiente para especificar a configuração completa de uma *firewall* e cuja semântica não depende da ordem das regras. A partir desta linguagem, fornecemos uma tradução correta para a nossa abstração do router, permitindo uma configuração simples e segura. Isto é acompanhado de um conjunto de condições para que a tradução seja também completa. Finalmente, fornecemos também uma ferramenta que traduz regras da linguagem MIGNIS para comandos compatíveis com dispositivos CISCO IOS.

Palavras Chave

Firewall, Cisco IOS, MIGNIS, Segurança de redes, Semântica de *firewalls*, *Network Address Translation* (NAT)

Contents

List of Figures

List of Tables

Listings

Acronyms

ACL	Access Control List
API	Application Program Interface
ARP	Address Resolution Protocol
ASA	Adaptive Security Appliance
BGP	Border Gateway Protocol
CLI	Command Line Interface
DHCP	Dynamic Host Configuration Protocol
DNS	Domain Name System
EIGRP	Enhanced Interior Gateway Routing Protocol
GUI	Graphical User Interface
GRE	Generic Routing Encapsulation
HTTP	Hypertext Transfer Protocol
ICMP	Internet Control Message Protocol
IGMP	Internet Group Management Protocol
IP	Internet Protocol
ISP	Internet Service Provider
L2TP	Layer 2 Tunneling Protocol
LAN	Local Area Network
MAC	Media Access Control

MDL	Model Definition Language
NAT	Network Address Translation
NVI	NAT Virtual Interface
OSPF	Open Shortest Path First
QoS	Quality Of Service
RIP	Routing Information Protocol
SMTP	Simple Mail Transfer Protocol
SSH	Secure Shell
TCP	Transport Control Protocol
UDP	User Datagram Protocol
VPN	Virtual Private Network
WAN	Wide Area Nework
WFQ	Weighed Fair Queuing
ZBFW	Zone-based Firewall

1 Introduction

Contents

In the current age, computer networks and devices have become an integral part of our civilization. It is estimated that over eleven billion devices [3] are connected to the Internet. Any pair of these network devices can connect to each other and exchange resources. While global connectivity is one of the Internet's main achievements, there are many situations where it is not ideal. In these situations, network administrators often use a firewall to control the flow of traffic and limit connectivity. The process of configuring a firewall can be very complex, depending on the security requirements. Maintaining and updating the configuration is an even more error prone process. In this chapter we will present a brief overview of the Internet and expand on the use and importance of firewalls. We will then formulate the problem at hand and explain how we intend to solve it.

1.1 The Internet

The communication networks that preceded the Internet started to be developed in the United States, in the 1960s. Initially, they were not available to the public and their main uses were for academic and governmental purposes. Since then, the Internet has been formed and is now available publicly, reaching almost 50% of the world population [4]. This amounts to almost 4 billion users.

As the name suggests, the Internet is a network of networks, arranged in a close to hierarchical organization. Computer networks are classified according to their geographical scale and purpose. A Local Area Network (LAN) connects devices inside a short area, like a home or a building, while a Wide Area Nework (WAN) provides connection over city or country-wide areas. A Virtual Private Network (VPN) is a different kind of computer network, it is built upon an already existing computer network, like the Internet, and provides users with the illusion of being in the same private network.

Operating between computer networks are devices known as routers. These devices are usually connected to two or more networks and are responsible for routing data between those networks, allowing internetwork communication. The Internet Protocol (IP) is the protocol used between two communicating network hosts and the information contained in it is used by routers to know where to forward the data they receive.

In IPv4, the most used version of IP, each network host is identified by a 32 bit address. Routers use this address in combination with their routing tables to know where to forward network traffic. Above the IP protocol run two main transport protocols: TCP and UDP. The Transport Control Protocol (TCP) offers a stateful and reliable transport channel, while the User Datagram Protocol (UDP) serves as a low overhead and stateless transport method. IPv4 also defines a range of private addresses, which are meant for private use and are not reachable through the Internet.

Surprisingly for IPv4's creators, the number of Internet-connected devices has far exceeded the 4.3 billion addresses available with the protocol's 32 bit address space. The new version, IPv6, fixes this shortage with a 128 bit address space, but its worldwide deployment may still take a few decades to be complete. In the mean time, Network Address Translation (NAT) is the functionality used to attenuate the shortage. With NAT, several network hosts may use the same IPv4 address to communicate in the Internet. Because of this, NAT is mostly used to translate between the private address range of a network and a public address, provided by an Internet Service Provider (ISP).

1.2 Growth of Internet security and firewalls

As the number of Internet users rises, so does the number of users with mischievous intentions. The amount of sensible data carried by or accessible the Internet and this malicious group of users creates a great risk. When speaking of big enterprises, this risk can be quantified up to billions of dollars [5].

One method used by network administrators to increase network security is a firewall. At its most basic definition, a firewall functions as a packet filter, allowing and dropping packets according to a configuration. They are usually placed at the border between a private network, with trusted users, and an untrusted public network like the Internet. While it is outside the scope of this work, it is also worth noting there exist other types of firewalls, usually placed on the network devices themselves and blocking traffic at an application level.

The main purpose of a firewall is to limit connectivity between hosts from different networks. Usually, a firewall is configured to only allow the minimum of traffic possible, as required by the users. This approach diminishes the effectiveness of external threats, since those would have to use an explicitly allowed traffic flow.

Due to their position, at the border of a private network, firewalls are also commonly used to employ NAT. While the translations main motivation is to attenuate IPv4 address shortage, they also serve as a security measure by hiding the real address of private hosts. In Figure 1.1, we present an example with a set of networks and a firewall placed between them. In this case we can observe possibly dangerous traffic flows, like the one between the daughter's computer and the suspicious website.

1.3 Motivation and existing solutions

Network firewalls can be implemented in a variety of network devices. It is possible to have a computer running a router-like Unix distribution serving as the gateway to a private LAN. In these cases, it is practical to use that same machine as the network firewall.

While computers can be used, most LANs just use a router as the gateway for all inside traffic. This requires a firewall to be configured on the router itself.

In any of the previous cases, the configuration of the firewall may not be a simple task. A network

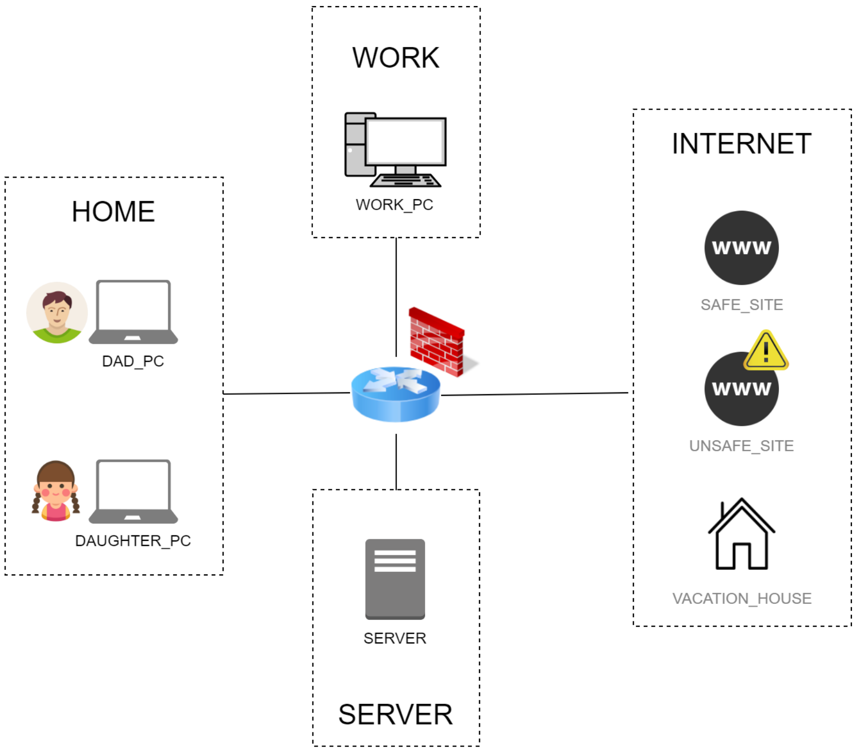


Figure 1.1: Various networks and hosts separated by a firewall.

administrator must take several requirements in consideration when configuring a firewall and overlaps between requirements often happen. The order in which the rules are inserted is often important in firewall configuration, adding complexity to the process and making the management of firewall configurations almost an art.

Having a difficult time maintaining a firewall does not just result in wasted time. When dealing with a complex configuration, it is possible for small mistakes such as ordering to result in significant security risks. As such, providing network administrators with a tool to help with this process will not only save time, but also reduce the chance of a configuration mistake.

One family of helpful tools includes those that verify and check a firewall configuration for mistakes, warning about unintended behaviors. [6] and [7] are two examples of tools that do so with a graphical approach, while [8] follows a non-visual approach. In [9], an *expert system*-like tool allows users to verify Cisco access list configurations.

The other family of tools consists of those that help design and apply a firewall configuration. Examples of such tools are presented in Chapter 2.

In this work we are going to look at MIGNIS [1] that is part of the second family and has a precise semantics that is order-independent and simple to read syntax which applies to both filtering and NAT functionalities. It is currently used in the MIGNIS tool [10], offering a translation to *iptables* commands. An indispensable aspect of such a tool is that the translation is theoretically proven to comply with the *iptables* semantics. While this tool is compatible with most Linux distributions, it does not serve any purpose for the configuration of firewalls in routers.

1.4 Objective

The objective of this work is to provide network administrators with a tool to assist in the configuration of firewall and NAT functionalities in routers. This tool must, while simplifying the process, ensure the higher level semantic is followed precisely by the router.

To achieve this, we will make use of the existing MIGNIS semantics, which are well defined and NAT-aware. Due to the widespread use of Cisco routers, our goal will be to translate MIGNIS configurations into Cisco IOS commands, proving this translation to be sound and complete. The soundness result would guarantee that all low-level flows have a corresponding high-level flow, and thus captured by our high-level semantics, whereas the completeness result ensures that all high-level flows are implementable.

1.5 Document structure

In Chapter 2 we describe some problems that arise in the configuration of firewalls, along with several firewall design models and firewall configuration tools with graphical interfaces. Chapter 3 presents our target platform, Cisco IOS, going over its main features and introducing an abstraction of its behavior. The MIGNIS semantic is also formally introduced in Chapter 4. In Chapter 5, we prove the soundness and completeness of the translation used in our MIGNIS implementation. Finally, in Chapter 6, we draw conclusions about the work that was done and identify some of the limitations of our solution, along with possible future work.

2 Related Work

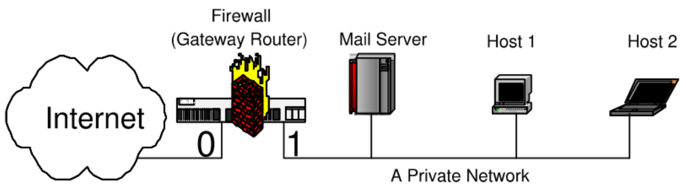
Contents

This chapter will focus on work done on firewall configuration and correctness. We will start by going over the flaws of traditional firewall design. After that, several firewall design models will be presented. The final section will analyze two different firewall configuration tools with graphical interfaces.

2.1 Shortfalls of conventional firewalls

In conventional firewall implementations, the decision to pass or discard a packet is dependent on a sequence of rules, which specify criteria to match packets against and actions to take in case of a match. This behavior adds two dimensions to the configuration: the content of each rule and the order between the rules. The placement of rules in the sequence matters and not always in obvious ways.

In [2], Gouda and Liu present three different categories of problems associated with conventional firewall design: consistency, completeness and compactness. In order to assist with the explanation of these problems, we will use the example presented in this same paper, a sequence of four firewall rules, displayed in Figure 2.1.



1. Rule r_1: $(\mathbf{I} = \mathbf{0}) \wedge (\mathbf{S} = \mathbf{any}) \wedge (\mathbf{D} = \mathbf{Mail\ Server}) \wedge (\mathbf{N} = \mathbf{25}) \wedge (\mathbf{P} = \mathbf{tcp}) \rightarrow \mathbf{accept}$
   (This rule allows incoming SMTP packets to proceed to the mail server.)
2. Rule r_2: $(\mathbf{I} = \mathbf{0}) \wedge (\mathbf{S} = \mathbf{Malicious\ Hosts}) \wedge (\mathbf{D} = \mathbf{any}) \wedge (\mathbf{N} = \mathbf{any}) \wedge (\mathbf{P} = \mathbf{any}) \rightarrow \mathbf{discard}$
   (This rule discards incoming packets from previously known malicious hosts.)
3. Rule r_3: $(\mathbf{I} = \mathbf{1}) \wedge (\mathbf{S} = \mathbf{any}) \wedge (\mathbf{D} = \mathbf{any}) \wedge (\mathbf{N} = \mathbf{any}) \wedge (\mathbf{P} = \mathbf{any}) \rightarrow \mathbf{accept}$
   (This rule allows any outgoing packet to proceed.)
4. Rule r_4: $(\mathbf{I} = \mathbf{any}) \wedge (\mathbf{S} = \mathbf{any}) \wedge (\mathbf{D} = \mathbf{any}) \wedge (\mathbf{N} = \mathbf{any}) \wedge (\mathbf{P} = \mathbf{any}) \rightarrow \mathbf{accept}$
   (This rule allows any incoming or outgoing packet to proceed.)

Figure 2.1: Firewall configuration example from [2].

In Figure 2.1's list of rules, **I**, represents the number of the ingress interface for a packet. **S** and **D** represent, respectively, the source and destination addresses. **N** represents the destination port and **P** the protocol running over IP.

2.1.1 Consistency

The first problem, consistency, stems from the ordering between rules and possible conflicts between them. By conflicts, we mean overlaps in the criteria used to match packets. If a packet does match the criteria for several different rules, it will only be affected by the first of those rules, following the sequence's order.

In the presented example, rule r_2 is intended to discard any packet originated from known malicious hosts and rule r_1 accepts any Simple Mail Transfer Protocol (SMTP) packet – more specifically, any TCP packet addressed to port 25 – destined to the mail server. If we consider a SMTP packet destined to the mail server but originated from a malicious host, it is possible to see where the rules conflict. Looking solely at rule r_2, it would seem that a packet like this should be dropped but, because of r_1, the packet will instead be accepted by the firewall.

The main takeaway from this problem is that, when checked in a sequence, rules lose their original meaning. Rule r_2's meaning changes from *discard any packet originated from malicious hosts* to *discard any packet from malicious hosts that is not SMTP and destined to the mail server*. To discern the real meaning of a rule, it becomes necessary to check the relation between itself and all of the preceding rules.

2.1.2 Completeness

Another problem that occurs in conventional firewalls is the lack of completeness. This means not all packets are accounted by the firewall rules. The usual solution is to insert a rule similar to r_4 at the end of the sequence. In this example, the rule accepts all unaccounted for packets, but in other cases it might drop them instead. However, using such a rule is not a good idea because it is not easy to remember all possible traffic flows, especially in large networks. This could lead to situations where traffic gets, accidentally, accepted (or dropped).

2.1.3 Compactness

Conventional firewalls can also contain redundant rules. A redundant rule is a rule that, if removed, does not change the behavior of the firewall in any way. Removing such rules can lead to a more compact firewall, improving readability and maintainability. Rule r_3, from Figure 2.1, is a redundant rule, since its removal would not change how packets are treated by the firewall. Any packet reaching and matching r_3 will also reach and match r_4 if the former rule is removed.

2.2 Firewall design models

We will now look at different design models, going over their advantages and disadvantages.

2.2.1 Structured firewall design

In [2] Gouda and Liu propose an interesting, diagram-based, representation for firewall rules. A translation between this design and a sequence of rules is also provided, along with a proof of the sequence's equivalence to the original design.

A firewall decision diagram (FDD) is a directed graph in the form of a tree, meaning it is acyclic. Each non-leaf node of the tree is labeled with a field. Fields represent the parameters used by the firewall to distinguish packets. Each field has its own discrete domain, made up by an interval of integers belonging to $\mathbb{Z}_0^+$. Using the previous example, we could say the $interface$ field has domain [0,1]. Leaf nodes can only be labeled with two values: $accept$ or $discard$.

By definition, a path from the root of the tree to one of its leafs must contain a set of unique node labels. In other words, traversing from top to bottom we only see each field once, at most.

Edges in the graph are also labeled, but in a different way. Let's consider $L(e)$ to denote the label of edge e. Additionally, $F(u)$ will denote the field assigned to a node u. If node u is the tail of edge e, then $L(e)$ must be a subset of $F(u)$'s domain. Using a concrete example, if edge e originates from a node labeled with the $interface$ field, then the label of e must be a subset of [0,1], $interface$'s domain.

Figure 2.2, taken from the work in question, shows an example of an FDD over two fields, F_1 and F_2. Both fields have the same domain: the interval [1, 10].

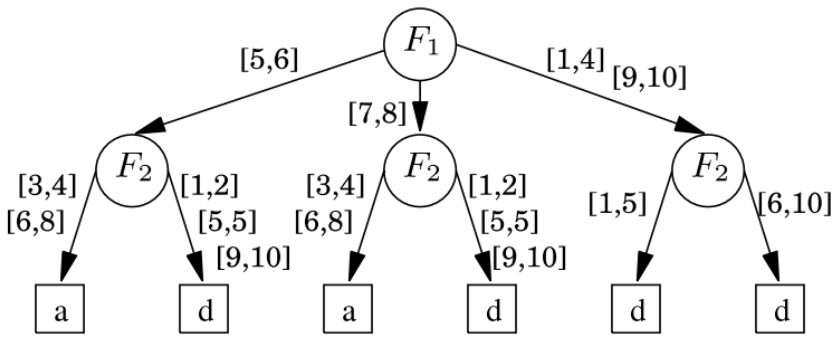


Figure 2.2: FDD example from [2].

Analyzing Figure 2.2 shows two additional restrictions present in FDDs:

- *Consistency*: If two edges, e and e', originate from the same node, then their labels must not overlap. This can be represented by: $L(e) \cap L(e') = \varnothing$.
- *Completeness*: If $E(u)$ is the set of all edges originating from node u, then the union of the labels from $E(u)$ is equal to $F(u)$'s domain. A more concise representation would be $\bigcup_{e \in E(u)} L(e) = D(F(u))$, where $D(f)$ represents the domain of field f.

We can now make a parallel between this generic explanation and a firewall configuration. As was explained before, each field in an FDD represents a different attribute for a packet. This attribute can be the source or destination address, the protocol, the ingress or egress interface, or any other relevant information. Labels represent subsets of these fields' domains. A label corresponding to the

$source\ address$ field could be, for example, the address block 192.168.1.0/24 represented as a continuous set of integers.

When matching a packet, FDDs are traversed from top to bottom, each node representing a test on a specific field. Falling back to Figure 2.2, let's suppose a packet has fields $F_1 = 5$ and $F_2 = 9$ and the firewall configuration is the FDD pictured. Starting from the root, the first step is to test against field F_1, which is equal to 5. The next step, looking at all outgoing edges, is to follow the one which label contains this value, which is the leftmost one ([5,6]). We now arrive at a node labeled with field F_2 and repeat the process, taking us to a node labeled with d ($discard$). This means the packet would be discard by the firewall.

Any path from the root to one of the leaf nodes is called a decision path. It can be represented as $\langle u_1 e_1 \cdots u_k e_k u_{k+1} \rangle$ where u_1 is the root and u_{k+1} is a leaf node. Such path corresponds to the following firewall rule: $F_1 \in S_1 \wedge \cdots \wedge F_d \in S_d \longrightarrow \langle decision \rangle$, where we consider our FDD to be over a number of fields d. The value of S_i is defined as follows:

$$S_i = \begin{cases} L(e_j) & \text{if there is a } u_j \text{ in the decision path such that } F(u_j) \text{ is the same as } F_i \\ D(F_i) & \text{in all remaining cases.} \end{cases}$$

This definition allows us to create a list of rules from any FDD. The tree in Fig. 2.2 contains six unique decision paths, so it is possible to extract six unique firewall rules from it. These rules are shown in Figure 2.3, also taken from the paper in question.

$$\begin{array}{ll} r_1\colon F_1 \in [5,6] \wedge F_2 \in [3,4] \cup [6,8] & \to a \\ r_2\colon F_1 \in [5,6] \wedge F_2 \in [1,2] \cup [5,5] \cup [9,10] & \to d \\ r_3\colon F_1 \in [7,8] \wedge F_2 \in [3,4] \cup [6,8] & \to a \\ r_4\colon F_1 \in [7,8] \wedge F_2 \in [1,2] \cup [5,5] \cup [9,10] & \to d \\ r_5\colon F_1 \in [1,4] \cup [9,10] \wedge F_2 \in [1,5] & \to d \\ r_6\colon F_1 \in [1,4] \cup [9,10] \wedge F_2 \in [6,10] & \to d \end{array}$$

Figure 2.3: Rules generated from the FDD in Fig. 2.2, taken from [2].

From the two restrictions listed before, *consistency* and *completeness*, it is possible to deduce that, for any possible packet, one and only one rule will match against it. More specifically, *consistency* assures us that, at maximum, only one rule will be matched by any packet. While *completeness* sets the lower bound, making it so that at least one rule is matched.

What has been presented allows for a firewall administrator to design its firewall as an FDD and convert it to a set of rules accepted by the firewall. This set of rules fixes two of the problems in section 2.1 but still carries the *compactness* problem. Another problem is also visible in Fig. 2.3, the existence of *non-simple* rules. A *simple* rule is a rule where, for $1 \leq i \leq d$, S_i is a continuous interval of integers. Most firewalls only accept *simple* rules, meaning the set of rules in the referenced picture is not ideal, since none of its rules is *simple*.

With the above problems in mind, the authors propose several additional algorithms which help attenuate them and generate a set of rules, starting from a regular FDD. The steps taken in this process are illustrated in Fig. 2.4.

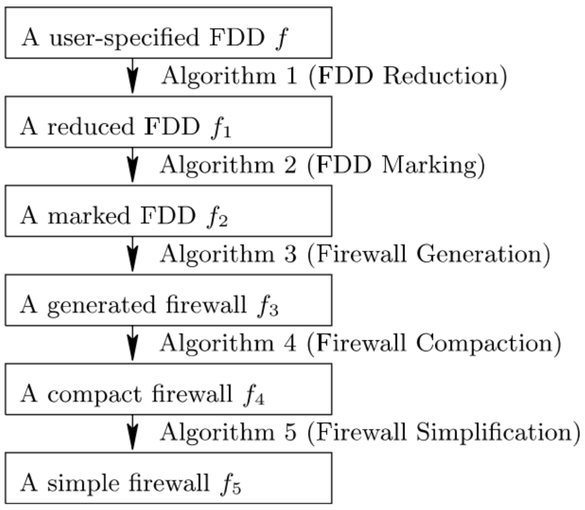

A user-specified FDD f
Algorithm 1 (FDD Reduction)
A reduced FDD f_1
Algorithm 2 (FDD Marking)
A marked FDD f_2
Algorithm 3 (Firewall Generation)
A generated firewall f_3
Algorithm 4 (Firewall Compaction)
A compact firewall f_4
Algorithm 5 (Firewall Simplification)
A simple firewall f_5

Figure 2.4: Steps and algorithms to go from an FDD to a final, equivalent, firewall configuration. Taken from [2].

Using the terminology from Fig. 2.4 we can say that algorithms 1 and 4 deal with the compactness of the firewall, while algorithms 2 and 5 are related to the existence and treatment of *simple* rules. In every step, the semantic of the firewall does not change, meaning the final firewall keeps the same semantics as the original FDD.

2.2.1.A Summary

The work presented in [2] sets up an intuitive and easy to use method to create a firewall configuration. It tackles all the problems presented in Section 2.1 and improves on all of those. The definition used for the firewall configuration is language-agnostic, but it is easily applied to most real firewalls.

While the presented solution is quite effective, it fails to consider stateful firewalls. Most modern firewalls use some kind of connection tracker which keeps information about established connections and takes that into consideration when inspecting packets. On the other side, stateless firewalls, which the work in question is aimed at, simply follow its configuration and do not have any notion of connections. This difference in behavior makes it so that it is not possible to apply this work to a stateful firewall.

Another feature not considered is NAT. Many firewalls also serve as address translators for the networks they serve, leading many firewall utilities to also provide NAT functionality. Since this solution is not NAT aware, it does not serve our purposes.

2.2.2 *iptables*

The *iptables* tool [11], available in most Linux distributions as part of the Netfilter framework, is a networking software that allows users to configure packet filtering, address translation and other useful operations over network packets. As a result, this tool allows a Linux machine to be used as a firewall while also dealing with NAT. While we will be presenting a very implementation focused view of *iptables*, the main goal of this section is to expose the firewall design model in which *iptables* is based.

A firewall rule in *iptables* follows a very similar structure to the ones in Figure 2.1. However, the criteria used to match a packet is much more broad. Likewise, the action to be taken is also much more flexible, allowing a rule to modify a packet in several ways. Figure 2.5 shows a basic *iptables* configuration.

```
## Accept all traffic from the eth0 interface and ICMP traffic from eth1 directed to this machine
iptables -t filter -A INPUT -i eth0 -j ACCEPT
iptables -t filter -A INPUT -i eth1 -p icmp ACCEPT

## Make the all filter chains drop unmatched traffic
iptables -t filter -P INPUT DROP
iptables -t filter -P OUTPUT DROP
iptables -t filter -P FORWARD DROP

#Drop all traffic from an external host (in eth1) directed to our local network (eth0)
iptables -t filter -A FORWARD -i eth1 -s 20.0.0.20 -o eth0 -j DROP

## Translate traffic directed to eth1 port 555 to a local machine and let it through
iptables -t nat -A PREROUTING -i eth1 -dport 555 -j DNAT --to 192.168.0.50
iptables -t filter -A FORWARD -i eth1 -d 192.168.0.50 -dport 555 -j ACCEPT
```

Figure 2.5: Examples of *iptables* rules.

iptables rules are placed in lists, just as in conventional firewalls, which are called *chains*. These chains belong to one of five *tables*: `filter`, `nat`, `mangle`, `raw` and `security`. Chains in the `filter` table include rules that either drop or let packets through. As the name implies, rules in the `nat` table serve to apply NAT, while rules in the `mangle` table deal with any other kind of packet mangling. The latter two tables have very specialized uses: `raw` chains are used when we wish certain packets not to be tracked and rules in the `security` table are used in conjunction with Mandatory Access Control.

By default, *iptables* contains a set of built-in chains which are tested against a packet at specific moments of its life cycle inside the host. These chains are named **PREROUTING**, **POSTROUTING**,**INPUT**,**OUTPUT** and **FORWARD**. A table does not necessarily contain all of the built-in chains. Figure 2.7 shows how chains and tables are ordered for any possible flow of traffic.

One feature that makes *iptables* unique, compared to conventional firewalls, is the ability to jump between chains. A user is allowed to create custom chains and jump to them from any of the built-in ones. It is also possible to return to the previous chain. This mechanic works similarly to a function stack in a regular programming language, as can be seen in Figure 2.6.

When a packet matches a rule, the action to be taken is defined by the *target* of that rule. This target

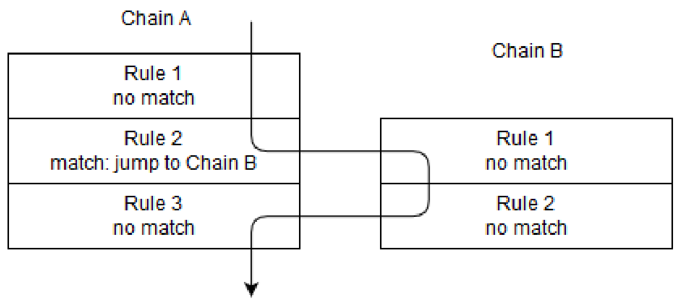


Figure 2.6: Order in which rules are tested when a packet jumps from one chain to another.

can be a simple command, like `ACCEPT` or `DROP`, an address translation (for rules in the `nat` table), a jump to another chain or a modification to one of the packet's attributes. The existence of *extensions* to the iptables software allows for an almost limitless variety of targets.

While the `ACCEPT` and `DROP` targets seem to mirror each other, they do not have exactly opposite effects. If a packet matches a rule with target `ACCEPT`, then its traversal in the corresponding table stops. However, when the target is `DROP`, the packet stops traversing every table, effectively dropping the packet. So, if a packet is accepted in one table but dropped in another, it ends up being dropped by *iptables*, despite matching an `ACCEPT` rule.

Another feature present in *iptables* is connection tracking. This is ensured by the `conntrack` module of Netfilter and it introduces a notion of *state* to the firewall. With `conntrack`, it becomes possible to use existing connections as part of the criteria to match a rule. As an example, a user can define a rule that matches to all packets in an established connection. By default, all packets are tracked. Rules in table `raw` allow a packet to be tagged as `UNTRACKED`, causing it to be ignored by `conntrack`.

2.2.2.A Summary

As a whole, *iptables* presents a very complete and versatile solution to firewall configuration. It builds upon conventional firewall design, adding features like NAT and connection awareness. While doing this, it also carries the previously presented shortfalls associated with conventional firewall design.

Due to the ability to jump between chains, this design model actually introduces a new problematic property: the ability to create cycles. In [12], Jeffrey and Samak show that detecting cycles in a configuration is an NP-complete problem. This also applies to the detection of unreachable rules[1], which falls under the *compactness* problem presented earlier.

In conclusion, while the design model followed by *iptables* allows for a feature-rich configuration, it does not provide a solution to any of the problems listed earlier. In fact, being able to create cycles only complicates the act of creating and maintaining a configuration. These conclusions are part of the

[1]The authors of [13] provide a linear solution to this problem, under the assumption packets have a fixed-length header.

reason Adão et al chose *iptables*, in [1], as the first target of the initial MIGNIS tool.

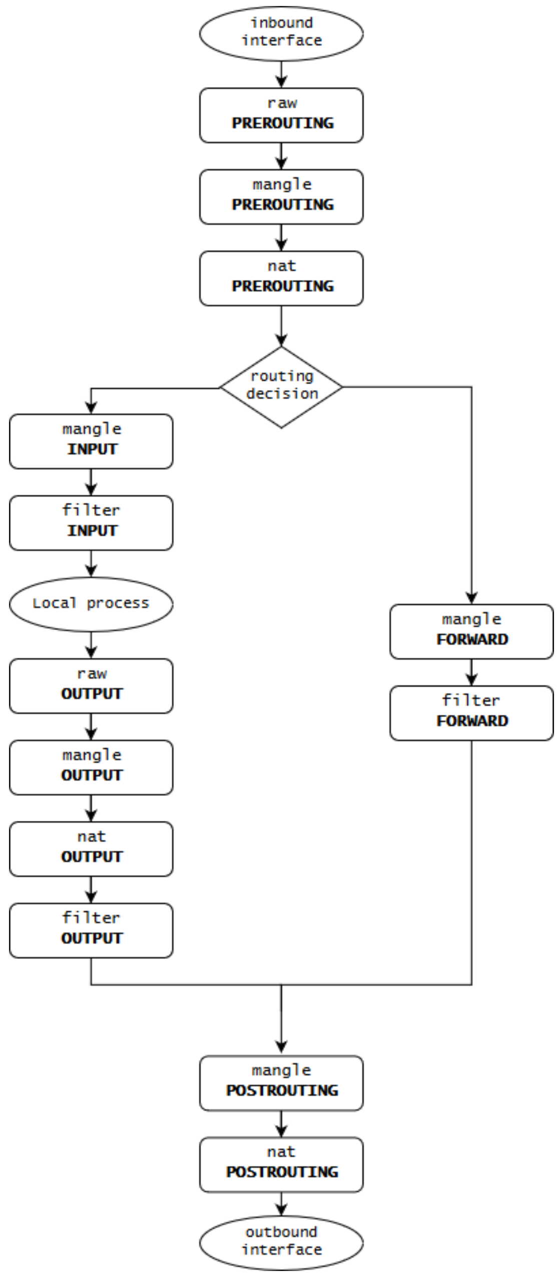


Figure 2.7: Chain traversal in *iptables*.

2.2.3 Model Definition Language

In [14], Bartal et al present a toolkit that allows high-level firewall configuration while abstracting users from the low-level firewall syntax and any vendor specific concept. This toolkit includes a Model Definition Language (MDL) that is used to define the firewall behavior.

The MDL was designed so that firewall rules are fully independent from the network topology. This is achieved by the definition of *roles*, which represent all the relevant entities governed by the firewall. The firewall administrator, the web server, company workers, the Internet, are all examples of *roles*. Having defined the *roles*, it is only necessary to establish the relationship between them. This is what defines which connections are allowed or not.

As expected, there needs to be an association between a *role* and real network hosts, where the network topology becomes relevant. Even so, the definition of *roles* and their relations being topology independent greatly improves the portability of a configuration.

One drawback from the MDL is the lack of support for NAT, which has an effect in its usefulness.

2.2.4 *hlfl*

The *High Level Firewall Language* (*hlfl*) [15] is a firewall language focused on simplicity and abstraction from the syntax used in firewalls. It supports Cisco IOS routers, *iptables*, *ipfilter* and more firewall syntaxes. Rules follow the format:

`"protocol" ("local") "operator" ("remote") ["on"] [interfaces] keywords`.

An example of a rule is: `tcp (172.22.0.1 22) <<=> (any 1000-) [ed2]`. In this rule, we allow *any* host, with source TCP port equal or greater to 1000, to establish a connection to TCP port 22 in host 172.22.0.1. This rule is only valid for traffic arriving at interface `ed2`, and allows bilateral communication after the session is established.

The main drawbacks of *hlfl* are the lack of NAT support and the fact that rules must still be correctly ordered, so a rule can not be taken at face value without checking the rest of the configuration.

2.2.4.A Additional models

Several other firewall design languages and models exist. In [16] the authors propose *FLIP*, a high level language for policy configuration, along with a translation into a set of conventional firewall rules. Shorewall [17] is an *iptables* configuration tool which also carries its own high level syntax in order to simplify the configuration process. A XML-based high level language is proposed in [18], along with a translation into *iptables* rules.

2.3 Graphical interface-based firewall configuration

Having looked at firewall design models, we will now look at existing graphical-based software that aims to facilitate firewall configuration.

2.3.1 *firewalld*

The *firewalld* daemon [19] is the result of an open-source project aiming to provide a front-end to the networking tools available on Linux like *iptables* and *ipset*. Its design is split into two layers. A *Core* layer, containing several modules, which interfaces with the existing Linux tools and a *D-Bus* layer, which provides a common Application Program Interface (API) to everything in the *Core* layer.

The focus of this section will be on *firewall-config*, one of the several applications built upon the *D-Bus* API, but the only that provides a Graphical User Interface (GUI). However, since all these applications make use of the *firewalld* daemon, the concepts we will go over can be applied to any of them.

The configuration of *firewalld* is focused around the concept of *zones* and *services*. By defining which source addresses and interfaces belong to a *zone*, a user can decide which types of traffic are allowed for devices in that zone. Zones can be defined with a default policy, which applies to all traffic that does not match any rule. Figure 2.8 shows a list of source addresses assigned to a zone, along with the *firewall-config* UI.

A *service* is defined through a combination of port numbers, protocols and destinations addresses. These allow traffic to be identified and controlled according to its purpose. Figure 2.9 shows how the `dns` service is configured.

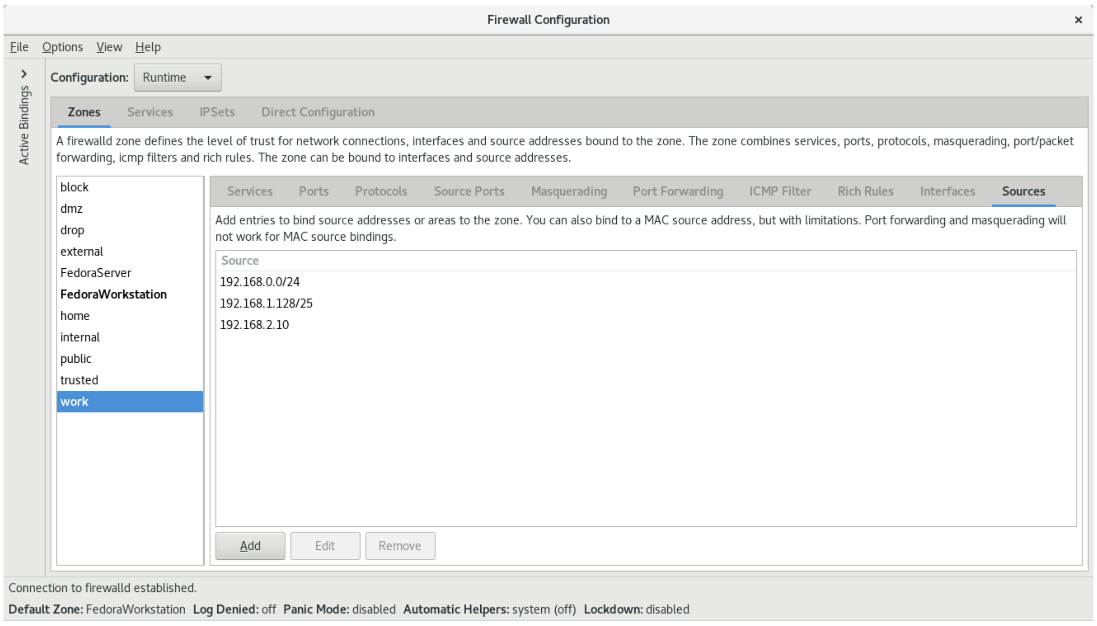
In order to restrict or modify traffic in a zone-by-zone basis, firewalld provides several different categories of rules:

Services By assigning a service to a zone, any traffic originating from that zone, which matches the service in question, is allowed through the firewall.

Source and Destination Ports A user can assign a port, or port range, to a zone. This allows any incoming traffic from that zone as long as its destination and/or source ports match the ones defined.

Protocols Layer 3 or 4 protocols can also be used to decide if incoming traffic should be allowed or not.

Masquerading This rule is implemented as an on/off toggle. If enabled, it masquerades any incoming traffic forwarded by the firewall host. Masquerading is a source address translation where address of the outbound interface is used.


Firewall Configuration
File Options View Help
Active Bindings
Zones Services IPSets Direct Configuration
Services Ports Protocols Source Ports Masquerading Port Forwarding ICMP Filter Rich Rules Interfaces Sources
Add Edit Remove

Configuration: Runtime

A firewalld zone defines the level of trust for network connections, interfaces and source addresses bound to the zone. The zone combines services, ports, protocols, masquerading, port/packet forwarding, icmp filters and rich rules. The zone can be bound to interfaces and source addresses.

block
dmz
drop
external
FedoraServer
FedoraWorkstation
home
internal
public
trusted
work

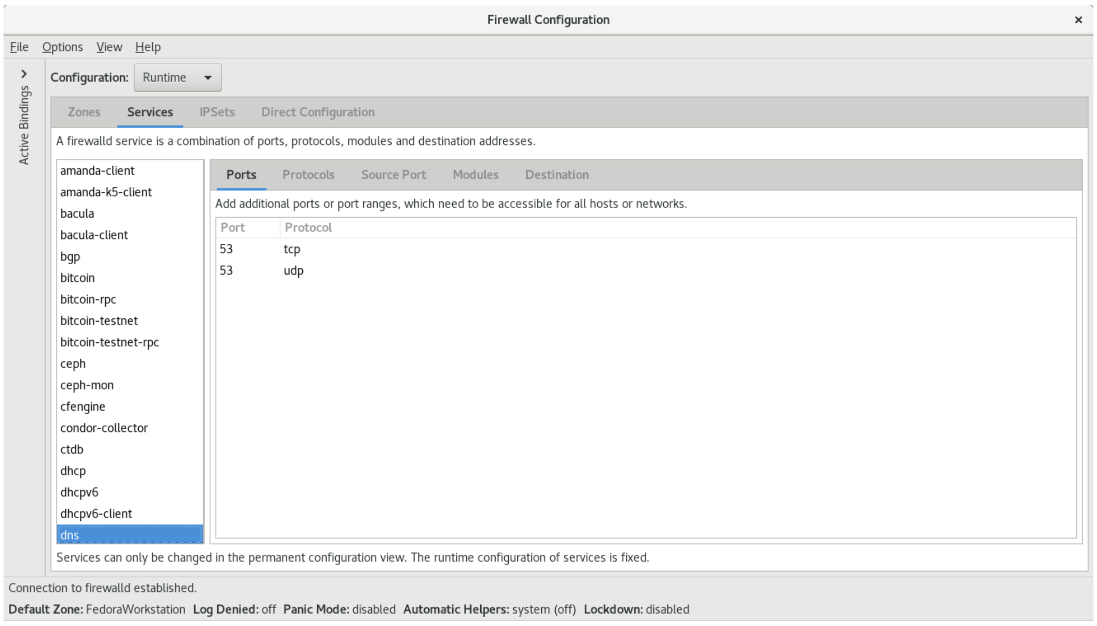
Add entries to bind source addresses or areas to the zone. You can also bind to a MAC source address, but with limitations. Port forwarding and masquerading will not work for MAC source bindings.

Source
192.168.0.0/24
192.168.1.128/25
192.168.2.10

Connection to firewalld established.

Default Zone: FedoraWorkstation **Log Denied:** off **Panic Mode:** disabled **Automatic Helpers:** system (off) **Lockdown:** disabled

Figure 2.8: Adding source addresses to a zone in *firewall-config*.


Firewall Configuration
File Options View Help
Active Bindings
Zones Services IPSets Direct Configuration
Ports Protocols Source Port Modules Destination

Configuration: Runtime

A firewalld service is a combination of ports, protocols, modules and destination addresses.

amanda-client
amanda-k5-client
bacula
bacula-client
bgp
bitcoin
bitcoin-rpc
bitcoin-testnet
bitcoin-testnet-rpc
ceph
ceph-mon
cfengine
condor-collector
ctdb
dhcp
dhcpv6
dhcpv6-client
dns

Add additional ports or port ranges, which need to be accessible for all hosts or networks.

Port	Protocol
53	tcp
53	udp

Services can only be changed in the permanent configuration view. The runtime configuration of services is fixed.

Connection to firewalld established.

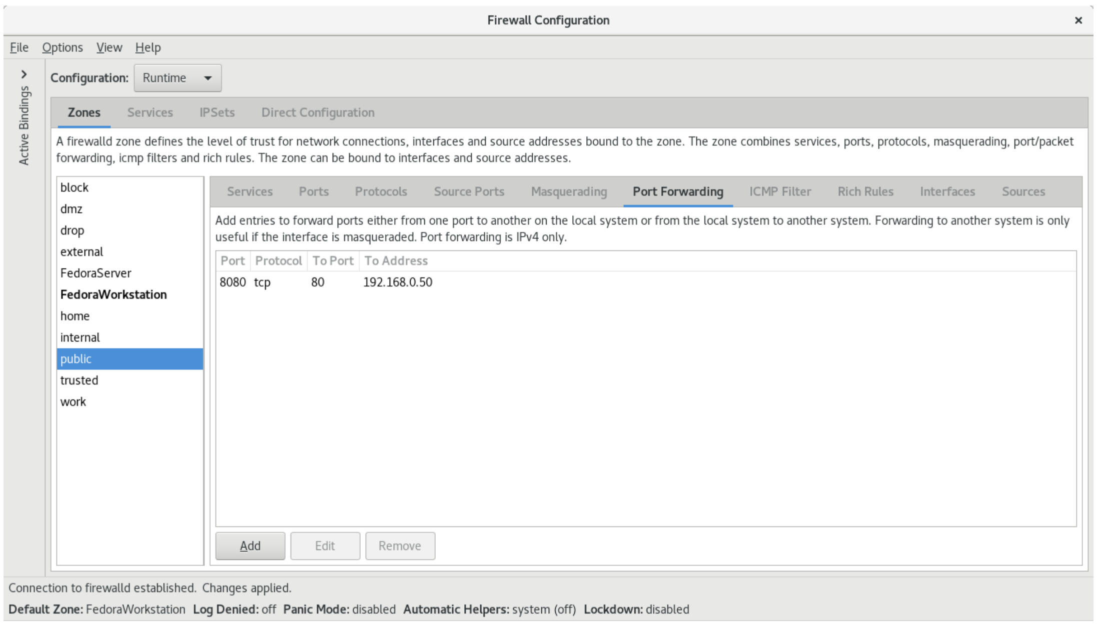
Default Zone: FedoraWorkstation **Log Denied:** off **Panic Mode:** disabled **Automatic Helpers:** system (off) **Lockdown:** disabled

Figure 2.9: Configuration of the `dns` service in *firewall-config*.

Port Forwarding This kind of rule allows a user to define address and port translation. The required arguments for such a rule are a `local_port` and a `to_port`. Any incoming traffic addressed to the `local_port` of the firewall host has its destination port translated to `to_port`. A rule can also include a `to_addr` argument, which means the destination address of the traffic also gets translated. Figure 2.10 shows an example of such a rule.

ICMP Filter These rules allow for a fine-tuning of which types of ICMP traffic should be allowed through the firewall.

Rich Rule Finally, a user can also define a *rich rule*. These rules allow a user to combine most of the previous rules, in something which looks more like a traditional firewall rule. Some options only available with rich rules are: rate-limiting, logging and auditing.


Firewall Configuration
File Options View Help
Active Bindings
Zones Services IPSets Direct Configuration
Services Ports Protocols Source Ports Masquerading Port Forwarding ICMP Filter Rich Rules Interfaces Sources
Add Edit Remove

Configuration: Runtime

A firewalld zone defines the level of trust for network connections, interfaces and source addresses bound to the zone. The zone combines services, ports, protocols, masquerading, port/packet forwarding, icmp filters and rich rules. The zone can be bound to interfaces and source addresses.

block
dmz
drop
external
FedoraServer
FedoraWorkstation
home
internal
public
trusted
work

Add entries to forward ports either from one port to another on the local system or from the local system to another system. Forwarding to another system is only useful if the interface is masqueraded. Port forwarding is IPv4 only.

Port	Protocol	To Port	To Address
8080	tcp	80	192.168.0.50

Connection to firewalld established. Changes applied.
Default Zone: FedoraWorkstation **Log Denied:** off **Panic Mode:** disabled **Automatic Helpers:** system (off) **Lockdown:** disabled

Figure 2.10: Setting up a **port forwarding** rule in *firewall-config*.

firewalld also offers a functionality named *Direct Configuration* where a user can create *iptables* chains and rules, following the regular *iptables* syntax. This is especially useful when a user requires very specific firewall rules, but is already using *firewalld*.

Overall, using *firewall-config* provides a user-friendly and simple approach to the configuration of a firewall. However, there are some aspects it could improve upon.

While the definition of a *zone* implies that a network host should not belong to multiple zones, such restriction is not imposed by the software. This leads to ambiguity when reading the configuration. One explanation we found noted that the source address of incoming traffic is always checked before its ingress interface. From that information, we can rule that if a certain packet matches zone A through its ingress interface and zone B through its source address, it will be treated as traffic from zone B. Unfortunately, if both matches had been through the source address, the *firewall-config* interface would not help solving this problem.

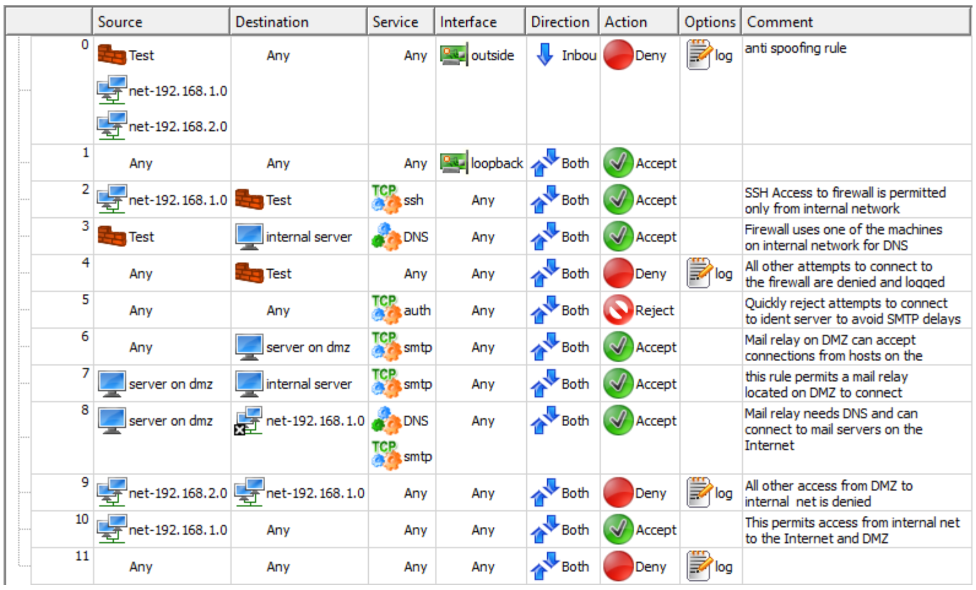
The readability of a configuration is also something that needs improvement. Our fresh configuration of *firewalld* contains eleven default zones. In *firewall-config*, each zone contains eight unique tabs, one for each kind of rule presented earlier. This gives a total of eighty-eight different screens required to check before assessing the behavior of the firewall.

2.3.2 Firewall Builder

Firewall Builder [20] is a firewall configuration tool developed by NetCitadel. It aims to standardize the configuration of firewalls across different platforms, providing a fully graphical interface that is as platform independent as possible. Its target platforms include: Cisco IOS, Cisco ASA, *iptables* and *pf*[2].

Along with the abstraction of the syntax used by the firewall, *FWBuilder* also validates rules, looking for redundant entries, and automatically deploys a configuration, if the target platform allows it.

Firewall rules in *FWBuilder* follow a very conventional format. Figure 2.11 shows a list of *policy* rules. These are all the rules in our firewall intended to accept or drop traffic, which only excludes NAT rules.

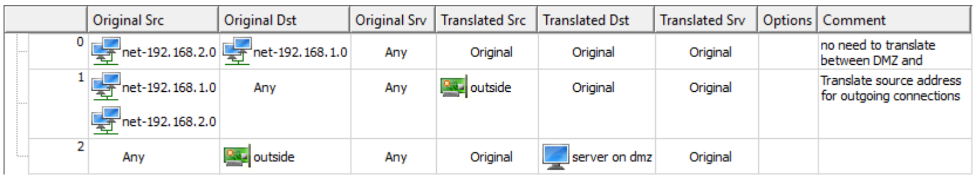


	Source	Destination	Service	Interface	Direction	Action	Options	Comment
0	Test net-192.168.1.0 net-192.168.2.0	Any	Any	outside	Inbou	Deny	log	anti spoofing rule
1	Any	Any	Any	loopback	Both	Accept		
2	net-192.168.1.0	Test	ssh	Any	Both	Accept		SSH Access to firewall is permitted only from internal network
3	Test	internal server	DNS	Any	Both	Accept		Firewall uses one of the machines on internal network for DNS
4	Any	Test	Any	Any	Both	Deny	log	All other attempts to connect to the firewall are denied and logged
5	Any	Any	auth	Any	Both	Reject		Quickly reject attempts to connect to ident server to avoid SMTP delays
6	Any	server on dmz	smtp	Any	Both	Accept		Mail relay on DMZ can accept connections from hosts on the
7	server on dmz	internal server	smtp	Any	Both	Accept		this rule permits a mail relay located on DMZ to connect
8	server on dmz	net-192.168.1.0	DNS smtp	Any	Both	Accept		Mail relay needs DNS and can connect to mail servers on the Internet
9	net-192.168.2.0	net-192.168.1.0	Any	Any	Both	Deny	log	All other access from DMZ to internal net is denied
10	net-192.168.1.0	Any	Any	Any	Both	Accept		This permits access from internal net to the Internet and DMZ
11	Any	Any	Any	Any	Both	Deny	log	

Figure 2.11: List of policy rules in *Firewall Builder*.

Implementing NAT in the firewall is equally intuitive. Figure 2.12 shows how NAT rules are organized and displayed.

[2]A firewall software similar to *iptables*. Part of the OpenBSD project.



	Original Src	Original Dst	Original Srv	Translated Src	Translated Dst	Translated Srv	Options	Comment
0	net-192.168.2.0	net-192.168.1.0	Any	Original	Original	Original		no need to translate between DMZ and
1	net-192.168.1.0 net-192.168.2.0	Any	Any	outside	Original	Original		Translate source address for outgoing connections
2	Any	outside	Any	Original	server on dmz	Original		

Figure 2.12: List of NAT rules in *Firewall Builder*.

From our experience, *FWBuilder* does a good job finding a standard interface that fits all its target platforms, while still being able to implement some platform unique features. It seems especially useful for users who need to maintain and deploy firewall configurations onto different platforms. The abstraction of the language used by the firewall is also a positive aspect of the tool.

While the tool does check for redundant rules, all other problems present in conventional firewalls can also affect a *FWBuilder* user. This is an inevitable consequence of the model to design the firewall.

3

Firewall and NAT in Cisco IOS

Contents

We will start this chapter by giving an overview of the features and capabilities of the average Cisco router. After that, the IOS features relevant to this work will be presented, along with an low-level abstraction of a router, that is a simplification of a CISCO IOS device. Since IOS is a very complex system, our abstraction will be limited to the relevant features of NAT and packet filtering.

3.1 Overview of Cisco routers and IOS

3.1.1 Router platforms and configuration interface

Cisco's catalog offers a wide array of specialized routers. From core routers with very high throughput, like the NCS 5500 platform [21], to virtual routers [22] meant to be deployed on a cloud environment, there are many options to choose from. The context of this work makes it more relevant to network edge or branch routers, which are commonly placed at the border of local networks. From the most recent Cisco catalog, the ISR 880 [23] and the ASR 1000 [24] platforms are two examples of the devices this work is targeting.

One common feature between a big majority of the available router platforms is the operating system they are running, Cisco IOS. Being developed since the 1990s, IOS became a staple of Cisco's routers and switches. As a result of this policy, almost all platforms share an equal configuration interface. The Command Line Interface (CLI) offered by IOS allows a network administrator full control over a router's features.

The CLI possesses several execution modes, each serving a different purpose. The *User EXEC* mode is the default and least privileged mode, serving mostly to consult the status of the router without revealing any possibly sensitive information. Under the *Privileged EXEC* mode, usually protected behind a password, a user can execute all commands that were disabled for a regular user. Both of the previous modes also allow users to clear tables related to network and physical protocols, like the ARP cache.

The global configuration mode can only be accessed from inside the *Privileged EXEC* mode and is where the router's running configuration can be modified. Inside this mode, a user can configure any of the features made available by the router, from routing protocols to security features.

3.1.2 IOS features

Given that IOS is deployed in a very diverse array of platforms, its list of features is too extensive for us to fully present. However, we will go over the main categories of features offered by an IOS router, giving some examples and showing how they all interact when a packet enters a router.

Starting with network-layer protocols, IOS is compatible with most routing protocols (RIP, OSPF, EIGRP, BGP) while also offering the ability to setup a DHCP or DNS server. NAT, which is very relevant to this work, is also supported by IOS. Tunneling is also supported, on both network (GRE) and data-link (L2TP) layers.

Most routers also have access to some data-link features, like 802.1Q compatibility, allowing a user to setup multiple virtual LANs. The ability to filter traffic using layer 2 properties, like a MAC address, is very common too.

Another set of features offered by IOS is related to Quality Of Service (QoS). This includes queuing algorithms, like Weighed Fair Queuing (WFQ) and its variations, which grant the ability to shape traffic and prioritize certain services.

Finally, security features are also widely available in IOS routers. Support for various forms of VPNs and IPSec tunneling capabilities is standard. Most platforms also include hardware acceleration for encryption algorithms, diminishing the performance penalty associated with them. Present alongside the previous features, and very important to this work, is the zone-based firewall. This feature allows routers to act as firewalls by policing all traffic that goes through it. Limiting remote access to the router to Secure Shell (SSH) connections is also an important security option.

3.1.3 Life cycle of a packet

When a packet enters an IOS router, all the features that have been presented need to have access to it and possibly change something about its state. This is a very complex process, given the number of features involved. Additionally, since both hardware and software platforms have evolved over time, the order in which features are applied has also changed. While [25] is the only official and public documentation we could find that explains this process, and since we are mostly interested in NAT and packet filtering functionalities, we restrict our simplified router model to the behaviour depicted in Figure 3.1.

3.2 IP access lists

Access Control Lists (ACLs) are lists of rules with the main purpose of allowing and discarding packets. They are identified using either a number or a name. IOS provides two types of ACLs, standard and extended. The former type only allows source address matching and is protocol agnostic, while the latter is protocol aware and makes use of both source and destination addresses. Due to this difference, we will use named extended ACLs throughout the remainder of this work. Each rule in an extended ACL has a well defined syntax, which can be seen in Listing 3.1. Rules are made of five mandatory parameters:

1. **Entry number:** A positive integer placed at the start of each rule, it defines the ordering of rules inside an access list. Rules are checked in increasing order.

```
ip access-list extended example_1
    10 permit ip  host 192.168.1.50                    any
    20 permit tcp 192.168.1.0 0.0.0.255                any eq ftp telnet
    30 deny   udp 192.168.1.0 0.0.0.255 range 1 10000  10.0.0.0 0.255.255.255
```

Listing 3.1: Extended access list example.

2. **Action:** Defines the action to take if a packet matches the rule in question. The only options for this field are *permit* and *deny*.

3. **Protocol:** Specify a protocol to match packets against. It is possible to pick from transport (UDP, TCP) or network (IGMP, ICMP, OSPF, ...) layer protocols. Keyword ip applies to any network protocol.

4. **Source Address:** A network address or block that should match the source address of the packet being inspected. Blocks are defined using a network address followed by a wildcard mask [1]. When referring to a single host, the keyword *host* can be placed before the address, removing the need to insert a '0.0.0.0' wildcard mask. Keyword *any* can also be used in cases where any source address serves. If TCP or UDP were specified for the rule in question, it is also possible to define intervals of ports.

5. **Destination Address:** Works the same way as the **source address** parameter, but is checked against the destination fields of the packet.

Users can also add optional parameters at the end of the rule. The *log* optional keyword is especially useful to track dropped packets. It is important to note that, if a packet is not matched by any of the visible rules in an ACL, it hits a final, implicit, `deny ip any any` rule.

As we will see in Section 3.4, the **action** parameter of an ACL has a somewhat loose meaning. When an ACL is used in the context of NAT operations, it simply serves as a way to filter which traffic should be NAT'ed by a certain NAT rule. In that context, a packet matching a *deny* rule does not get dropped, but rather goes through that NAT rule without being translated.

3.3 Zone-based firewall

The Zone-based Firewall (ZBFW) is a Cisco IOS feature providing a stateful firewall which can inspect traffic between all router interfaces, as well as local router traffic. It is not available on all base IOS images, so a security license may be needed to access it.

[1] Wildcard masks work as inverted subnet masks, meaning a 0 in the wildcard mask is the same as a 1 in a subnet mask.

```
ip access-list extended example_1
    10 permit ip host 192.168.1.50 any
    20 permit tcp 192.168.1.0 0.0.0.255 any eq ftp telnet
    30 deny udp 192.168.1.0 0.0.0.255 range 1 10000 10.0.0.0 0.255.255.255

class-map type inspect match-any example_cmap
    match access-group name example_1

policy-map type inspect example_pmap
    class type inspect example_cmap
        inspect
    class class-default
        drop log

zone-pair security zone1-zone2 source zone1 destination zone2
    service-policy type inspect example_pmap
```

Listing 3.2: ZBFW configuration example..

Zones are a fundamental concept in the ZBFW. Each interface can be assigned to a zone and one zone can support multiple interfaces. Firewall rules are applied, independently, to each directed pair of zones. The router itself is also represented by a zone, named *self*.

Firewall rules are defined using a *policy-map*, which is a structure that applies policies (*pass, inspect, drop* to specified classes of traffic, using *class-maps*. A *class-map* can use a wide variety of criteria to classify traffic. Commonly used criteria are application protocols, like HTTP, and ACLs.

By default, a *policy-map* contains a catch-all policy which drops all traffic, so all packets flowing through the firewall are dropped, unless explicitly allowed through. The *inspect* policy allows a packet to flow through the firewall and opens a pinhole for returning traffic.

For the purpose of this work, using a single ACL inside a *class-map* is enough to classify all relevant traffic, which is then applied an *inspect* policy. Listing 3.2 shows how the access-list in Listing 3.1 would be used to control traffic from *zone1* to *zone2*.

3.4 Network Address Translation

Address translation in IOS can be achieved through two different features [26]:

Legacy NAT The oldest NAT feature in IOS and also known as inside/outside NAT. Each interface must be assigned one of two domains, *inside* or *outside* and translations only occur when traffic flows from one domain to another. This asymmetry results in several constraints when defining NAT rules, the most impactful one being that source address masquerading is not available when going from an *outside* interface to an *inside* one. Another limiting constraint is the existence of only two domains when dealing with several (three or more) interfaces. In such cases, it is very likely to have two interfaces assigned to the same domain while needing to translate traffic between them.

NAT Virtual Interface (NVI) NVI was introduced to IOS in order to provide a domain-free NAT solution. It allows translations between all NAT-enabled interfaces, offering a symmetrical approach. As a result of this, none of the previous constraints is present. However, NVI does also have a downside. In legacy NAT, static NAT rules can be used in combination with an ACL, causing the translation to only occur if a packet matches the ACL. In NVI, there is no such option, meaning a static rule is applied to all relevant traffic.

After considering both options and their constraints, we decided to use NVI for our work. This decision was made mostly because of the masquerading and intra-domain constraints present in legacy NAT, which felt too restraining compared to NVI's biggest downside, the global static rules.

Regardless of the NAT feature used, translation rules in IOS can be of two kinds: *static* or *dynamic*. Static rules are always inserted as source address translation rules, but they work on both ways of traffic. Listing 3.3 shows an example of such rule. This translation can be triggered in two different ways. A source NAT occurs if a TCP packet with source address 192.168.1.50:80 goes through the router, regardless of the destination address. This source NAT translates the source address to 85.10.10.10:8080. Conversely, if the destination address of a TCP packet is 85.10.10.10:8080, it gets destination NAT'ed to 192.168.1.50:80, regardless of the origin of the packet.

```
ip nat source static tcp 192.168.1.50 80 85.10.10.10 8080
```

Listing 3.3: Static NAT entry.

Dynamic rules are mostly used for masquerading purposes. Listing 3.4 presents a masquerade NAT rule. This rule requires the creation of an ACL, which will match all traffic that should be translated. Matched traffic is then translated using interface *g1/0*'s address, with the *overload* keyword specifying that this address can be used by several hosts, by overloading its ports. The translation that occurs is always a source address translation.

```
ip nat source list acl_nat_1 interface g1/0 overload
```

Listing 3.4: Dynamic NAT entry.

3.5 Connection tracking

Both features we have presented are stateful, which means their behavior changes according to previous events.

The firewall, ZBFW, needs to keep track of open connections so it can allow returning traffic through, implementing the *inspect* option. While there is no public documentation stating the information kept by the firewall, we assume it stores the minimum it requires to function: the addresses and ports of the hosts on both ends of the connection in addition to protocol information.

On the other hand, the NAT service needs to store the addresses of both hosts, as well as the addresses they are translated to/from and any protocol information.

3.6 Order of operation

Given that IOS is very rich in features, it is necessary to establish an order in which operations are made over packets. It is important to account for all features we selected and acting on packets, considering our objective. The operations taken into consideration are: NVI NAT, ZBFW, interface ACLs and routing decisions. This order was infered from multiple public sources. Figure 3.1 shows how the previous operations act upon packets.

There are several traffic flows that we must take into consideration. Each of these flows travels through features in a different order:

1. Forwarded traffic: *INGRESS_ACL* $\rightarrow$ *DNAT* $\rightarrow$ *FIREWALL* $\rightarrow$ *SNAT* $\rightarrow$ *EGRESS_ACL*

2. Host generated traffic: *DNAT* $\rightarrow$ *FIREWALL* $\rightarrow$ *SNAT* $\rightarrow$ *EGRESS_ACL*

3. Host destined traffic: *INGRESS_ACL* $\rightarrow$ *DNAT* $\rightarrow$ *FIREWALL* $\rightarrow$ *SNAT*

It is important to note that, for all non-forwarded traffic, we will only consider traffic which goes through *DNAT* and *SNAT* unchanged. That is, we decided to only consider these flows under the condition that no translation occurs. We will not consider also any traffic from the router to itself. These decisions were prompted by the lack of public documentation explicitly referring how NAT and the firewall interact with this class of traffic.

3.7 Low-level Abstraction of a Router

Having presented IOS and its most relevant features, we will now give a formal abstract model and a formal semantics of a low-level router, that is a simplified version of a Cisco IOS router. Considering that this model will be used to match the MIGNIS semantics present in [1], we will follow an approach as similar to it as possible. In the following definitions and examples, $sa(p)$ and $da(p)$ will denote, respectively, the source and destination addresses of a packet, including port information if relevant. $prt(p)$ will also denote the protocol in the packet's header.

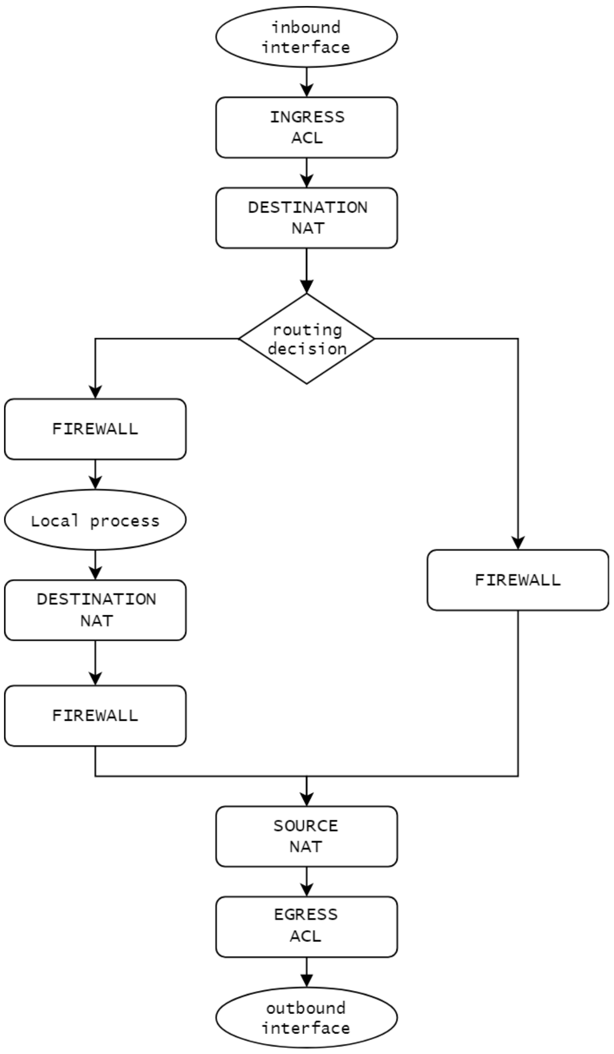


Figure 3.1: Order of operations for the relevant features in a Cisco IOS router.

3.7.1 State abstraction

We will start by abstracting the connection tracking components in IOS. In the firewall, each connection should be represented at least by a tuple $(h_A,\ h_B,\ prt)$ where h_A and h_B are the addresses of both hosts in the connection and prt is information about the protocol being use between the hosts. If the protocol in question uses ports (UDP or TCP), that information is included in h_A and h_B. A set of these tuples, s_{fw}, represents the state of our abstract firewall.

As with the firewall, the state kept by the NAT feature can also be represented by a set of tuples, s_{nat}. However, in s_{nat} we need at least to account for the translated addresses hence storing more information $(src,\ dst,\ src',\ dst',\ prt)$. In these tuples, src and dst represent the source and destinations addresses of the initial packet while src' and dst' represent the addresses of the reply packet. prt represents the protocol.

Ideally, we want to consider only one set of tuples for our abstraction of state. While the transition from a tuple in s_{fw} to a tuple in s_{nat} is simple, the problem is that s_{nat} does not need to keep information about all active connections. This is due to the fact that the NAT service only needs to keep information about connections where packets are translated, while the firewall needs to track all connections. Because of this, our global state s is populated with tuples from both sets. From the NAT state, s_{nat}, we will include every tuple, since those are the ones which hold more information. Tuples from s_{fw} are only included if the connection they represent is not in s_{nat}, which means NAT is not used. These tuples are transformed from $(h_A,\ h_B,\ prt)$ to $(h_A,\ h_B,\ h_B,\ h_A,\ prt)$ in order to match the ones from s_{nat}.

Definition 1. A packet p belongs to a connection in s if one of the following conditions holds true:

(i) $(sa(p),\ da(p),\ src,\ dst,\ prt(p)) \in s$

(ii) $(src,\ dst,\ sa(p),\ da(p),\ prt(p)) \in s$

This relation is denoted as $p \vdash_s src, dst$, where src and dst represent the source and destination addresses of the p', the expected reply to p.

3.7.2 Access lists and address translation

As we saw in Section 3.2, both firewall and NAT features in IOS are implemented using access lists. Considering this, we provide an abstraction for access lists and their entries, which we will refer to as rules.

In the following definitions, we will use the concept of address ranges. A range is defined as a set of IP addresses accompanied by a set of ports. When checking if an address $addr$ is part of an address range n, each component of $addr$ is matched against its corresponding set in n. If the set of ports in n is empty, or $addr$ does not specify a port, only the IP address needs to be matched.

Definition 2. A **basic rule** is defined as a tuple $(n_1,\ n_2,\ \phi,\ t)$, where n_1 and n_2 are address ranges, ϕ represents a stateful operation over a packet and t, the target of the rule, is either $accept$ or $drop$.

As an example, we can use the ACL in Listing 3.1 and, from its second entry, generate the following basic rule: $(192.168.1.0/24{:}*,\ *{:}\{21, 23\},\ \texttt{tcp},\ accept)$, where $*$ matches any IP address or port. We will now define how a packet matches a basic rule.

Definition 3. A packet p **matches**, in state s, a basic rule $r_i = (n_1,\ n_2,\ \phi,\ t)$ if $sa(p) \in n_1$, $da(p) \in n_2$ and $\phi(p, s)$. We denote this match with the expression $p, s \models_r r_i$.

Definition 4. Let $R = [r_1, r_2, ..., r_n]$ be a list of rules. A packet p matches R in state s, with target t, if

$$\exists_{i \leq n}\colon r_i = (n_1,\ n_2,\ \phi,\ t) \wedge p, s \models_r r_i \wedge \forall_{j<i}\, p, s \not\models_r r_j$$

This match is denoted by $p, s \models_R t$. Conversely, if a packet does not match any rule in a list, we use the expression $p, s \not\models_R$.

While a basic rule can represent any ACL entry, it does not account for translation operations, like the one represented in Listing 3.3. To represent such commands, it is necessary to differentiate between static and dynamic entries.

Definition 5. A **static translation rule** is defined as a tuple $(la,\ ga,\ \phi)$, where la and ga are ranges of addresses representing, respectively, local and global addresses, and ϕ is a stateful operation over a packet.

We can use, as an example, Listing 3.3 to build the static rule $(192.18.1.50{:}80,\ 85.10.10.10{:}8080,\ \texttt{tcp})$. A packet can match these rules in two different ways, since they are used for both source and destination translations.

Definition 6. A packet p matches, in state s, a static translation rule $t_i = (la,\ ga,\ \phi)$ if:

(i) $sa(p) \in la \land \phi(p, s)$, denoted by $p, s \models_t t_i, snat$;

(ii) $da(p) \in ga \land \phi(p, s)$, denoted by $p, s \models_t t_i, dnat$.

Static translation rules have a deterministic ordering and can be considered as being part of a list.

Definition 7. Let $T = [t_1, t_2, ..., t_n]$ be a list of static translation rules. A packet p matches T, in state s, if:

(i) $\exists_{i \leq n} \colon t_i = (la,\ ga,\ \phi) \land p, s \models_t t_i, snat \land \forall_{j<i}\, p, s \not\models_t t_j$, denoted by $p, s \models_T^S ga$;

(ii) $\exists_{i \leq n} \colon t_i = (la,\ ga,\ \phi) \land p, s \models_t t_i, dnat \land \forall_{j<i}\, p, s \not\models_t t_j$, denoted by $p, s \models_T^D la$.

The case where any of the presented conditions is not satisfied is denoted by, respectively, $p, s \not\models_T^S$ or $p, s \not\models_T^D$.

Dynamic NAT rules can only be used for source address translation. However, due to being implemented differently to static rules, they allow for a more fine-tuned packet selection. Like with static rules, dynamic rules in IOS can be considered as part of a list, with a deterministic ordering.

Definition 8. A **dynamic translation rule** is defined as a tuple $(n_1,\ n_2,\ \phi,\ t)$, where n_1 and n_2 are address ranges, ϕ represents a stateful operation over a packet and t is the range of addresses used for translation.

Definition 9. A packet p matches a dynamic translation rule $d_i = (n_1,\ n_2,\ \phi,\ t)$, in state s, if $sa(p) \in n_1$, $da(p) \in n_2$ and $\phi(p, s)$. We denote this match with the expression $p, s \models_d d_i$.

Definition 10. Let $D = [d_1, d_2, ..., d_n]$ be a list of dynamic translation rules. A packet p matches D, in state s, if:

$$\exists_{i \leq n} \colon d_i = (n_1,\ n_2,\ \phi,\ t) \wedge p, s \models_d d_i \wedge \forall_{j<i}\, p, s \not\models_d d_j$$

We denote this match with $p, s \models_D t$, using $p, s \not\models_D$ for the cases where no match is found.

The previous definitions allow us to define our low-level router abstraction that is a simplified version of a IOS router. This model will take into account both ingress and egress ACLs, the ACLs used between each directed pair of zones in the ZBFW and all possible NAT entries.

Definition 11. A low-level router $\mathcal{F}$, with n external interfaces, is composed of the following lists of rules:

(i) $2n$ lists of basic rules, I_d^i, where $d \in \{in, out\}$ and $i \in \{1, ..., n\}$. These represent the access lists in each interface.

(ii) A list of static translation rules, T, and a list of dynamic translation rules, D.

(iii) $(n+1)^2$ lists of basic rules, F_o^i, where $i, o \in \{1, ..., n\} \cup \{l\}$. These represent the rules between each zone in the firewall. A list F_y^x applies to all traffic entering in interface x and leaving through interface y. The *self* zone is represented by the letter l.

3.7.3 Semantics

In Table 3.1 we present the semantics for how our low-level router $\mathcal{F}$ deals with a packet p in state s. To help distinguish this semantics from similar ones that will appear throughout this work, we use the term ll, standing for *low-level*. Throughout this section we also use the expressions $si(p)$ and $di(p)$, which denote, respectively, the ingress and egress interfaces of a router, according to its source and destination addresses. Symbol $\mathcal{L}$ is used to denote all addresses assigned to the router's interfaces as well as any loopback interface. This definition allows us to distinguish local from non-local traffic.

The first three rules (DEst$_{ll}$, DNew$_{ll}$ and DNAT$_{ll}$) apply to packets that go through the pre-routing NAT module. This relation is denoted by $(s, p) \ \downharpoonleft_{ll}^{\delta}\ \tilde{p}$, where p is the original packet and $\tilde{p}$ is the same packet after going through the pre-routing translation module, in state s. Each different rule defines a set of required conditions for the transition to occur. Rule DEst$_{ll}$ applies to packets belonging to an already established connection, using information in s to perform the translation. In this case, the destination address of the packet is changed to the source address of the expected reply (the internal address of the destination host). On the other hand, rules DNew$_{ll}$ and DNAT$_{ll}$ apply to packets which do not belong to an established connection. The former also adds the condition that no match is found in the static translation table, T, meaning packet p goes unchanged through this transition. The latter implies that

$$\frac{p \vdash_s src, dst}{(s,p)\ \downharpoonleft^{\delta}_{ll}\ p[da \mapsto src]}\ [\mathsf{DEst}_{ll}] \qquad \frac{p \nvdash_s \quad p, s \not\models^{D}_{T}}{(s,p)\ \downharpoonleft^{\delta}_{ll}\ p}\ [\mathsf{DNew}_{ll}]$$

$$\frac{p \nvdash_s \quad p, s \models^{D}_{T} t \quad dst \in t}{(s,p)\ \downharpoonleft^{\delta}_{ll}\ p[da \mapsto dst]}\ [\mathsf{DNAT}_{ll}]$$

$$\frac{p \vdash_s src, dst}{(s,p,\tilde{p})\ \downharpoonleft^{\sigma}_{ll}\ \tilde{p}[sa \mapsto dst]}\ [\mathsf{SEst}_{ll}] \qquad \frac{p \nvdash_s \quad \tilde{p}, s \not\models^{S}_{T} \quad \tilde{p}, s \not\models_{D}}{(s,p,\tilde{p})\ \downharpoonleft^{\sigma}_{ll}\ \tilde{p}}\ [\mathsf{SNew}_{ll}]$$

$$\frac{p \nvdash_s \quad \tilde{p}, s \models^{S}_{T} t \quad src \in t}{(s,p,\tilde{p})\ \downharpoonleft^{\sigma}_{ll}\ \tilde{p}[sa \mapsto src]}\ [\mathsf{SNATs}_{ll}] \qquad \frac{p \nvdash_s \quad \tilde{p}, s \not\models^{S}_{T} \quad \tilde{p}, s \models_{D} t \quad src \in t}{(s,p,\tilde{p})\ \downharpoonleft^{\sigma}_{ll}\ \tilde{p}[sa \mapsto src]}\ [\mathsf{SNATd}_{ll}]$$

$$\frac{\begin{array}{c} sa(p) \notin \mathcal{L} \quad da(\tilde{p}) \notin \mathcal{L} \quad i \in si(p) \quad o \in di(\tilde{p}) \\ p, s \models_{I^{i}_{in}} accept \qquad (s,p)\ \downharpoonleft^{\delta}_{ll}\ \tilde{p} \qquad p \vdash_s \vee\, \tilde{p}, s \models_{F^{i}_{o}} accept \\ (s,p,\tilde{p})\ \downharpoonleft^{\sigma}_{ll}\ p' \qquad p', s \models_{I^{o}_{out}} accept \end{array}}{s \xrightarrow{p,p'}_{ll} s \uplus (p,p')}\ [\mathsf{Forward}_{ll}]$$

$$\frac{\begin{array}{c} sa(p) \notin \mathcal{L} \quad da(p) \in \mathcal{L} \quad i \in si(p) \\ p, s \models_{I^{i}_{in}} accept \qquad (s,p)\ \downharpoonleft^{\delta}_{ll}\ p\colon [\mathsf{DNew}_{ll}, \mathsf{DEst}_{ll}] \\ p \vdash_s \vee\, p, s \models_{F^{i}_{l}} accept \qquad (s,p,\tilde{p})\ \downharpoonleft^{\sigma}_{ll}\ p\colon [\mathsf{SNew}_{ll}, \mathsf{SEst}_{ll}] \end{array}}{s \xrightarrow{p,p}_{ll} s \uplus (p,p)}\ [\mathsf{Input}_{ll}]$$

$$\frac{\begin{array}{c} sa(p) \in \mathcal{L} \quad da(p) \notin \mathcal{L} \quad o \in di(p) \\ (s,p)\ \downharpoonleft^{\delta}_{ll}\ p\colon [\mathsf{DNew}_{ll}, \mathsf{DEst}_{ll}] \qquad p \vdash_s \vee\, p, s \models_{F^{l}_{o}} accept \\ (s,p,\tilde{p})\ \downharpoonleft^{\sigma}_{ll}\ p\colon [\mathsf{SNew}_{ll}, \mathsf{SEst}_{ll}] \qquad p, s \models_{I^{o}_{out}} accept \end{array}}{s \xrightarrow{p,p}_{ll} s \uplus (p,p)}\ [\mathsf{Output}_{ll}]$$

Table 3.1: Semantics of the low-level router.

a match was found in T. Rules in T are of the form $(la,\ ga,\ \phi)$ and relation $p, s \models_T^D t$ means a match was found between the global address, ga, of a rule and the destination address of the packet being matched. When such match occurs, the destination address is then translated to the local address, la, of the same rule.

The four following rules (SEst$_{ll}$, SNew$_{ll}$, SNATs$_{ll}$ and SNATd$_{ll}$) apply to packets on the post-routing NAT module, denoted by $(s, p, \tilde{p}) \ \downharpoonleft_{ll}^{\sigma} \ p'$, where p represents the original packet and $\tilde{p}$ represents the packet after going through the pre-routing NAT module. Rule SEst$_{ll}$ follows the same logic as DEst$_{ll}$, using the information in state s to translate the packet. Unlike with destination NAT, in source NAT we need to check both static and dynamic translation rules, causing a small difference in the definition of rule SNew$_{ll}$ and the existence of an additional rule, SNATd$_{ll}$, for when a dynamic translation rule is matched. From SNATs$_{ll}$ and SNATd$_{ll}$ one can also note that static translation rules take precedence over dynamic ones.

Finally, the last three rules (Forward$_{ll}$, Input$_{ll}$ and Output$_{ll}$) result in the state transition $s \xrightarrow{p,p'}_{ll} s'$, which denotes a transition from state s to s' and that packet p was accepted by the firewall and transformed into p' as the result of address translations. Each rule applies to one of the flows described earlier in Section 3.6. In all mentioned rules, state s' is defined as the result of the operation $s \uplus (p, p')$. This operation adds the connection established by packet p, translated to p', to state s. The new connection can be represented by the tuple $(sa(p), da(p), da(p'), sa(p'), prt(p))$. State s remains unchanged if the tuple already exists, which means the connection was established previously.

We can start by looking at rule Forward$_{ll}$ since it's the most complex one. For a packet p to be accepted by the firewall and translated to p' the following conditions must be met:

- $p, s \models_{I_{in}^i} accept$, it must be accepted by the inbound ACL placed at the ingress interface;
- $(s, p) \ \downharpoonleft_{ll}^{\delta} \ \tilde{p}$, the packet must go through the pre-routing NAT module. We denote the resulting packet as $\tilde{p}$, even if no translation occurred;
- $p \vdash_s \vee \ \tilde{p}, s \models_{F_o^i} accept$, it must either belong to an established connection or have its translated version, $\tilde{p}$, be explicitly accepted in F_o^i, the list of rules applied to traffic flowing from interface i to interface o;
- $(s, p, \tilde{p}) \ \downharpoonleft_{ll}^{\sigma} \ p'$, it must go through the post-routing NAT module. The resulting packet is denoted as $\tilde{p}$, even if not translated.
- $p', s \models_{I_{out}^o} accept$, the packet accepted by the post-routing NAT module, p', must be accepted by the outbound ACL placed at the egress interface.

The two remaining rules follow a similar logic. It is worth remarking that we restrict NAT translations to the Forward$_{ll}$ rule and force all other flows to be unaffected by any address translation. The syntax used to enforce this behavior is shown in $(s, p) \downharpoonright^{\delta}_{ll} p$: [DNew$_{ll}$, DEst$_{ll}$], where we mean that packet p is accepted as p by δ through rule DNew$_{ll}$ or DEst$_{ll}$. This decision was motivated by the unpredictability and lack of documentation concerning NAT for locally generated or addressed traffic.

4

MIGNIS

Contents

In [1], the authors present the MIGNIS tool. Its purpose is to assist in the configuration of `iptables`, a firewall and NAT utility available in most Linux distributions. MIGNIS' main feature is the translation from a firewall specification language, defined by the authors, to `iptables` compatible commands. To achieve the purpose of this work, we will make use of the same specification language, which we will denote as the MIGNIS language.

The MIGNIS language was designed in a way to avoid some of the problems presented in Section 2. It can be described as a declarative language, where the order between rules does not matter. Rules are also very simple to read and interpret, allowing any reader to easily understand the purpose of each rule.

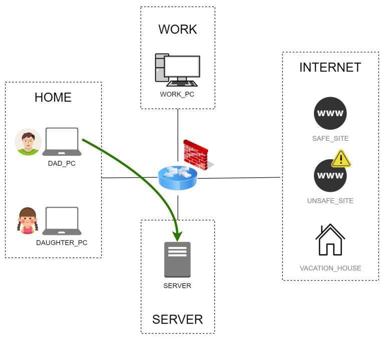
4.1 Syntax and Semantics

The four types of rule available in MIGNIS syntax are:

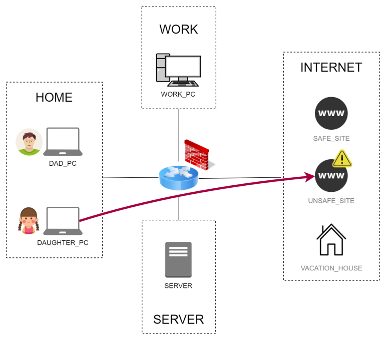
$n_1 \;/\; n_2 \mid \phi$	(DROP rule)
$n_1 > n_2 \mid \phi$	(ACCEPT rule)
$n_1 > [n_2]\; n_t \mid \phi$	(DNAT rule)
$n_1\; [n_t] > n_2 \mid \phi$	(SNAT rule)

The name given to the rules is descriptive enough to understand their purpose, but we will take a closer look at the syntax of each rule. A DROP rule forbids all traffic from n_1 to n_2, as long as ϕ is satisfied. It is important to note that this rule takes priority over any other rule in a MIGNIS configuration and even applies to packets in established connections. In practice, this means we can block incoming traffic from a malicious host even if there is an ACCEPT rule that allows it. In case any NAT occurs, we consider n_1 and n_2 as the real addresses for the hosts communicating. In other words, n_1 and n_2 are to be checked after destination NAT has occurred and before source NAT occurs. An ACCEPT rule allows n_1 to establish a connection with n_2, if ϕ holds. This rule also implicitly allows n_2 to communicate with n_1, as long as n_1 is the one starting the connection. Rules DNAT and SNAT follow the same logic, combining it with NAT operations. Rule DNAT allows n_1 to establish a connection with n_t by addressing n_2, as long as ϕ is satisfied. Rule SNAT allows n_1 to establish a connection with n_2, if ϕ holds, and applies a source address translation from n_1 to n_t to all traffic in this flow

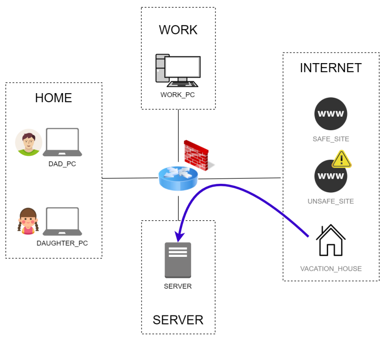
Figure 4.1 presents simple examples that can be used to demonstrate MIGNIS usage. In Figure 4.1a a rule that would allow the dad's PC to access the server would be DAD_PC $>$ SERVER. Dropping all traffic from the daughter's PC to the dangerous website, as shown in Figure 4.1b would be achieved through rule DAUGHTER_PC / UNSAFE_SITE. The destination NAT in Figure 4.1c is implementable through rule VACATION_HOME $>$ [PUBLIC_IP:22] SERVER:22. A masquerade, like the one in Figure 4.1d, can be achieved with rule HOME [.] $>$ INTERNET. One example not present in the figures concerns a double NAT. We can represent a double NAT as a merge of the DNAT and SNAT rules. Using the example in the previous figures, we would implement a double NAT from the dad's PC to the work PC as rule DAD_PC [.] $>$ [INTERNAL_IP:7878] WORK_PC:22.



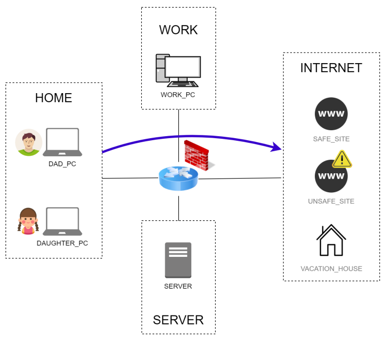
(a) Allow the dad's PC to access the server.



(b) Drop all traffic from the daughter's PC to the dangerous website.



(c) Allow the vacation house to access the server through a destination NAT.



(d) Allow and masquerade all traffic from home to the Internet.

Figure 4.1: Traffic flow examples.

The semantics for the MIGNIS language are presented in Table 4.1. For simplicity, we use a similar notation as the one in Chapter 3. A set of MIGNIS rules is defined as a configuration, denoted by C. If a RULE in C matches a packet p in state s, this is denoted by $p, s \models_C$ RULE. If the rule in question implies address translation (DNAT and SNAT rules), we also include the target address n_t in the notation. These cases are denoted by $p, s \models_C$ RULE(n_t). If a packet p in state s does not match a specific rule, we use the notation $p, s \not\models_C$ RULE.

In this table we present two new relations. $(s, p) \;\downharpoonleft^{hl}\; p'$ denotes that packet p is accepted as p' by a firewall in state s. Four different rules lead to this relation: ACCEPT$_{hl}$, DNAT$_{hl}$, SNAT$_{hl}$ and DSNAT$_{hl}$. The semantics present in those rules follow the already explained logic for the MIGNIS language. It is worth noticing that, in each of the mentioned rules, we always check if the packet does not match

$$\frac{p, s \models_C \textsf{ACCEPT} \qquad p, s \not\models_C \textsf{DROP}}{(s, p) \downharpoonleft^{hl} p} \text{[ACCEPT}_{hl}]$$

$$\frac{p, s \models_C \textsf{DNAT}(n_t) \qquad dst \in n_t \qquad p[da \mapsto dst], s \not\models_C \textsf{DROP, SNAT}}{(s, p) \downharpoonleft^{hl} p[da \mapsto dst]} \text{[DNAT}_{hl}]$$

$$\frac{p, s \models_C \textsf{SNAT}(n_t) \qquad src \in n_t \qquad p, s \not\models_C \textsf{DROP, DNAT}}{(s, p) \downharpoonleft^{hl} p[sa \mapsto src]} \text{[SNAT}_{hl}]$$

$$\frac{\begin{array}{c} p, s \models_C \textsf{DNAT}(n_t) \qquad dst \in n_t \qquad \tilde{p} = p[da \mapsto dst] \qquad \tilde{p}, s \not\models_C \textsf{DROP} \\ \tilde{p}, s \models_C \textsf{SNAT}(n'_t) \qquad src \in n'_t \end{array}}{(s, p) \downharpoonleft^{hl} \tilde{p}[sa \mapsto src]} \text{[DSNAT}_{hl}]$$

$$\frac{p \nvdash_s \qquad (s, p) \downharpoonleft^{hl} p'}{s \xrightarrow{p,p'}_{hl} s \uplus (p, p')} \text{[New}_{hl}]$$

$$\frac{p \vdash_s src, dst \qquad \tilde{p} = [da \mapsto src] \qquad \tilde{p}, s \not\models_C \textsf{DROP} \qquad p' = \tilde{p}[sa \mapsto dst]}{s \xrightarrow{p,p'}_{hl} s} \text{[Est}_{hl}]$$

Table 4.1: MIGNIS language semantics.

a DROP rule in the MIGNIS configuration, since that would always take precedence. Rule DSNAT$_{hl}$ applies to packets matching both a DNAT and an SNAT rule in the configuration.

The other relation is $s \xrightarrow{p,p'}_{hl} s'$, which denotes the firewall's state transition from state s to state s', while accepting packet p as p'. This relation is a result of the rules New$_{hl}$ and Est$_{hl}$. The first one applies to packets that do not belong to any active connection and are accepted by the firewall with relation $(s,p) \;\downharpoonleft^{hl}\; p'$. This rule implies a state change, denoted by $s \uplus (p,p')$. The second rule, Est$_{hl}$, applies to packets in established connections. In this situation, the only requirement is that the packet, after destination NAT, does not match any DROP rule in the MIGNIS configuration.

5
From MIGNIS to Cisco IOS

Contents

Having presented both the low-level firewall abstraction and the high-level MIGNIS language, we now offer a translation from a MIGNIS configuration to IOS commands. To achieve this, we will define an intermediate-level firewall, with similarities to the other two firewalls. This firewall will act as a middle step in our translation, splitting the translation into two: one from the MIGNIS semantics to our intermediate semantics, and another from this to the low-level router abstraction. We then translate the low-level router rules to CISCO IOS commands.

5.1 Intermediate Firewall

Definition 12. An intermediate firewall $\mathcal{F}_{\mathcal{I}}$ is composed by 6 sets of rules $\{S_{D_1}, S_{STT}, S_{DYN}, S_D, S_A, S_{D_2}\}$. Sets S_{D_1}, S_D and S_{D_2} contain basic rules with target $drop$ while set S_A contains basic rules with target $accept$. Set S_{STT} contains static translation rules and set S_{DYN} contains dynamic translation rules. All these rules follow the definitions established in Chapter 3.

One main difference between this firewall and the low-level one is the use of sets instead of lists. This requires us to define how a packet matches a set, since our previous low-level definitions concern lists of rules.

Definition 13. Let $S = \{r_1, r_2, ..., r_n\}$ be a set of basic rules. We define that packet p matches set S, in state s, with target t if:

$$\exists_{r_i} \in S \colon r_i = (n_1,\ n_2,\ \phi,\ t) \wedge p, s \models_r r_i$$

This relation is denoted by $p, s \models_S^{il} t$. If no match occurs, we use $p, s \not\models_S^{il}$.

Definition 14. Let $S = \{t_1, t_2, ..., t_n\}$ be a set of static translation rules. We define that packet p matches set S, in state s, with target t if:

(i) $\exists_{t_i} \in S \colon t_i = (la,\ ga,\ \phi) \wedge p, s \models_t t_i, snat$, denoted by $p, s \models_S^{il} ga, snat$

(ii) $\exists_{t_i} \in S \colon t_i = (la,\ ga,\ \phi) \wedge p, s \models_t t_i, dnat$, denoted by $p, s \models_S^{il} la, dnat$

In case one of the previous conditions is not satisfied, we write, respectively, $p, s \not\models_S^{il} snat$ or $p, s \not\models_S^{il} dnat$.

Definition 15. Let $S = \{d_1, d_2, ..., d_n\}$ be a set of dynamic translation rules. We define that packet p matches set S, in state s, with target t if:

$$\exists_{d_i} \in S \colon d_i = (n_1,\ n_2,\ \phi,\ t) \wedge p, s \models_d d_i$$

This relation is denoted by $p, s \models_S^{il} t$. If no match occurs, we use $p, s \not\models_S^{il}$.

$$\frac{p \vdash_s src, dst}{(s,p)\ \downharpoonleft_{il}^{\mathsf{DNAT}}\ p[da \mapsto src]}\ [\mathsf{DEst}_{il}] \qquad \frac{p \nvdash_s \quad p, s \nvDash^{il}_{S_{STT}} dnat}{(s,p)\ \downharpoonleft_{il}^{\mathsf{DNAT}}\ p}\ [\mathsf{DNew}_{il}]$$

$$\frac{p \nvdash_s \quad p, s \vDash^{il}_{S_{STT}} t, dnat \quad dst \in t}{(s,p)\ \downharpoonleft_{il}^{\mathsf{DNAT}}\ p[da \mapsto dst]}\ [\mathsf{DNAT}_{il}]$$

$$\frac{p \vdash_s src, dst}{(s,p,\tilde{p})\ \downharpoonleft_{il}^{\mathsf{SNAT}}\ \tilde{p}[sa \mapsto dst]}\ [\mathsf{SEst}_{il}] \qquad \frac{p \nvdash_s \quad p, s \nvDash^{il}_{S_{STT}} snat \qquad p, s \nvDash^{il}_{S_{DYN}}}{(s,p,\tilde{p})\ \downharpoonleft_{il}^{\mathsf{SNAT}}\ \tilde{p}}\ [\mathsf{SNew}_{il}]$$

$$\frac{p \nvdash_s \quad p, s \vDash^{il}_{S_{STT}} t, snat \quad src \in t}{(s,p,\tilde{p})\ \downharpoonleft_{il}^{\mathsf{SNAT}}\ \tilde{p}[sa \mapsto src]}\ [\mathsf{SNATs}_{il}] \qquad \frac{p \nvdash_s \quad p, s \nvDash^{il}_{S_{STT}} snat \quad p, s \vDash^{il}_{S_{DYN}} t \quad src \in t}{(s,p,\tilde{p})\ \downharpoonleft_{il}^{\mathsf{SNAT}}\ \tilde{p}[sa \mapsto src]}\ [\mathsf{SNATd}_{il}]$$

$$\frac{\begin{array}{c} sa(p) \notin \mathcal{L} \quad da(\tilde{p}) \notin \mathcal{L} \\ p, s \nvDash^{il}_{S_{D_1}} \quad (s,p)\ \downharpoonleft_{il}^{\mathsf{DNAT}}\ \tilde{p} \quad p \vdash_s \vee\ \tilde{p}, s \nvDash^{il}_{S_D} \\ p \vdash_s \vee\ \tilde{p}, s \vDash^{il}_{S_A} \quad (s,p,\tilde{p})\ \downharpoonleft_{il}^{\mathsf{SNAT}}\ p' \quad p', s \nvDash^{il}_{S_{D_2}} \end{array}}{s \xrightarrow{p,p'}_{il} s \uplus (p,p')}\ [\mathsf{Forward}_{il}]$$

$$\frac{\begin{array}{c} sa(p) \notin \mathcal{L} \quad da(p) \in \mathcal{L} \\ p, s \nvDash^{il}_{S_{D_1}} \quad (s,p)\ \downharpoonleft_{il}^{\mathsf{DNAT}}\ p \colon [\mathsf{DNew}_{il}, \mathsf{DEst}_{il}] \\ p \vdash_s \vee\ p, s \nvDash^{il}_{S_D} \qquad p \vdash_s \vee\ p, s \vDash^{il}_{S_A} \qquad (s,p,\tilde{p})\ \downharpoonleft_{il}^{\mathsf{SNAT}}\ p \colon [\mathsf{SNew}_{il}, \mathsf{SEst}_{il}] \end{array}}{s \xrightarrow{p,p}_{il} s \uplus (p,p)}\ [\mathsf{Input}_{il}]$$

$$\frac{\begin{array}{c} sa(p) \in \mathcal{L} \quad da(p) \notin \mathcal{L} \\ (s,p)\ \downharpoonleft_{il}^{\mathsf{DNAT}}\ p \colon [\mathsf{DNew}_{il}, \mathsf{DEst}_{il}] \qquad p \vdash_s \vee\ p, s \nvDash^{il}_{S_D} \quad p \vdash_s \vee\ p, s \vDash^{il}_{S_A} \\ (s,p,\tilde{p})\ \downharpoonleft_{il}^{\mathsf{SNAT}}\ p \colon [\mathsf{SNew}_{il}, \mathsf{SEst}_{il}] \qquad p, s \nvDash^{il}_{S_{D_2}} \end{array}}{s \xrightarrow{p,p}_{il} s \uplus (p,p)}\ [\mathsf{Output}_{il}]$$

Table 5.1: Intermediate firewall semantics.

In Table 5.1 we present the semantics associated with our intermediate level firewall. We can see many similarities to our low level firewall. Firstly, we split NAT rules into two kinds, static and dynamic. The second similarity is the inability that, after DNAT and before SNAT have occurred, we can not drop packets in established connections. This is an important difference from the MIGNIS semantics, where such drop is possible. Finally, the last similarity is that we do not consider traffic flows where NAT occurs and the router is one of the endpoints of the communication.

5.2 Translating from MIGNIS to the intermediate firewall

Table 5.2 defines the translation between rules in a MIGNIS configuration C and rules in an intermediate firewall $\mathcal{F}_\mathcal{I}$.

$$\frac{n_1 > n_2 \mid \phi}{(n_1,\ n_2,\ \phi,\ \mathit{accept}) \in S_A} \qquad \frac{n_1\ /\ n_2 \mid \phi}{(n_1,\ n_2,\ \phi,\ \mathit{drop}) \in S_{D_1}, S_D, S_{D_2}}$$

$$\frac{n_1 > [n_2]\ n_t \mid \phi\ \wedge\ n_t\ [n_2] > n_1 \mid \phi\ \wedge\ n_2 \neq \epsilon \wedge n_1 = *}{\begin{array}{c}(n_1,\ n_t,\ \phi,\ \mathit{drop}) \in S_{D_1} \\ (n_1,\ n_t,\ \phi,\ \mathit{accept}) \in S_A \quad (n_t,\ n_1,\ \phi,\ \mathit{accept}) \in S_A \\ (n_t,\ n_2,\ \phi) \in S_{STT}\end{array}}$$

$$\frac{n_1\ [n_t] > n_2 \mid \phi\ \wedge\ n_t = \epsilon}{(n_1,\ n_2,\ \phi,\ \mathit{accept}) \in S_A \quad (n_1,\ n_2,\ \phi,\ \epsilon) \in S_{DYN}}$$

Table 5.2: Translation from a MIGNIS configuration to an intermediate firewall configuration.

The translations in Table 5.2, besides the ACCEPT one, are not immediate. We will start by looking at the DROP one, which adds a drop rule to all drop sets in $\mathcal{F}_\mathcal{I}$. The reason behind this decision is the behavior of the intermediate firewall towards packets in established connections. As we can see in Table 5.1, packets belonging to established connections are immune to rules in S_D, which contradicts the higher level MIGNIS semantics. Our attempt to fix this is to drop the packet at S_{D_1}, before any DNAT, or at S_{D_2}, after all SNAT. This solution works when $\tilde{p} = p$ (no DNAT) or $\tilde{p} = p'$ (no SNAT), but fails when a packet is translated twice. As a result, we will need to forbid any drop rules that apply to the return flow of a double NAT connection. This constraint is necessary to achieve the soundness of the translation, and is enforced by condition (i) of Definition 16.

The first NAT translation only accepts DNAT and SNAT pairs. This approach was motivated by the semantics of static translation rules S_{STT} (Definition 14), that implements any DNAT as a rule that also serves as an SNAT. This is another difference between the intermediate semantic and the MIGNIS semantics, since the latter allows for non-symmetrical NAT rules (Definition 17).

One other important detail is the $drop$ rule added to S_{D_1}, which serves to block direct connections between n_1 and n_t. If this rule was not added, n_1 could establish direct connections with n_t as a result of rule $n_1 > [n_2]\ n_t \mid \phi$. Again, this carries implications in the completeness of our translation, but is required if we are to maintain its soundness (condition (i) of Definition 18).

The final translation deals with all SNAT rules where n_t is empty (represented by ϵ). It represents all masquerade SNAT rules, which are the only ones we implement as dynamic translations.

We can now define our requirements for a translation to be sound and complete. The necessity of these requirements is related to the differences between the MIGNIS and the low-level semantics. While MIGNIS comes with very flexible semantics, this is not matched by our low-level router abstraction (and consequently our target implementation CISCO IOS). To achieve soundness, we will require a MIGNIS configuration C to be *safe* and *nat-complete*.

A *safe* configuration tackles the previously presented problems where our semantics can not drop a packet that is both part of an established connection and that goes through a double NAT. We achieve this by forbidding drop rules that could possibly match a packet on the reply flow of a double NAT conversation (condition (i) of Definition 16). In Figure 5.1 we show a situation where a double NAT rule between the dad's PC and the work PC (DAD_PC [.] > [INTERNAL_IP:7878] WORK_PC:22) would conflict with a drop rule in the opposite direction (WORK_PC / DAD_PC). This combination results in an *unsafe* configuration. A *safe* configuration is also free of local to local rules (condition (ii) of Definition 16). This restriction is necessary since we do not consider this kind of packets in our intermediate semantics.

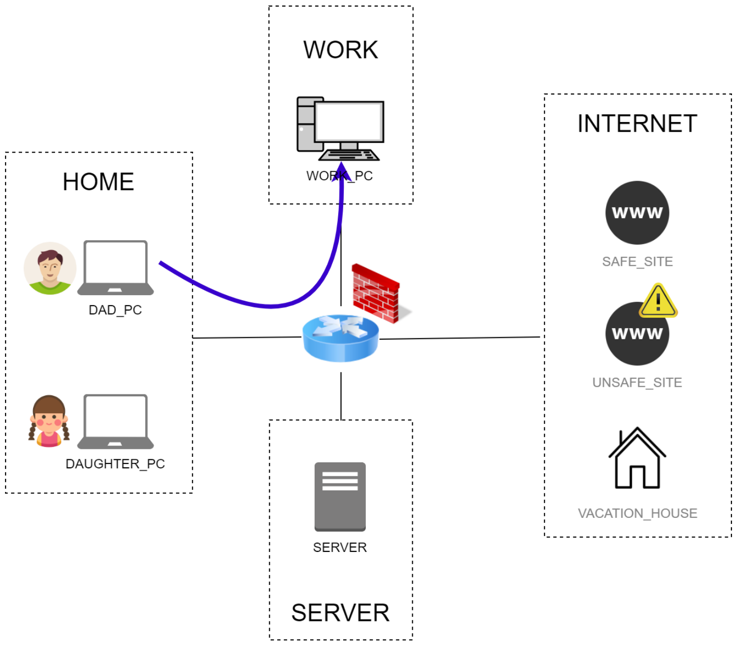


Figure 5.1: The presence of rule DAD_PC [.] > [INTERNAL_IP:7878] WORK_PC:22 does not allow rule WORK_PC / DAD_PC to be correctly implemented in our intermediate-level firewall.

Definition 16. A MIGNIS configuration C is *safe* iff:

(i) For every possible double NAT (both source and destination address translation) flow, there is no DROP rule trying to drop traffic in its reply direction. In terms of syntax, we can define this restriction in the following way: for every pair of rules $n_1 > [n_2]\ n_t \mid \phi$ and $n_1'\ [n_2'] > n_t' \mid \phi'$ in C, we define $n_1 \cap n_1' = n_a$ and $n_t \cap n_t' = n_b$. If $n_a \neq \emptyset$, $n_b \neq \emptyset$ and there exists a packet p for which $\phi(p, s)$ and $\phi'(p, s)$ both hold, then any host in n_a can initiate a double NAT connection with any host in n_b. Therefore, we require that, for every $n_x\ /\ n_y \mid \phi''$ in C, $n_x \cap n_b = \emptyset$ or $n_y \cap n_a = \emptyset$.

(ii) There is no ACCEPT, DNAT or SNAT rule where both endpoints are local addresses. This can be syntactically enforced by checking that for every $n_1 > [n_t]\ n_2 \mid \phi$, $n_1\ [n_t] > n_2 \mid \phi$ or $n_1\ > n_2 \mid \phi$ rules, in C, it holds true that $n_1 \cap \mathcal{L} = \emptyset$ or $n_2 \cap \mathcal{L} = \emptyset$.

Nat-completeness is also necessary for a sound translation. This constraint forces every NAT rule in a MIGNIS configuration to match one of the translations in Table 5.2. With this, we are sure that there are no untranslated NAT rules. This is critical to achieve soundness because a high-level NAT rule without an intermediate-level equivalent could cause a similar packet to be treated in different ways. Using the example in Figure 5.1, this means that if a rule allowing the server to be accessed from everywhere through a DNAT ($* >$ [PUBLIC_IP:22] SERVER:22) is present in our configuration, the symmetric rule must also be explicitly present (SERVER:22 [PUBLIC_IP:22] $> *$).

Definition 17. A MIGNIS configuration C is *nat-complete* iff each non-masquerade SNAT rule is accompanied by its equivalent (opposite direction) DNAT rule and vice-versa. Additionally, all non-masquerade SNAT rules must use a wildcard as the destination address and DNAT ones must use a wildcard as the source address. Syntactically, we can enforce that for each $n_t\ [n_2] > n_1 \mid \phi$ rule, where $n_2 \neq \epsilon$, n_1 should be $*$ and there must be another $n_1 > [n_2]\ n_t \mid \phi$ rule, and vice-versa.

When it comes to the completeness of our translation, we require several more constraints on our configuration C.

A *well-formed* MIGNIS configuration (Definition 18) covers several combinations of rules that could affect the completeness of a translation.

The first case (i) concerns the drop rule inserted in S_{D_1} when a DNAT rule is translated. To achieve completeness, we do not allow any other rule that would try to accept a packet matching the already placed drop rule. Figure 5.2 shows an example where the translation of a DNAT rule from the vacation house to the server (VACATION_HOME $>$ [PUBLIC_IP:22] SERVER:22) would result in an intermediate-level rule that drops any direct packets in the same path. Because of this, implementing a direct communication rule (VACATION_HOME $>$ SERVER:22) is not possible.

The second case (ii) relates to our aggressive translation of high-level DROP rules. In order to avoid accidentally dropping traffic, we need to check if any of the DROP rules is not indirectly dropping a DNAT packet before its destination address is translated or an SNAT packet after its source address has been translated. We recall that, intuitively, the addresses used in a DROP rule refer to the real addresses of endpoints and that the previous cases would result in the DROP rule taking effect when at least one of the addresses in the packet is translated.

Finally, condition (iii) restricts NAT rules to non-local traffic, since our intermediate semantics does not contemplate translated packets in non-forward rules.

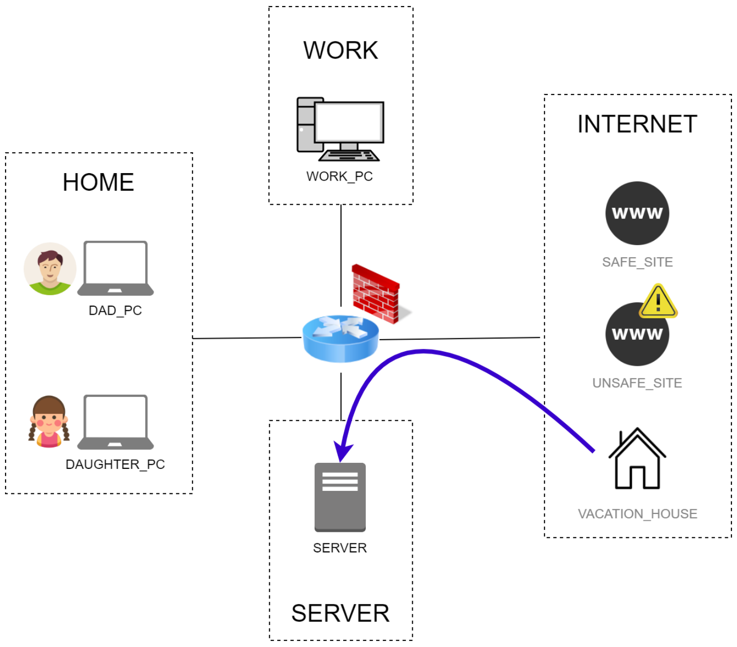


Figure 5.2: The implicit drop rule in the translation of VACATION_HOME $>$ [PUBLIC_IP:22] SERVER:22 directly contradicts rule VACATION_HOME $>$ SERVER:22.

Definition 18. A MIGNIS configuration C is *well-formed* iff:

(i) For every DNAT rule between n_a and n_b, there must be no other DNAT, SNAT or ACCEPT rule that also allows connections from any subset of n_a to any subset of n_b. Syntactically, this rule can be expressed as follows. Let $n_1 > [n_2]\ n_t \mid \phi$ be any of the DNAT rules in C. We require that $n_1 \cap n_1' = \emptyset$ or $n_t \cap n_2' = \emptyset$ for each DNAT $(n_1' > [n_2']\ n_t' \mid \phi')$, SNAT $(n_1'\ [n_t'] > n_2' \mid \phi')$ or ACCEPT $(n_1' > n_2' \mid \phi')$ rule in C. This condition is verified for all DNAT rules in C;

(ii) For every DROP rule, $n_a\ /\ n_b \mid \phi$, in C, there must be no DNAT $(n_1 > [n_2]\ n_t \mid \phi')$ or SNAT $(n_t\ [n_1] > n_2 \mid \phi')$ rules where $n_a \cap n_1 \neq \emptyset$, $n_b \cap n_2 \neq \emptyset$ and $\phi(p,s) \wedge \phi'(p,s)$ for some packet p and state s.

(iii) Every DNAT or SNAT rule in C must not use any local address as an endpoint. This is syntactically expressed by: for every $n_1 > [n_t]\ n_2 \mid \phi$ or $n_1\ [n_t] > n_2 \mid \phi$ rule, it must hold true that $n_1 \cap \mathcal{L} = \emptyset$ and $n_2 \cap \mathcal{L} = \emptyset$.

Another necessary condition for completeness is *reply-awareness*. This constraint is related to the conditions (i) and (ii) in the definition of *well-formedness*, which exist because of the translation of DNAT (i) and DROP (ii) rules indirectly dropping traffic. A *reply-aware* configuration applies this logic to traffic on the return flow of an already established connection. This makes it so that the return traffic is not accidentally dropped.

Definition 19. A MIGNIS configuration C is *reply-aware* iff:

(i) For every ACCEPT rule, $n_1 > n_2 \mid \phi$, in C, there must not be another rule indirectly dropping its reply traffic. This means there can be no DNAT rule ($n_b > [n_x]\ n_a \mid \phi'$) where $n_a \cap n_1 \neq \emptyset$, $n_b \cap n_2 \neq \emptyset$ and $\phi(p,s) \wedge \phi'(p,s)$.

(ii) For every DNAT rule, $n_1 > [n_2]\ n_t \mid \phi$, in C, there must not be any other rule indirectly dropping its reply traffic. In terms of syntax, we can say that there must not exist any DROP rule ($n_b\ /\ n_a \mid \phi'$) where $n_a \cap n_1 \neq \emptyset$, $n_b \cap n_2 \neq \emptyset$ and $\phi(p,s) \wedge \phi'(p,s)$, for some packet p and state s. Additionally, a DNAT rule ($n_b > [n_x]\ n_a \mid \phi'$) where $n_a \cap n_1 \neq \emptyset$, $n_b \cap n_t \neq \emptyset$ and $\phi(p,s) \wedge \phi'(p,s)$, for some packet p and state s, is also forbidden.

(iii) For every SNAT rule, $n_1\ [n_t] > n_2 \mid \phi$, in C, there must not be any other rule indirectly dropping its reply traffic. Syntactically, this can be expressed by saying that there must not exist any DROP rule ($n_b\ /\ n_a \mid \phi'$) where $n_a \cap n_t \neq \emptyset$, $n_b \cap n_2 \neq \emptyset$ and $\phi(p,s) \wedge \phi'(p,s)$, for some packet p and state s. DNAT rules ($n_b > [n_x]\ n_a \mid \phi'$) where $n_a \cap n_t \neq \emptyset$, $n_b \cap n_2 \neq \emptyset$ and $\phi(p,s) \wedge \phi'(p,s)$, for some packet p and state s, are also forbidden.

Our final constraint is *nat-consistency*, which removes ambiguity from a configuration, not allowing a packet to match both a translating and non-translating rule at the same time.

Definition 20. A MIGNIS configuration C is *nat-consistent* iff, for any packet p and state s we have that $p, s \models_C$ ACCEPT iff $p, s \not\models_C$ DNAT and $p, s \not\models_C$ SNAT.

We now propose a Lemma that makes use of both the translation in Table 5.2 and the previous definitions to establish a relation between rules in our high-level and intermediate-level semantics. These relations are a key part in proving the soundness and completeness of the translation.

Lemma 1. Let C be a MIGNIS configuration and $\mathcal{F}_\mathcal{I}$ an intermediate firewall. It holds true that:

(i) if $p, s \models^{il}_{S_{STT}} n_t, snat$, then $p, s \models_C$ SNAT$(n_t) \wedge n_t \neq \epsilon$;

(ii) if $p, s \models^{il}_{S_{STT}} n_t, dnat$, then $p, s \models_C$ DNAT$(n_t) \wedge n_t \neq \epsilon$;

(iii) $p, s \models^{il}_{S_{DYN}} n_t$ iff $p, s \models_C$ SNAT$(n_t) \wedge n_t = \epsilon$;

(iv) $p, s \not\models^{il}_{S_D}$, $p, s \not\models^{il}_{S_{D_2}}$ iff $p, s \not\models_C$ DROP;

(v) if $p, s \not\models^{il}_{S_{D_1}}$, then $p, s \not\models_C$ DROP;

Additionally, if C is *nat-complete*:

(vi) if $p, s \models_C$ SNAT$(n_t) \wedge n_t \neq \epsilon$, then $p, s \models^{il}_{S_{STT}} n_t, snat$;

(vii) if $p, s \models_C$ DNAT$(n_t) \wedge n_t \neq \epsilon$, then $p, s \models^{il}_{S_{STT}} n_t, dnat$.

If C is *well-formed*:

(viii) if $p, s \models_C$ DNAT, then $p, s \not\models^{il}_{S_{D_1}}$;

(ix) if $p, s \models_C$ SNAT(n_t), then $p[sa \mapsto src], s \not\models^{il}_{S_{D_2}}$, where $src \in n_t$.

Proof. We will look at (i) and (vi) simultaneously:

$$\begin{aligned}
&p, s \models^{il}_{S_{STT}} n_t, snat \\
&\iff \exists (n_1, n_t, \phi) \in S_{STT} \cdot p, s \models^{il}_{S_{STT}} (n_1, n_t, \phi), snat \\
&\iff \exists (n_1, n_t, \phi) \in S_{STT} \cdot sa(p) \in n_1 \wedge \phi(p, s) \\
&\Rightarrow / \Leftarrow^* \exists\ n_1\ [n_t] > n_2 \mid \phi \in C \cdot sa(p) \in n_1 \wedge da(p) \in n_2 \wedge \phi(p, s) \wedge n_t \neq \epsilon \\
&\iff p, s \models_C \text{SNAT}(n_t) \wedge n_t \neq \epsilon
\end{aligned}$$

All the equivalences in the above equation follow the given definitions, except for the step between the third and fourth lines, which is not obvious. The $\Rightarrow$ direction can be proved by looking at the translations in Table 5.2 and noticing that a rule in S_{STT} can only come from a pair of DNAT and SNAT rules, with $n_t \neq \epsilon$, in C. One restriction for this translation is that $n_2 = *$, which allows us to add $da(p) \in n_2$ in the fourth line. The opposite direction, $\Leftarrow^*$ is marked with a * because it only holds true when C is *nat-complete*. If C does not follow this condition, a SNAT rule may not be translated, since it may not belong to a pair of rules matching our translation requirements. However, when the condition holds, all SNAT rules are translated according to the referenced table and result in a rule in S_{STT}.

We can now do the same for (ii) and (vii):

$$\begin{aligned}
&p, s \models^{il}_{S_{STT}} n_t, dnat \\
&\iff \exists (n_t, n_1, \phi) \in S_{STT} \cdot p, s \models^{il}_{S_{STT}} (n_t, n_1, \phi), dnat \\
&\iff \exists (n_t, n_1, \phi) \in S_{STT} \cdot da(p) \in n_1 \wedge \phi(p, s) \\
&\Rightarrow / \Leftarrow^* \exists\ n_2 > [n_1]\ n_t \mid \phi \in C \cdot da(p) \in n_1 \wedge sa(p) \in n_2 \wedge \phi(p, s) \wedge n_t \neq \epsilon \\
&\iff p, s \models_C \text{DNAT}(n_t) \wedge n_t \neq \epsilon
\end{aligned}$$

The proof is similar to the previous one. The $\Leftarrow$ direction follows the translation in Table 5.2 and its associated restrictions, while $\Rightarrow^*$ relies on C being *nat-complete*.

Looking solely at (iii):

$$\begin{aligned}
&p, s \models^{il}_{S_{DYN}} n_t \\
&\iff \exists (n_1, n_2, \phi, n_t) \in S_{DYN} \cdot p, s \models^{il}_{S_{DYN}} (n_1, n_2, \phi, n_t) \\
&\iff \exists (n_1, n_2, \phi, n_t) \in S_{DYN} \cdot sa(p) \in n_1 \wedge da(p) \in n_2 \wedge \phi(p, s) \\
&\iff \exists\ n_1\ [n_t] > n_2 \mid \phi \in C \cdot sa(p) \in n_1 \wedge da(p) \in n_2 \wedge \phi(p, s) \wedge n_t = \epsilon \\
&\iff p, s \models_C \text{SNAT}(n_t) \wedge n_t = \epsilon
\end{aligned}$$

As in the previous examples, all equivalences are immediate besides the one between the third and fourth lines. In that equivalence, we rely on the translation provided in Table 5.2. Unlike in the previous examples, the equivalence does not depend on C being *nat-complete*, since SNAT masquerade rules are always translated.

Finally, we look at (iv):

$$\begin{aligned}
&p, s \not\models^{il}_{S_D} \\
&\iff \forall (n_1, n_2, \phi, t) \in S_D \cdot p, s \not\models^{il}_{S_D} (n_1, n_2, \phi, t) \\
&\iff \forall (n_1, n_2, \phi, t) \in S_D \cdot sa(p) \notin n_1 \vee da(p) \notin n_2 \vee \neg\phi(p, s) \\
&\iff \forall\ n_1\ /\ n_2 \mid \phi \in C \cdot sa(p) \notin n_1 \vee da(p) \notin n_2 \vee \neg\phi(p, s) \\
&\iff \forall\ n_1\ /\ n_2 \mid \phi \in C \cdot p, s \not\models\ n_1\ /\ n_2 \mid \phi \\
&\iff p, s \not\models_C \text{DROP}
\end{aligned}$$

All the equivalences, except the one between the third and fourth lines, follow the definitions. For this equivalence, we need to look at the translations in Table 5.2. A rule in S_D can only come from a DROP rule in C, proving the $\Rightarrow$ direction. The same translation also tells us that a DROP rule in C is always translated to S_D, which gives us $\Leftarrow$. Proving the equivalence for $p, s \not\models^{il}_{S_{D_2}}$ is exactly the same, since the set is part of the same translation.

Line (v) uses a similar proof to the one above, but between the third and fourth lines, only the $\Rightarrow$ direction holds true. This is proved by remembering that all DROP rules are translated to S_{D_1}, so if there is no rule in that set, it is implied that there was no DROP rule. However, $\Leftarrow$ does not hold true, since a packet not matching any DROP rules in C can match a rule in S_{D_1} if such rule comes from the translation of a DNAT rule.

$\blacksquare$

In Theorem 2, we formally prove the soundness and completeness of the translation in Table 5.2, along with the necessary constraints. We note that the (i) clause of the theorem concerns the soundness of the translation. This means that an *il* flow will always be captured by a *hl* flow. Such behavior is fundamental when evaluating the translation from a security standpoint as it guarantees that we are not ignoring any potential malicious flow just by considering the *hl* abstraction. On the other hand, the (ii) clause shows the completeness of the translation, which is important from a functionality standpoint, but does not affect the security aspect. An incomplete translation, would result in some of the *hl* flows missing their *il* counterparts. The following proof is a simplified version of the full proof in Appendix A.

Theorem 2. *Let $\mathcal{F}_{\mathcal{I}}$ be the intermediate firewall obtained from applying the translation in Table 5.2 to a safe and nat-complete MIGNIS configuration, C. We have that, for any packets p, p' and states s, s':*

(i) if $s \xrightarrow{p,p'}_{il} s'$, then $s \xrightarrow{p,p'}_{hl} s'$;

(ii) if C is also well-formed, reply-aware, nat-consistent and $s \xrightarrow{p,p'}_{hl} s'$, then $s \xrightarrow{p,p'}_{il} s'$.

Proof. $\boxed{(il^* \implies hl)}$ We begin by looking at (i), $s \xrightarrow{p,p'}_{il} s' \implies s \xrightarrow{p,p'}_{hl} s'$, if C is *safe* and *nat-complete*. There are three different il rules that result in $s \xrightarrow{p,p'}_{il} s'$: Forward$_{il}$ and Input$_{il}$, Output$_{il}$.

Since p, p' can be any packets and s, s' can represent any states, it is necessary to consider two different cases: [$p \vdash_s$] and [$p \nvdash_s$].

$\boxed{\text{Case } p \vdash_s}$ For rule Forward$_{il}$ we make use of the *safeness* of the configuration and Lemma 1 (line (v)) to reach the EST$_{hl}$ rule. Remaining rules Input$_{il}$ and Output$_{il}$ can reach EST$_{hl}$ by using the same logic followed for Forward$_{il}$, alongside Lemma 1.

$\boxed{\text{Case } p \nvdash_s}$ When p is not in an established connection, the only way to obtain $s \xrightarrow{p,p'}_{hl} s'$ is through the use of rule New$_{hl}$ which requires the hl transition $(s, p) \ \lrcorner^{hl} \ p'$. This transition can be the result of rules ACCEPT$_{hl}$, SNAT$_{hl}$, DNAT$_{hl}$ and DSNAT$_{hl}$. To prove this, we start with rule Forward$_{il}$ and analyze its hypothesis for several different cases: $p = \tilde{p} = p'$, $p \neq \tilde{p} \neq p'$, $p \neq \tilde{p} = p'$ and $p = \tilde{p} \neq p'$. Using both Lemma 1 and the fact that C is *nat-complete*, we reach $(s, p) \ \lrcorner^{hl} \ p'$ and, consequently, $s \xrightarrow{p,p'}_{hl} s'$. The proof for rules Input$_{il}$ and Output$_{il}$ is similar, without the requirement for a *nat-complete* configuration.

$\boxed{(hl' \implies il)}$ We now look at the Theorem's second line, which states that $s \xrightarrow{p,p'}_{hl} s' \implies s \xrightarrow{p,p'}_{il} s'$, if C is *safe*, *nat-complete*, *well-formed*, *reply-aware* and *nat-consistent*.

There are only two ways to reach $s \xrightarrow{p,p'}_{hl} s'$, rules New$_{hl}$ and Est$_{hl}$. Each one raises its hypotheses which are used to reach $s \xrightarrow{p,p'}_{il} s'$. In both cases, we can reach rules ACCEPT$_{hl}$, SNAT$_{hl}$, DNAT$_{hl}$ and DSNAT$_{hl}$. From there we can reach Forward$_{il}$. When starting from New$_{hl}$, we use Lemma 1 and the fact C is *well-formed*. With Est$_{hl}$, the fact that C is *reply-aware* is crucial, along with Lemma 1. ■

5.3 From the intermediate to low-level firewall

We can now present the translation from an intermediate firewall $\mathcal{F}_\mathcal{I}$ to a low-level router $\mathcal{F}$ with n interfaces.

Definition 21. Let $\mathcal{F}_\mathcal{I}$ be an intermediate firewall made up by sets $\{S_{D_1}, S_{STT}, S_{DYN}, S_D, S_A, S_{D_2}\}$ and let $\mathcal{F}$ be a low-level router made up by lists of basic rules I^i_d, where $d \in \{in, out\}$ and $i \in \{1, ..., n\}$, list of static translation rules T, list of dynamic rules D and lists of basic rules F^i_o, where $i, o \in \{1, ..., n\} \cup \{l\}$. The translation from $\mathcal{F}_\mathcal{I}$ to $\mathcal{F}$ is defined as follows:

(i) Let I^i_{in}, for $i \in \{1, ..., n\}$, be any possible sorting of set S_{D_1} and add basic rule $(*, *, True, accept)$ at the end of the list;

(ii) Let I^i_{out}, for $i \in \{1, ..., n\}$, be any possible sorting of set S_{D_2} and add basic rule $(*, *, True, accept)$ at the end of the list;

(iii) Let F^i_o, for $i, o \in \{1, ..., n\} \cup \{l\}$, be any possible sorting of set S_D followed by any possible sorting of set S_A and add basic rule $(*, *, True, drop)$ at the end of the list;

(iv) Let T be any possible sorting of the set S_{STT};

(v) Let D be any possible sorting of the set S_{DYN}.

Because of the similarities between both firewalls, the translation is very simple. It is important to remark that the F^i_o lists in $\mathcal{F}$ are purposefully overpopulated. The only reason this is done is to simplify both the translation and the proofs present in this work. We can propose an algorithm to simplify and make the lists more efficient.

Proposition 3. Let $\mathcal{F}$ be a low-level router with n interfaces and $sn(intf)$ to denote the subnet of interface $intf$. We can define, for $i, o \in \{1, ..., n\} \cup \{l\}$:

$$\overline{F^i_o} = \{r \mid r = (n_1, n_2, \phi, t) \in F^i_o \wedge n_1 \cap sn(i) \neq \emptyset \wedge n_2 \cap sn(o) \neq \emptyset\}$$

We can then say that $p, s \models_{F^i_o} t$ iff $p, s \models_{\overline{F^i_o}} t$.

Proof. From the ll rules in Table 3.1 it is immediate that a packet p is only matched against F^n_m if $n_1 \in sn(n) \wedge n_2 \in sn(m)$. Therefore our definition of $\overline{F^i_o}$ does not change anything for packets being matched against it. ■

Another important aspect of the translation is the transition from sets to lists. While the latter have defined ordering and imply determinism, the former could lead to non-determinism. For example, with a MIGNIS firewall C one could have $p, s \models_C$ SNAT$(n_t) \wedge p, s \models_C$ SNAT(n'_t) while $n_t \cap n'_t = \emptyset$. To tackle this disparity, we present the following definition.

Definition 22. A MIGNIS configuration C is considered *deterministic* if $p, s \models_C n_1\ [n_t] > n_2 \mid \phi \wedge p, s \models_C$ $p, s \models_C n_1'\ [n_t'] > n_2' \mid \phi'$, then we have that $n_t = n_t'$. The same must hold true for any pair of DNAT rules. A syntactic approach to this definition is to check if $n_1 \cap n_1' = \emptyset \vee n_2 \cap n_2' = \emptyset \vee n_t = n_t'$.

We can also extend this definition to our intermediate level firewall.

Definition 23. An intermediate firewall $\mathcal{F}_\mathcal{I}$ is *deterministic* if, for each pair of rules $r, r' \in S_{STT}$, where $r = (n_1, n_2, \phi, t)$ and $r = (n_1', n_2', \phi', t')$, if $p, s \models r, dnat \wedge p, s \models r', dnat$, then $t = t'$. Additionally, let $p, s \models_{il}$ SNAT(t) denote either $p, s \models^{il}_{S_{STT}} t, snat$ or $p, s \models^{il}_{S_{DYN}} t$. Then we also require that if $p, s \models_{il}$ SNAT$(t) \wedge p, s \models_{il}$ SNAT(t'), then $t = t'$.

Proposition 4. Let C be a MIGNIS firewall and $\mathcal{F}_\mathcal{I}$ be an intermediate firewall obtained from the translation of C.

- if C is *deterministic* then $\mathcal{F}_\mathcal{I}$ is *deterministic*.
- if $\mathcal{F}_\mathcal{I}$ is *deterministic* and C is *nat-complete*, then C is *deterministic*.

Proof. From Lemma 1 we have that, if C is *nat-complete*, $p, s \models^{il}_{S_{STT}} n_t, dnat$ iff $p, s \models_C$ DNAT(n_t). For SNAT we have an equivalent statement, $\tilde{p}, s \models^{il}_{S_{STT}} n_t, snat \vee \tilde{p}, s \models^{il}_{S_{DYN}} n_t$ iff $p, s \models_C$ SNAT(n_t).

Coupling these with the Definitions 22 and 23, it is immediate to note that if C is *deterministic* then $\mathcal{F}_\mathcal{I}$ is *deterministic*, and vice-versa. ■

We now present a Lemma that, similarly to Lemma 1, is used to establish a relation between rules in the *il* configuration and the *ll* configuration.

Lemma 5. Let $\mathcal{F}$ be a low-level router generated from the translation of intermediate firewall $\mathcal{F}_\mathcal{I}$ according to Definition 21. We have, for any state s and packets $p, \tilde{p}, p'$, that:

(i) $(s, p) \downarrow^{\delta}_{ll} \tilde{p} \Rightarrow / \Leftarrow^* (s, p) \downarrow^{\text{DNAT}}_{il} \tilde{p}$;

(ii) $(s, p, \tilde{p}) \downarrow^{\sigma}_{ll} p' \Rightarrow / \Leftarrow^* (s, p, \tilde{p}) \downarrow^{\text{SNAT}}_{il} p'$;

Where $\Leftarrow^*$ only holds if $\mathcal{F}_\mathcal{I}$ is a *deterministic* intermediate firewall.

Proof. DEst$_{il}$ $\iff$ DEst$_{ll}$ and SEst$_{il}$ $\iff$ SEst$_{ll}$: Since $p \vdash_s src, dst$ is defined the same way for both semantics and acts in a deterministic way, we have that both pairs of rules are equivalent.

DNew$_{il}$ $\iff$ DNew$_{ll}$ and SNew$_{il}$ $\iff$ SNew$_{ll}$: From the translation in Definition 21 we have that T is simply a random arrangement of S_{STT}, likewise for D and S_{DYN}. Therefore, if there is no match at ll, there will be no match at il, and vice-versa.

DNAT$_{il}$ $\Rightarrow^*$ DNAT$_{ll}$ and SNATs$_{il}$ $\Rightarrow^*$ SNATs$_{ll}$: If there are two rules r, r' in S_{STT} such that $p, s \models r, dnat$ and $p, s \models r', dnat$, for $r = (t, ga, \phi)$ and $r' = (t', ga', \phi')$, then we have, by *determinism*, that

$t = t'$. When matching p against T, regardless of which rule is matched first, we will have $p, s \models_T t, dnat$. The proof for SNATs uses the same logic.

DNAT$_{il}$ $\Leftarrow$ DNAT$_{ll}$ and SNATs$_{il}$ $\Leftarrow$ SNATs$_{ll}$: If $p, s \models_T t, dnat$, then it means there is a rule r in T such that $p, s \models r, dnat$. This implies $r \in S_{STT}$, according to the translation between both sets. Therefore, we have that $p, s \models^{il}_{S_{STT}} t, dnat$. Proving for SNATs is similar.

SNATd$_{il}$ $\Rightarrow^*$ / $\Leftarrow$ SNATd$_{ll}$: The proof for both directions uses the same reasoning as the ones before, the only difference being that rules are placed in D and S_{DYN}, instead of T and S_{STT}. ■

In Theorem 6, we prove the conditions for the soundness and completeness of the translation from the intermediate configuration to a low-level one. Like in Theorem 2, clause (i) is the one that is important from a security point of view, since it is the one that requires the existence of an *il* flow for any *ll* flow.

Theorem 6. *Let $\mathcal{F}$ be the low-level router obtained from applying the translation in Definition 21 to an intermediate firewall $\mathcal{F}_\mathcal{I}$. We have that, for any packets p, p' and states s, s':*

(i) if $s \xrightarrow{p,p'}_{ll} s'$, then $s \xrightarrow{p,p'}_{il} s'$;

(ii) if $\mathcal{F}_\mathcal{I}$ is deterministic and $s \xrightarrow{p,p'}_{il} s'$, then $s \xrightarrow{p,p'}_{ll} s'$.

Proof. ($ll \implies il$) We will start by proving $s \xrightarrow{p,p'}_{ll} s' \implies s \xrightarrow{p,p'}_{il} s'$. To do so, we will go through Forward$_{ll}$, Input$_{ll}$ and Output$_{ll}$.

• We start with Forward$_{ll}$, which means $sa(p) \notin \mathcal{L}$ and $da(\tilde{p}) \notin \mathcal{L}$. We have (A) $p, s \models_{I^i_{in}} accept$, (B) $(s, p) \downharpoonleft^{\delta}_{ll} \tilde{p}$, (C) $p \vdash_s \vee\, \tilde{p}, s \models_{F^i_o} accept$, (D) $(s, p, \tilde{p}) \downharpoonleft^{\sigma}_{ll} p'$ and (E) $p', s \models_{I^o_{out}} accept$, where $i = si(p)$ and $o = di(\tilde{p})$.

From the translation in Definition 21, we have that I^i_{in} is populated with all rules in S_{D_1}, followed by a rule $r_a = (*, *, True, accept)$. (A) gives that p must have matched r_a, since all rules originating from S_{D_1} have target $drop$. With this, we have $p, s \not\models^{il}_{S_{D_1}}$.

Combining (B) and Lemma 5 immediately gives $(s, p) \downharpoonleft^{\text{DNAT}}_{il} \tilde{p}$. Similarly, (D) and the same Lemma give $(s, p, \tilde{p}) \downharpoonleft^{\text{SNAT}}_{il} p'$.

From (C) one can reach $p \vdash_s \vee\, \tilde{p}, s \not\models^{il}_{S_D}$ and $p \vdash_s \vee\, \tilde{p}, s \models^{il}_{S_A}$. To do so, let us consider both possible cases. If $p \vdash_s$ the conclusion is immediate. In the case where $p \nvdash_s$, we have $\tilde{p}, s \models_{F^i_o} accept$. List F^i_o is populated with rules from both S_D (only $drop$ rules) and S_A (only $accept$ rules). All rules from S_D are placed before the ones from S_A, which means that if $\tilde{p}, s \models_{F^i_o} accept$, then $\tilde{p}$ did not match any rule from S_D, giving $\tilde{p}, s \not\models^{il}_{S_D}$. In the same way, the only rules in F^i_o with target $accept$ come from S_A, so we also have $\tilde{p}, s \models^{il}_{S_A}$.

Following the same reasoning as we did for (A), it is immediate to see that (E) gives $p', s \not\models^{il}_{S_{D_2}}$.

With this, we can apply rule Forward$_{il}$ and reach $s \xrightarrow{p,p'}_{il} s'$.

• The proof for the remaining ll rules, (Input$_{ll}$ and Output$_{ll}$,follows a similar logic as the previous one, since all these share the same similarities with their il counterparts.

$(il \Longrightarrow^* ll)$ We now prove that $s \xrightarrow{p,p'}_{il} s' \Longrightarrow s \xrightarrow{p,p'}_{ll} s'$, as long as $\mathcal{F_I}$ is *deterministic*. As before, we will go through the three rules which result in the desired state transition: Forward$_{il}$, Input$_{il}$ and Output$_{il}$.

• If Forward$_{il}$ was applied, we know that $sa(p) \notin \mathcal{L}$ and $da(\tilde{p}) \notin \mathcal{L}$. We also have (a) $p, s \not\models^{il}_{S_{D_1}}$. (b) $(s,p) \downharpoonleft^{\text{DNAT}}_{il} \tilde{p}$, (c) $p \vdash_s \vee \tilde{p}, s \not\models^{il}_{S_D}$, (d) $p \vdash_s \vee \tilde{p}, s \models^{il}_{S_A}$, (e) $(s,p,\tilde{p}) \downharpoonleft^{\text{SNAT}}_{il} p'$ and (f) $p', s \not\models^{il}_{S_{D_2}}$.

All rules in I^i_{in}, for $i \in \{1, ..., n\}$, come from S_{D_1}, except for the last one which is $r_a = (*, *, True, accept)$. From (a), we can see that p will not match any of the rules in I^i_{in} originating from S_{D_1}, which means it will reach r_a. By definition, $p, s \models_r r_a$, which gives $p, s \models_{I^i_{in}} accept$. An equivalent reasoning can be followed to reach $p', s \models_{I^o_{out}} accept$ from (f).

By Lemma 5 and (b), we have $(s,p) \downharpoonleft^{\delta}_{ll} \tilde{p}$. The same Lemma and (e) also give $(s,p,\tilde{p}) \downharpoonleft^{\sigma}_{ll} p'$. Both steps use the fact that $\mathcal{F_I}$ is *deterministic*.

The only thing left is to deduce $p \vdash_s \vee \tilde{p}, s \models_{F^i_o} accept$, for $i, o \in \{1, ..., n\}$, from (c) and (d). Like in the previous proof, if $p \vdash_s$ the conclusion is trivial. The other case leaves us with $\tilde{p}, s \not\models^{il}_{S_D}$ and $\tilde{p}, s \models^{il}_{S_A}$. Regardless of i and o, we know that F^i_o will be populated with all rules from S_D followed by all rules from S_A. From $\tilde{p}, s \not\models^{il}_{S_D}$, we have that $\tilde{p}$ will not match any of the rules in the first group. Conversely, from $\tilde{p}, s \models^{il}_{S_A}$ we have that $\tilde{p}$ will match a rule in the second group, The matched rule has target $accept$, since it originates from S_A, which gives $\tilde{p}, s \models_{F^i_o} accept$.

Having met all conditions for rule Forward$_{ll}$, we can then state $s \xrightarrow{p,p'}_{ll} s'$.

• As with the proof in the inverse direction, the reasoning used for Input$_{il}$ and Output$_{il}$ is the same as the one just presented. Again, this is due to the similarity between il and ll rules. ■

5.4 Implementation

Our implementation of the translations presented throughout this work is available as a Python program. This program is an extension of the already developed MIGNIS tool, making use of its already existing code structure and adding most IOS related code in a separate file. In the rest of this section, we will also refer to our implementation as MIGNIS.

In order to achieve its desired functionality, MIGNIS starts by taking a firewall configuration file. This file is composed of multiple sections, one of them destined to MIGNIS rules.

The first section is OPTIONS, where a user can define an address and port corresponding to the target router. As the name implies, these not mandatory parameters. In INTERFACES, a user should define all the router's relevant interfaces, along with their alias and subnet. Section ALIASES is used to assign aliases to addresses, improving the readability of the configuration. Finally, FIREWALL is where the MIGNIS rules are specified. In Appendix B, we present an example of a small configuration file and the IOS commands generated from that file.

After parsing the configuration file, MIGNIS looks for any rule, or combination of rules, that could compromise the soundness and completeness of the translation. In other words, MIGNIS verifies if the provided configuration meets all conditions required in Theorem 2 and Theorem 6. There is one thing to note about this verification, concerning the *nat-completeness* of a configuration. Due to the overly restrictive nature of this definition, we made it so that MIGNIS does not check if the destination of a SNAT rule, or the source of a DNAT rule, is equal to $*$. This change can lead to translations that are not sound, but we warn the user in such cases.

Regarding the MIGNIS rules, the accepted syntax expands upon the one presented in Section 4.1. A user can define a rule including both a destination and source translation ($n_1 \ [n_t] > [n'_t] \ n_2 \mid \phi$). Such rule is equivalent to the pair of rules $n_1 > [n'_t] \ n_2 \mid \phi$ and $n_1 \ [n_t] > n_2 \mid \phi$, but becomes much easier to interpret. The ϕ component of a rule can only be used to restrict a rule to a certain transport protocol (UDP or TCP) or network protocol. When writing a rule, it is also possible to use the alias of an interface to refer to its entire subnet.

The transformation of a *low-level* router rule into a real IOS command is as follows:

- A **basic rule** (n_1, n_2, ϕ, t) is translated into:

  $t \ \ \phi^* \ \ n_1^* \ \ n_2^*$

  If the rule is present in I_d^i, this command is placed inside the following context:

  ```
  ip access-list extended acl_intf_[alias]_[direction]
  ```

  where `[alias]` is the alias assigned to interface i and `[direction]` refers to d. n_1^* is the IOS representation of n_1, which follows the formats shown in Listing 3.1, likewise for n_2^*. As stated before, ϕ can only be used to restrict the protocol being used, so ϕ^* can either specify the protocol or, if ϕ is empty, be `ip`, meaning the rule applies to all packets.

  Firewall basic rules, like those in F_o^i, are placed inside the context:

  ```
  ip access-list extended acl_[alias_i]-[alias_o]
  ```

  where `[alias_i]` and `[alias_o]` represent the aliases of interfaces i and o, respectively.

- A **static translation rule** (la, ga, ϕ), in T, is translated into:

  ip nat source static ϕ la ga

  This command is executed at a global configuration context, separating it from any specific interface or interface pairing. In this case, ϕ is an optional parameter in the IOS command, allowing it to be empty or assume values `udp` or `tcp`.

- A **dynamic translation rule** (n_1, n_2, ϕ, t), in D, is translated into:

  ```
  ip nat source list acl_nat_[num] interface [intf] overload
  ```

This command is also executed at a global configuration context. The `acl_nat_[num]` part matches an access-list implementing the basic rule $(n_1, n_2, \phi, accept)$. The restrictions to ϕ are the same as the ones presented for basic rules, due to the similarity in implementation.

In addition to the above translations, MIGNIS runs an extra set of commands so that all used features are correctly initialized.

- For each interface `[intf]`, with alias `[alias]`, the following commands are executed in the `interface [intf]` context:

  ```
  ip access-group acl_intf_[alias]_in in
  ip access-group acl_intf_[alias]_out out
  zone-member security [alias]
  ip nat enable
  ```

  The first two commands assign the I_d^i access-lists to the matching interface. The remaining two assign the interface to a ZBFW zone and enable NAT, respectively.

- For each ZBFW zone, `[zone]`, that in our case corresponds to each interface, the following command is executed at a global context:

  ```
  zone security [zone]
  ```

  In our case, the number of ZBFW zones is the same as the number of interfaces present in the MIGNIS configuration.

- For each directed pair of ZBFW zones, `[zone_1]` and `[zone_2]`, the following commands are executed at a global context:

  ```
  class-map type inspect match-any cmap_[zone_1]-[zone_2]
   match access-group name acl_[zone_1]-[zone_2]
  policy-map type inspect pmap_[zone_1]-[zone_2]
   class type inspect cmap_[zone_1]-[zone_2]
    inspect
   class class-default
    drop log
  zone-pair security [zone_1]-[zone_2] source [zone_1] destination [zone_2]
   service-policy type inspect pmap_[zone_1]-[zone_2]
  ```

  The above commands are responsible for linking the F_o^i access-lists to their matching pair of zones.

6 Conclusion

Contents

6.1 Conclusions

Firewall specification languages have been studied and improved over time, in attempts to improve both their readability and ease of use. Throughout this work we looked at a few different approaches to this subject, focusing especially on MIGNIS. Due to its declarative nature, the language allows the configuration of a firewall through rules that do not depend on their ordering. In addition to this, MIGNIS rules have a very simple and explicit syntax. Both these features make it so that MIGNIS configurations are easy to read and to check and reason about potential security issues.

Our contribution to this problem was to expand the existing MIGNIS software and make it compatible with Cisco IOS routers. To achieve this, we presented formal definitions and semantics for both MIGNIS, the high-level language, and a low-level abstraction of a router, that is a simplification of Cisco IOS for NAT and packet filtering. We then proved the correctness and completeness of the translations used between both languages. Our proof consisted in showing that, given our proposed translation, a low-level flow implies a high-level one, and vice-versa. With this proof, we can write rules in the high-level language, MIGNIS, and be confident that our low-level router will allow no more flows than the ones allowed by the high-level semantics, while abstracting all the low-level details. The proposed translation was implemented using Python, making use of the already existing MIGNIS software.

6.2 System Limitations and Future Work

In Chapter 5 we presented the proofs that assure the soundness and completeness of our translation. These proofs require our initial MIGNIS configuration to satisfy a set of constraints. While some of the constraints are reasonable, others look too demanding. This applies especially to our definitions of *safe* and *nat-complete*.

The first one forbids a very specific, but still legitimate, use of DROP rules. Due to how the ZBFW works, it is impossible to drop packets inside an established double NAT connection. This implies that any DROP rule in the reply direction of such connection is useless, meaning we had to exclude those from valid configurations. While this constraint is undesirable, its inconvenience to a MIGNIS user seems acceptable, since it will only be relevant in very specific cases.

The second constraint stems from the fundamental differences between how both languages deal with NAT. While MIGNIS allows fine-tuned translations, NAT commands in IOS, except for the masquerade ones, always implement bidirectional translations, often leading to unwanted translations. The definition of *nat-completeness* ensures that every non-masquerade NAT statement in a MIGNIS configuration is accompanied by its symmetric counter-part. Unfortunately, this constraint significantly changes the way a MIGNIS configuration is written.

It is worth nothing that these constraints result from the fact that MIGNIS and our low/level router (and consequently CISCO IOS) have different semantics. To ensure a sound and complete translation we need to restrict our configurations to those that behave the same way in both models.

It would be interesting to see, in case of another MIGNIS extension targeted at Cisco devices, a translation into ASA commands. ASAs are dedicated firewall devices, created by Cisco, which also offer NAT functionality.

Another possible line of future work is to do the reverse path, that is, starting from a low-level configuration generate an equivalent MIGNIS configuration. This would allow sharing of existing configurations among different platforms.

Bibliography

[1] P. Adão, C. Bozzato, G. Dei Rossi, R. Focardi, and F. L. Luccio, “Mignis: A semantic based tool for firewall configuration,” *Proceedings of the Computer Security Foundations Workshop*, vol. 2014-Janua, pp. 351–365, 2014.

[2] M. G. Gouda and A. X. Liu, “Structured firewall design,” *Computer Networks*, vol. 51, no. 4, pp. 1106–1120, 2007.

[3] Gartner, “Gartner Says 8.4 Billion Connected ”Things” Will Be in Use in 2017, Up 31 Percent From 2016.” [Online]. Available: http://www.gartner.com/newsroom/id/3598917

[4] ITU, “Facts and figures,” Tech. Rep., 2017. [Online]. Available: https://www.itu.int/en/ITU-D/Statistics/Documents/facts/ICTFactsFigures2017.pdf

[5] H. Cavusoglu, B. Mishra, and S. Raghunathan, “The Effect of Internet Security Breach Announcements on Market Value: Capital Market Reactions for Breached Firms and Internet Security Developers,” *International Journal of Electronic Commerce*, vol. 9, no. 1, pp. 69–104, 2004. [Online]. Available: https://pdfs.semanticscholar.org/2390/d0ba96c89d60a15e1940c80a05f026508a39.pdf

[6] F. Mansmann and W. Cheswick, “Visual analysis of complex firewall configurations,” *Proceedings of the Ninth International Symposium on Visualization for Cyber Security - VizSec ’12*, pp. 1–8, 2012.

[7] R. Boutaba, “PolicyVis: Firewall Security Policy Visualization and Inspection,” *System*, pp. 1–16, 2008. [Online]. Available: http://en.scientificcommons.org/42283650

[8] A. El-Atawy, T. Samak, Z. Wali, E. Al-Shaer, F. Lin, C. Pham, and S. Li, “An automated framework for validating firewall policy enforcement,” *Proceedings - Eighth IEEE International Workshop on Policies for Distributed Systems and Networks, POLICY 2007*, pp. 151–160, 2007.

[9] P. Eronen and J. Zitting, “An expert system for analyzing firewall rules,” *Proceedings of the 6th Nordic Workshop on Secure IT Systems (NordSec 2001)*, p. 100–107, 2001. [Online]. Available: http://www.site.uottawa.ca/~luigi/firewalls/expertsystemfirewall.pdf

[10] secgroup, “Mignis.” [Online]. Available: https://github.com/secgroup/Mignis

[11] netfilter.org project, “iptables.” [Online]. Available: https://www.netfilter.org/

[12] A. Jeffrey and T. Samak, “Model checking firewall policy configurations,” *Proceedings - 2009 IEEE International Symposium on Policies for Distributed Systems and Networks, POLICY 2009*, no. 1, pp. 60–67, 2009.

[13] L. Yuan, H. Chen, J. Mai, C. N. Chuah, Z. Su, and P. Mohapatra, “FIREMAN: A toolkit for firewall modeling and analysis,” *Proceedings - IEEE Symposium on Security and Privacy*, vol. 2006, pp. 199–213, 2006.

[14] Y. Bartal, A. Mayer, K. Nissim, and A. Wool, “Firmato,” *ACM Transactions on Computer Systems*, vol. 22, no. 4, pp. 381–420, 2004. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1035582.1035583

[15] “High Level Firewall Language.” [Online]. Available: https://www.cusae.com/hlfl

[16] B. Zhang, E. Al-Shaer, R. Jagadeesan, J. Riely, and C. Pitcher, “Specifications of a high-level conflict-free firewall policy language for multi-domain networks,” *Proceedings of the 12th ACM symposium on Access control models and technologies - SACMAT ’07*, p. 185, 2007. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1266840.1266871

[17] “Shorewall.” [Online]. Available: http://www.shorewall.net/

[18] F. Cuppens, N. Cuppens-Boulahia, T. Sans, and A. Miège, “A Formal Approach to Specify and Deploy a Network Security Policy,” *Formal Aspects in Security and Trust*, pp. 203–218.

[19] firewalld.org project, “firewalld.” [Online]. Available: https://firewalld.org/

[20] NetCitadel, “Firewall Builder.” [Online]. Available: http://fwbuilder.sourceforge.net/

[21] Cisco, “Cisco ASR 1000 Series Aggregation Services Routers Data Sheet.” [Online]. Available: https://www.cisco.com/c/en/us/products/collateral/routers/asr-1000-series-aggregation-services-routers/datasheet-c78-731632.html

[22] ——, “Cisco Network Convergence System 5500 Series Data Sheet.” [Online]. Available: https://www.cisco.com/c/en/us/products/collateral/routers/network-convergence-system-5500-series/datasheet-c78-736270.html

[23] ——, “Cisco Cloud Services Router 1000v Data Sheet.” [Online]. Available: https://www.cisco.com/c/en/us/products/collateral/routers/cloud-services-router-1000v-series/datasheet-c78-733443.html

[24] ——, “Cisco 880 Series Integrated Services Routers - Data Sheet.” [Online]. Available: https://www.cisco.com/c/en/us/products/collateral/routers/887-integrated-services-router-isr/data_sheet_c78_459542.html

[25] ——, “NAT Order of Operation.” [Online]. Available: https://www.cisco.com/c/en/us/support/docs/ip/network-address-translation-nat/6209-5.html

[26] ——, “P Addressing: NAT Configuration Guide, Cisco IOS Release 15.” [Online]. Available: https://www.cisco.com/c/en/us/td/docs/ios-xml/ios/ipaddr_nat/configuration/15-mt/nat-15-mt-book/iadnat-addr-consv.html

A

High-level to intermediate-level Theorem Proof

Restatement of Theorem 2. Let $\mathcal{F_I}$ be the intermediate firewall obtained from applying the translation in Table 5.2 to a *safe* and *nat-complete* MIGNIS configuration, C. We have that, for any packets p, p' and states s, s':

(i) if $s \xrightarrow{p,p'}_{il} s'$, then $s \xrightarrow{p,p'}_{hl} s'$;

(ii) if C is also *well-formed*, *reply-aware*, *nat-consistent* and $s \xrightarrow{p,p'}_{hl} s'$, then $s \xrightarrow{p,p'}_{il} s'$.

Proof. $\boxed{(il^* \implies hl)}$ We begin by looking at (i), $s \xrightarrow{p,p'}_{il} s' \implies s \xrightarrow{p,p'}_{hl} s'$, if C is *safe* and *nat-complete*. There are three different il rules that result in $s \xrightarrow{p,p'}_{il} s'$: Forward$_{il}$. Input$_{il}$ and Output$_{il}$.

Since p, p' can be any packets and s, s' can represent any states, it is necessary to consider two different cases: [$p \vdash_s$] and [$p \nvdash_s$]. We will start with the former.

$\boxed{\text{Case } p \vdash_s}$ The first il rule we will look at is Forward$_{il}$. This means that $sa(p) \notin \mathcal{L}$ and $da(\tilde{p}) \notin \mathcal{L}$.

$\boxed{\text{Case } p \vdash_s \bullet \text{Forward}_{il}}$ Let's suppose that (A_1) $p \vdash_s src, dst$. Looking at Forward$_{il}$, this means we also have (B_1) $p, s \not\models^{il}_{S_{D_1}}$, (C_1) (s,p) $\downharpoonleft^{\text{DNAT}}_{il}$ $\tilde{p}$, (D_1) $(s,p,\tilde{p})$ $\downharpoonleft^{\text{SNAT}}_{il}$ p' and (E_1) $p', s \not\models^{il}_{S_{D_2}}$.

One can immediately tell from (A_1), (C_1) and (D_1) that the rules used in both NAT modules were DEst$_{il}$ and SEst$_{il}$. This means we can define $\tilde{p} = p[da \mapsto src]$ and $p' = \tilde{p}[sa \mapsto dst]$.

If $p \neq \tilde{p} \neq p'$, we need to recall that C is *safe*, which means a packet in the reply direction of a double NAT can not be subject to a DROP rule. This gives us $\tilde{p}, s \not\models_C$ DROP in the reply direction. In the original direction, we can say that if $\tilde{p}, s \models_C$ DROP, then $p \vdash_s$ would be a contradiction, since the connection would never be established. Therefore we can also conclude that $\tilde{p}, s \not\models_C$ DROP.

When $p = \tilde{p} \neq p'$, it comes from Lemma 1 (line (v)) and (B_1) that $p, s \not\models_C$ DROP, which is equal to $\tilde{p}, s \not\models_C$ DROP. If $p \neq \tilde{p} = p'$, we can apply the same logic using line (iv) from Lemma 1 and (E_1) and end up with $\tilde{p}, s \not\models_C$ DROP. The remaining case, $p = \tilde{p} = p'$ follows the same reasoning as the previous cases.

From $\tilde{p}, s \not\models_C$ DROP and (A_1) we can use rule EST$_{hl}$ to reach $s \xrightarrow{p,p'}_{hl} s'$.

$\boxed{\text{Case } p \vdash_s \bullet \text{Input}_{il}}$ Rule Input$_{il}$ has similar hypotheses, excluding (E_1), but adds additional constraints. From (s,p) $\downharpoonleft^{\text{DNAT}}_{ll}$ p: [DNew$_{il}$, DEst$_{il}$] and $(s,p,\tilde{p})$ $\downharpoonleft^{\text{SNAT}}_{ll}$ p: [SNew$_{il}$, SEst$_{il}$], we have that $p = \tilde{p} = p'$. Additionally, (A_1) gives that both NAT rules used were DEst$_{il}$ and SEst$_{il}$. Reaching $\tilde{p}, s \not\models_C$ DROP follows the same reasoning used in case $p = \tilde{p} \neq p'$ of the Forward$_{il}$ rule.

$\boxed{\text{Case } p \vdash_s \bullet \text{Output}_{il}}$ For rule Output$_{il}$ the proof is the same as the one for Input$_{il}$, but following the $p \neq \tilde{p} = p'$ case of Forward$_{il}$.

$\boxed{\text{Case } p \nvdash_s}$ When p is not in an established connection, the only way to obtain $s \xrightarrow{p,p'}_{hl} s'$ is through the use of rule New$_{hl}$ which requires the hl transition (s,p) $\downharpoonleft^{hl}$ p'. This transition can be the result of rules ACCEPT$_{hl}$, SNAT$_{hl}$, DNAT$_{hl}$ and DSNAT$_{hl}$.

$\boxed{\text{Case } p \nvdash_s \bullet \text{Forward}_{il}}$ As before, we will first look at rule Forward$_{il}$, which raises the following hypotheses: (A_2) $p \nvdash_s$, (B_2) $p, s \not\models^{il}_{S_{D_1}}$, (C_2) (s,p) $\downharpoonleft^{\text{DNAT}}_{il}$ $\tilde{p}$, (D_2) $\tilde{p}, s \not\models^{il}_{S_D}$, (E_2) $\tilde{p}, s \models^{il}_{S_A}$, (F_2)

$(s, p, \tilde{p}) \downharpoonright^{\text{SNAT}}_{il} p'$ and (G_2) $p', s \not\models^{il}_{S_{D_2}}$.

From Lemma 1 (line (iv)) and (D_2) it is immediate that (H_2) $\tilde{p}, s \not\models_C$ DROP.

$\boxed{\text{Case } p \nvdash_s \bullet \text{Forward}_{il} \bullet p = \tilde{p} = p'}$ The first case to consider is when $p = \tilde{p} = p'$. Looking at (C_2) $(s, p) \downharpoonright^{\text{DNAT}}_{il} \tilde{p}$ gives two possible cases: either p got accepted by rule DNew$_{il}$ or DNAT$_{il}$. Looking at the latter rule, we can see that $p, s \models^{il}_{S_{STT}} t, dnat$, matching a rule $(t, ga, \phi) \in S_{STT}$, and $da(p) = da(\tilde{p}) = dst \in t$. However, recalling the translations in Table 5.2 also tell us that there must have been a pair of rules in C matching $n_1 > [ga]\ t \mid \phi \ \wedge \ t\ [ga] > n_1 \mid \phi \ \wedge \ ga \neq \epsilon \wedge n_1 = *$. Such translation would also result in a rule $(n_1, t, \phi, drop) \in S_{D_1}$. Combining the previous rule with the fact that $da(p) = dst \in t$ means $p, s \models^{il}_{S_{D_1}}$, directly contradicting (B_2). As a result of this contradiction, we conclude that only rule DNew$_{il}$ could have been applied, which gives $p, s \not\models^{il}_{S_{STT}} dnat$ and, by Lemma 1 we also have that $p, s \not\models_C$ DNAT.

Having concluded that rule DNew$_{il}$ was used, we now follow the same train of thought for the SNAT translation in (F_2) $(s, p, \tilde{p}) \downharpoonright^{\text{SNAT}}_{il} p'$. There are rules that could have been used: SNew$_{il}$, SNATs$_{il}$ and SNATd$_{il}$

The first rule, SNew$_{il}$, implies that $p, s \not\models^{il}_{S_{STT}} snat$ and $p, s \not\models^{il}_{S_{DYN}}$. By Lemma 1 it stands that $p, s \not\models_C$ SNAT. From (E_2) and $p = \tilde{p}$ one can say that $p, s \models^{il}_{S_A}$, meaning there is a rule $r_a = (n_1, n_2, \phi, accept) \in S_A$ such that $p, s \models_r r_a$. Recalling the translation table, there are three possible MIGNIS translations that could have resulted in r_a. The first one would be $n_1 > n_2 \mid \phi$, where it is immediate that $p, s \models n_1 > n_2 \mid \phi$. In this case, we can use rule ACCEPT$_{hl}$ to reach $(s, p) \downharpoonright^{hl} p'$, since (H_2) and $p = \tilde{p}$ also means that $p, s \not\models_C$ DROP. Another way to reach r_a would be a pair of rules of the form $n_1 > [n_t]\ n_2 \mid \phi \ \wedge \ n_2\ [n_t] > n_1 \mid \phi \ \wedge \ n_t \neq \epsilon \wedge n_1 = *$. However, such translation would imply a rule $(n_1, n_2, \phi, drop) \in S_{D_1}$, which would match p, giving $p, s \models^{il}_{S_{D_1}}$, a direct contradiction to (B_2). A third way to reach r_a would be the translation of $n_2 > [n_t]\ n_1 \mid \phi \ \wedge \ n_1\ [n_t] > n_2 \mid \phi \ \wedge \ n_t \neq \epsilon \wedge n_1 = *$. In this case, it is immediate that $p, s \models \ n_1\ [n_t] > n_2 \mid \phi$, contradicting $p, s \not\models_C$ SNAT.

Having exhausted all the possibilities for SNew$_{il}$, it is time to look at situations where SNATs$_{il}$ is applied. We immediately have that $p, s \models^{il}_{S_{STT}} t, snat$. This means there is a rule $r_s = (la, t, \phi) \in S_{STT}$ such that $sa(p) \in la$. Recalling our translation table, the only way to reach r_S would be through the translation of $n_1 > [t]\ la \mid \phi \ \wedge \ la\ [t] > n_1 \mid \phi \ \wedge \ t \neq \epsilon \wedge n_1 = *$. Using the fact that $n_1 = *$, it becomes obvious that $p, s \models \ la\ [t] > n_1 \mid \phi$. This takes us to rule SNAT$_{hl}$ and gives $(s, p) \downharpoonright^{hl} p'$, where $p' = p[sa \mapsto src]$ and $src \in t$.

The only remaining way to achieve $(s, p, \tilde{p}) \downharpoonright^{\text{SNAT}}_{il} p'$ is by using rule SNATd$_{il}$. This rule implies that $p, s \models^{il}_{S_{DYN}} t$. We can express this by saying that there is a rule $r_d = (n_1, n_2, \phi, t) \in S_{DYN}$ and that $p, s \models_d r_d$. However, it comes from the translation table that all rules placed in S_{DYN} have $t = \epsilon$, which means $(s, p, \tilde{p}) \downharpoonright^{\text{SNAT}}_{il} p[sa \mapsto \epsilon]$. Since $p = p'$ and $sa(p) \neq \epsilon$, we can see that rule SNATd$_{il}$ would lead to a contradiction.

$\boxed{\text{Case } p \nvdash_s \bullet \text{ Forward}_{il} \bullet p \neq \tilde{p} \neq p'}$ We now look at the cases where $p \neq \tilde{p} \neq p'$. There are only two possible ways for this situation to happen. The first one is applying rules DNAT$_{il}$ and SNATs$_{il}$, while the second keeps rule DNAT$_{il}$ and applies SNATd$_{il}$.

The first case implies $p, s \models^{il}_{S_{STT}} t, dnat$ and $\tilde{p}, s \models^{il}_{S_{STT}} t', snat$. Looking at $p, s \models^{il}_{S_{STT}} t, dnat$, we can deduce that there is a rule $r_s = (t, ga, \phi) \in S_{STT}$ and that $p, s \models_s r_s$. It is mandatory that r_s must result from the translation of a rule $n_1 > [ga]\ t \mid \phi$ (accompanied by the restrictions we have shown earlier). Since $n_1 = *$, we can see that $p, s \models n_1 > [ga]\ t \mid \phi$.

Interpreting $\tilde{p}, s \models^{il}_{S_{STT}} t', snat$ is a similar process. There is a rule $r'_s = (la, t', \phi') \in S_{STT}$, which implies the presence of a MIGNIS rule $la\ [t'] > n'_1 \mid \phi'$, where $n'_1 = *$. It becomes immediate to see that $\tilde{p}, s \models\ la\ [t'] > n'_1 \mid \phi'$.

The previous two paragraphs have concluded $p, s \models_C\ DNAT(t)$ and $\tilde{p}, s \models_C\ SNAT(t')$, which, recalling (H_2) gives all conditions for rule DSNAT$_{hl}$ to be applied. Considering $dst \in t$ and $src \in t'$, we can say that $(s, p)\ \downharpoonleft^{hl}\ p' = \tilde{p}[sa \mapsto src] = p[da \mapsto dst, sa \mapsto src]$.

The second case implies $p, s \models^{il}_{S_{STT}} t, dnat$ and $\tilde{p}, s \models^{il}_{S_{DYN}} t'$. The first relation has already been explored and results in $p, s \models\ n_1 > [ga]\ t \mid \phi$.

$\tilde{p}, s \models^{il}_{S_{DYN}} t'$ tells us that there is a rule $r_d = (n'_1, n'_2, \phi', t') \in S_{DYN}$ such that $\tilde{p}, s \models_d r_d$. Such rule could only result from the translation of a MIGNIS rule $n'_1\ [t'] > n'_2 \mid \phi'$, where $t' = \epsilon$. From this follows that $\tilde{p}, s \models\ n'_1\ [t'] > n'_2 \mid \phi'$.

This case ends in the same way as the previous one, since all the conditions for rule DSNAT$_{hl}$ to be applied have been met.

$\boxed{\text{Case } p \nvdash_s \bullet \text{ Forward}_{il} \bullet p \neq \tilde{p} = p'}$ When $p \neq \tilde{p} = p'$ we can recall our previous cases and affirm that DNAT$_{il}$ was used, since $p \neq \tilde{p}$. This implies that there is a rule $r_C = n_1 > [n_2]\ n_t \mid \phi$ in C and that $p, s \models r_C$. It is also immediate that $\tilde{p} = p[da \mapsto dst]$, where $dst \in n_t$.

$\tilde{p} = p'$ can be the result of SNew$_{il}$ or SNATs$_{il}$. We exclude SNATd$_{il}$ because, as we've seen earlier, it would imply $\tilde{p} \neq p'$. If SNew$_{il}$ was applied, then $\tilde{p}, s \not\models_C$ SNAT and the hl rule leading to $(s, p)\ \downharpoonleft^{hl}\ p' = \tilde{p}$ is DNAT$_{hl}$. Else, we have that SNATs$_{il}$ was applied, this implies the existence of a rule $r'_C = n'_1\ [n'_t] > n'_2 \mid \phi'$ in C and $\tilde{p}, s \models r'_C$. In this case rule DSNAT$_{hl}$ is applied, leading to $(s, p)\ \downharpoonleft^{hl}\ \tilde{p}[sa \mapsto src]$, where $src \in n'_t$.

$\boxed{\text{Case } p \nvdash_s \bullet \text{ Forward}_{il} \bullet p = \tilde{p} \neq p'}$ The last remaining case is $p = \tilde{p} \neq p'$. From the $p = \tilde{p} = p'$ case we can immediately see that $p = \tilde{p}$ implies $p, s \not\models_C$ DNAT. Since $\tilde{p} \neq p'$, it is possible that either of SNATs$_{il}$ or SNATd$_{il}$ was applied. The former possibility was already contemplated in the $p = \tilde{p} = p'$ case, where we conclude that $p, s \models_C$ *SNAT*, which implies hl rule SNAT$_{hl}$. For rule SNATd$_{il}$ to be applied we can follow the same logic as in the second branch of $p \neq \tilde{p} \neq p'$, which also takes us to rule SNAT$_{hl}$.

$\boxed{\text{Case } p \nvdash_s \bullet \text{ Input}_{il} \circ \text{Output}_{il}}$ Rules Input$_{il}$ and Output$_{il}$ immediately give $p = \tilde{p} = p'$. From the similarity in hypotheses to the Forward$_{il}$ case, we can use its $p = \tilde{p} = p'$ branch as a proof for both these rules. It is worth noting that, unlike previously, only rules DNew$_{il}$ and SNew$_{il}$ could have been used. This means that in both input and output cases, the hl rule used must have be ACCEPT$_{hl}$.

$\boxed{(hl' \Longrightarrow il)}$ We now look at the Theorem's second line, which states that $s \xrightarrow{p,p'}_{hl} s' \Longrightarrow s \xrightarrow{p,p'}_{il} s'$, if C is *safe*, *nat-complete*, *well-formed*, *reply-aware* and *nat-consistent*.

We will start by exploring all cases where $sa(p) \notin \mathcal{L}$ and $da(p) \notin \mathcal{L}$, and then show the differences to the other cases.

There are only two ways to reach $s \xrightarrow{p,p'}_{hl} s'$, rules New$_{hl}$ and Est$_{hl}$. Each one raises its hypotheses which will be used to reach $s \xrightarrow{p,p'}_{il} s'$.

$\boxed{\text{Case New}_{hl}}$ When the hl rule used is New$_{hl}$, the hypotheses are (a_2) $p \nvdash_s$ and (b_2) $(s,p) \ \downharpoonleft^{hl} \ p'$. In the hl semantics, there are four different rules leading to $(s,p) \ \downharpoonleft^{hl} \ p'$. We will go through all four and reach $s \xrightarrow{p,p'}_{il} s'$ for each one.

$\boxed{\text{Case New}_{hl} \bullet \text{ACCEPT}_{hl}}$ Rule ACCEPT$_{hl}$ gives (a_3) $p, s \models_C$ ACCEPT, (b_3) $p, s \not\models_C$ DROP and (c_3) $p, s \not\models_C$ DNAT,SNAT. It is also immediate that $p = p'$.

Combining Lemma 1 with (b_3) gives $p, s \not\models^{il}_{S_D}$ and $p, s \not\models^{il}_{S_{D_2}}$. Adding the definition of *well-formedness* to (b_3) also means $p, s \not\models^{il}_{S_{D_1}}$. To prove the last sentence, suppose that $p, s \models^{il}_{S_{D_1}}$, then there is a $r_i = (n_1, n_2, \phi, drop)$ in S_{D_1} such that $p, s \models r_i$. From (b_3) it is clear that r_i did not originate from a DROP rule, since then $p, s \models_C$ DROP, a direct contradiction to (b_3). The only other way to generate r_i would be from the translation of a DNAT rule $n_1 > [n_t] \ n_2 \mid \phi$, which goes against the *well-formedness* of C.

From (c_3) and Lemma 1 we have $p, s \not\models^{il}_{S_{STT}} dnat$, $p, s \not\models^{il}_{S_{STT}} snat$ and $p, s \not\models^{il}_{S_{DYN}}$. Adding (a_2) we can apply rules DNew$_{il}$ and SNew$_{il}$, giving $\downharpoonleft^{\text{DNAT}}_{il} \ \tilde{p}$ and $(s, p, \tilde{p}) \ \downharpoonleft^{\text{SNAT}}_{il} \ p'$, where $p = \tilde{p} = p'$.

From (a_3) we have that there is a rule $r_a = n'_1 > n'_2 \mid \phi'$ in C such that $p, s \models_C r_a$. Since r_a is translated to S_A as $(n'_1, n'_2, \phi', accept)$, we also have that $p, s \models^{il}_{S_A}$.

Having met all requisites for rule Forward$_{il}$, we reach $s \xrightarrow{p,p'}_{il} s'$.

• For cases where either $sa(p) \in \mathcal{L}$ or $da(p) \in \mathcal{L}$, the proof is the same, resulting in Output$_{il}$ or Input$_{il}$, respectively. Both these rules can only be applied if $p = \tilde{p} = p'$, which is the case. The case where $sa(p) \in \mathcal{L}$ and $da(p) \in \mathcal{L}$ is disregarded, since the *safeness* of C does not allow local rules.

For all remaining hl rules, we will only consider cases where $sa(p) \notin \mathcal{L}$ and $da(p) \notin \mathcal{L}$. Since C is *well-formed*, it is clear that the following rules can not apply when $sa(p) \in \mathcal{L}$ or $da(p) \in \mathcal{L}$, as that would contradict the third clause in the definition of *well-formedness*.

$\boxed{\text{Case New}_{hl} \bullet \text{DNAT}_{hl}}$ If DNAT$_{hl}$ is applied, then we have that (a_4) $p, s \models_C$ DNAT(n_t), (b_4) $p', s \not\models_C$ DROP and (c_4) $p', s \not\models_C$ SNAT, for $p' = p[da \mapsto dst]$ where $dst \in n_t$.

Recalling that C is *well-formed*, and with (a_4), we have that $p, s \not\models^{il}_{S_{D_1}}$ by Lemma 1. By the same Lemma and (b_4) we have $p', s \not\models^{il}_{S_D}$ and $p', s \not\models^{il}_{S_{D_2}}$.

From (a_4) we know there is a rule $r_n = n_1 > [n_2] \ n_t \mid \phi$ in C such that $p, s \models_C r_n$. By Lemma 1 we have $p, s \models^{il}_{S_{STT}} n_t, dnat$, leading to rule DNAT$_{il}$ and $\downarrow^{\text{DNAT}}_{il}\ \tilde{p}$, where $\tilde{p} = p[da \mapsto dst] = p'$, for $dst \in n_t$. From $p' = \tilde{p}$ we also have $\tilde{p}, s \not\models^{il}_{S_D}$, as we already had that $p', s \not\models^{il}_{S_D}$.

By Lemma 1, (c_4) and $p' = \tilde{p}$ we have $\tilde{p}, s \not\models^{il}_{S_{STT}} snat$ and $\tilde{p}, s \not\models^{il}_{S_{DYN}}$. Combining (a_2) we can apply rule SNew$_{il}$ and reach $(s, p, \tilde{p})\ \downarrow^{\text{SNAT}}_{il}\ p' = \tilde{p}$.

The translation of r_n results in a rule $(n_1, n_t, \phi, accept)$ being placed in S_A. Since $\tilde{p} = p[da \mapsto dst]$ and $dst \in n_t$, we can see that if $p, s \models_C r_n$ then $\tilde{p}, s \models^{il}_{S_A}$.

This completes all conditions for Forward$_{il}$ and $s \xrightarrow{p,p'}_{il} s'$.

$\boxed{\text{Case New}_{hl} \bullet \text{SNAT}_{hl}}$ Applying SNAT$_{hl}$ gives the following hypotheses: $(a_5)\ p, s \models_C$ SNAT(n_t), $(b_5)\ p, s \not\models_C$ DROP and $(c_5)\ p, s \not\models_C$ DNAT, for $p' = p[sa \mapsto src]$ where $src \in n_t$.

Combining (a_4) with Lemma 1 and C's *well-formedness* gives $p', s \not\models^{il}_{S_{D_2}}$. Additionally, with (b_5) we have that $p, s \not\models^{il}_{S_D}$. One additional conclusion is that $p, s \not\models^{il}_{S_{D_1}}$. To prove this last sentence, we recall that any rule in S_{D_1} results of the translation of either a DNAT or a DROP rule. From (a_5) and the first clause in the definition of *well-formedness*, it comes that no DNAT translation will result in a rule that can drop p in S_{D_1}. From (b_5) we also have that the translation of any DROP rule to S_{D_1} will not drop p.

From (c_3) and Lemma 1 one can reach $p, s \not\models^{il}_{S_{STT}} dnat$, which coupled with (a_2) gives $\downarrow^{\text{DNAT}}_{il}\ \tilde{p} = p$ through rule DNew$_{il}$. This gives $\tilde{p}, s \not\models^{il}_{S_D}$, since the same conclusion had already been reached for p.

(a_5) implies there is a rule $r_s = n_1\ [n_t] > n_2 \mid \phi$ in C such that $p, s \models_C r_s$. By Lemma 1, this gives $\tilde{p}, s \models^{il}_{S_{STT}} n_t, snat \vee \tilde{p}, s \models^{il}_{S_{DYN}} n_t$. Depending on which one is true, we can apply one of SNATs$_{il}$ and SNATd$_{il}$ to reach $(s, p, \tilde{p})\ \downarrow^{\text{SNAT}}_{il}\ p' = p[sa \mapsto src]$, for $src \in n_t$.

Translating rule r_S results in a rule $(n_1, n_2, \phi, accept)$ in S_A. Since $p, s \models_C r_s$ and $\tilde{p} = p$, it is immediate that $\tilde{p}, s \models^{il}_{S_A}$.

From all this follows $s \xrightarrow{p,p'}_{il} s'$, by rule Forward$_{il}$.

$\boxed{\text{Case New}_{hl} \bullet \text{DSNAT}_{hl}}$ When rule DSNAT$_{hl}$ is applied, we have hypotheses $(a_6)\ p, s \models_C$ DNAT(n_t), $(b_6)\ \tilde{p}, s \not\models_C$ DROP and $(c_6)\ \tilde{p}, s \models_C$ SNAT(n'_t), for $p' = p[sa \mapsto src, da \mapsto dst]$ and $\tilde{p} = p[da \mapsto dst]$ where $dst \in n_t$ and $src \in n'_t$.

By (a_6) and Lemma 1 we have that $p, s \not\models^{il}_{S_{D_1}}$. This conclusion relies on C being *well-formed*. As seen in the previous case, the first clause of *well-formedness* assures that no rule in S_{D_1}, resulting from a hl DNAT rule, will match p. The second clause of the definition covers the translation of DROP rules, making it so that no hl DROP rule will match p. This covers all rules in S_{D_1}, allowing us to reach the previous conclusion.

Also by Lemma 1, (b_6) and (c_6) give us $\tilde{p}, s \not\models^{il}_{S_D}$ and $p', s \not\models^{il}_{S_{D_2}}$, the latter relying on C's *well-formedness*.

From (a_6) there must be a rule $r_d = n_1 > [n_2]\ n_t \mid \phi$ in C such that $p, s \models_C r_d$. By Lemma 1 we have $p, s \models^{il}_{S_{STT}} n_t, dnat$, leading to rule DNAT$_{il}$ and $\downarrow^{\text{DNAT}}_{il}\ \tilde{p}$.

Also from r_d we can say there is a rule $(n_1, n_t, \phi, accept)$ in S_A. Since $\tilde{p} = p[da \mapsto dst]$ and $dst \in n_t$, we can see that if $p, s \models_C r_s$ then $\tilde{p}, s \models^{il}_{S_A}$.

(c_6) implies there must be a rule $r_s = n'_1\ [n'_t] > n'_2 \mid \phi'$ in C such that $\tilde{p}, s \models_C r_s$. By Lemma 1, this gives $\tilde{p}, s \models^{il}_{S_{STT}} n_t, snat \vee \tilde{p}, s \models^{il}_{S_{DYN}} n_t$.

With all this we can apply rule Forward$_{il}$, giving $s \xrightarrow{p,p'}_{il} s'$.

$\boxed{\text{Case Est}_{hl}}$ If Est$_{hl}$ was used, we have hypotheses $(a_1)\ p \vdash_s src, dst$ and $(b_1)\ p[da \mapsto src], s \not\models_C$ DROP. From (a_1) it follows that rules DEst$_{hl}$ and SEst$_{hl}$ are applied, which means $\downarrow^{\text{DNAT}}_{il}\ \tilde{p}$ and $(s, p, \tilde{p})\ \downarrow^{\text{SNAT}}_{il}\ p'$, where $\tilde{p} = p[da \mapsto src]$ and $p' = \tilde{p}[sa \mapsto dst]$. The only two conditions not immediate from (a_1) are $p, s \not\models^{il}_{S_{D_1}}$ and $p', s \not\models^{il}_{S_{D_2}}$. To reach these conditions, we will consider the two directions that packet p may be taking: the original direction and the reply direction.

From (a_1), we know that p belongs to an established connection. The only way to establish this connection is through the New$_{hl}$ rule, which in turn implies one of the four following rules: ACCEPT$_{hl}$, SNAT$_{hl}$, DNAT$_{hl}$ or DSNAT$_{hl}$. Let's assume there was a packet p_i which lead to the execution of rule New$_{hl}$ and established the connection to which p belongs. Whichever of the four rules was applied to p_i, it holds true that the same rule should apply to p if it is going in the same direction as p_i, the original direction. In this case, proving the two remaining conditions follows the same logic used in the New$_{hl}$ section of the proof.

When p is in the reply direction of the connection, the previous logic does not apply. This is where C being *reply-aware* comes in. To prove this part, we will go over the four rules that could have lead to this established session: ACCEPT$_{hl}$, SNAT$_{hl}$, DNAT$_{hl}$ or DSNAT$_{hl}$.

$\boxed{\text{Case Est}_{hl} \bullet \text{ACCEPT}_{hl}}$ If the applied rule was ACCEPT$_{hl}$, then we can immediately say that $p = p' = p[da \mapsto src]$. From (b_1) and Lemma 1 we have $p', s \not\models^{il}_{S_{D_2}}$. To prove the other condition, $p, s \not\models^{il}_{S_{D_1}}$, we show that it is impossible to reach $p, s \models^{il}_{S_{D_1}}$. From (b_1) we have that no DROP rule affecting p exists in our configuration, which means the only rules in S_{D_1} that could match p must be resulting from the translation of a DNAT statement. However, the existence of such DNAT statement would go against the *reply-awareness* of C.

$\boxed{\text{Case Est}_{hl} \bullet \text{DNAT}_{hl}}$ If DNAT$_{hl}$ was applied when establishing the connection, the returning traffic is being source translated, so we have $p = p[da \mapsto src]$. Following the same logic as in the previous case, this gives that only a DNAT statement could insert a rule in S_{D_1} such that $p, s \models^{il}_{S_{D_1}}$. Again, this would go against the *reply-awareness* of C, giving us $p, s \not\models^{il}_{S_{D_1}}$. To prove $p', s \not\models^{il}_{S_{D_2}}$, we note that $p' = p[sa \mapsto dst]$. If there was a DROP rule such that $p', s \models_C$ DROP, C would not be *reply-aware* (according to the second clause of the definition). From Lemma 1 and $p', s \not\models_C$ DROP we have $p', s \not\models^{il}_{S_{D_2}}$.

$\boxed{\text{Case Est}_{hl} \bullet \text{SNAT}_{hl}}$ In the cases where the initially used rule was SNAT$_{hl}$, the returning traffic is going through a destination translation, giving $p[da \mapsto src] = p'$. This immediately gives $p', s \not\models^{il}_{S_{D_2}}$ from (b_1) and Lemma 1. Proving $p, s \not\models^{il}_{S_{D_1}}$ is similar to what was done previously. In this case, the third clause of the *reply-awareness* definition ensures that $p, s \not\models_C$ DROP. Combining this with Lemma 1, we return to the previous situation where the only way to achieve $p, s \models^{il}_{S_{D_1}}$ would be through the translation of a DNAT statement. This, however, also goes against the definition of *reply-awareness*. This gives $p, s \not\models^{il}_{S_{D_1}}$.

$\boxed{\text{Case Est}_{hl} \bullet \text{DSNAT}_{hl}}$ The proof for cases where the applied rule was DSNAT$_{hl}$ is simply a combination of the previous DNAT$_{hl}$ and SNAT$_{hl}$ cases. Since, in this case, we have both a DNAT and an SNAT rule, the fact that C is *reply-aware* become more constraining, allowing to follow the logic used in both previous cases. ■

B

MIGNIS configuration example and generated output

```
OPTIONS
router_address  localhost
router_port     5019

INTERFACES
lan      g1/0    10.0.0.0/24
ext      g2/0    0.0.0.0/0
neighbor g3/0    30.0.0.0/24

ALIASES
PC1         10.0.0.10
PC2         10.0.0.20
MAL1        50.0.0.10
MAL2        60.0.0.20
r_ext_ip    20.0.0.1

FIREWALL
#allow and masquerade traffic to the internet
lan [.] > ext

#setup a NAT for an HTTP server inside our LAN to be accessible from external networks
* > [r_ext_ip:80] PC1:80 tcp
PC1:80 [r_ext_ip:80] > * tcp

#allow full communication between these networks
lan <> neighbor

#Block malicious hosts from communicating with our local network
MAL1 / lan
MAL2 / lan
```

Listing B.1: Example of a MIGNIS configuration file.

```
zone security lan
!
zone security ext
!
zone security neighbor
!
ip access-list extended acl_intf_g1/0_lan_in
    permit ip 10.0.0.0 0.0.0.255 any
!
ip access-list extended acl_intf_g1/0_lan_out
    deny ip 60.0.0.20 0.0.0.0 10.0.0.0 0.0.0.255
    deny ip 50.0.0.10 0.0.0.0 10.0.0.0 0.0.0.255
    permit ip any any
!
ip access-list extended acl_intf_g2/0_ext_in
    deny tcp 0.0.0.0 255.255.255.255 10.0.0.10 0.0.0.0 eq 80
    deny ip 60.0.0.20 0.0.0.0 10.0.0.0 0.0.0.255
    deny ip 50.0.0.10 0.0.0.0 10.0.0.0 0.0.0.255
    permit ip 0.0.0.0 255.255.255.255 any
!
ip access-list extended acl_intf_g2/0_ext_out
    permit ip any any
!
ip access-list extended acl_intf_g3/0_neighbor_in
    deny tcp 30.0.0.0 0.0.0.255 10.0.0.10 0.0.0.0 eq 80
    permit ip 30.0.0.0 0.0.0.255 any
!
ip access-list extended acl_intf_g3/0_neighbor_out
    permit ip any any
!
ip access-list extended acl_ext-self
    deny ip 0.0.0.0 255.255.255.255 0.0.0.0 255.255.255.255
!
ip access-list extended acl_ext-lan
    deny ip 60.0.0.20 0.0.0.0 10.0.0.0 0.0.0.255
    deny ip 50.0.0.10 0.0.0.0 10.0.0.0 0.0.0.255
    permit tcp 0.0.0.0 255.255.255.255 10.0.0.10 0.0.0.0 eq 80
!
```

```
ip access-list extended acl_ext-neighbor
!
ip access-list extended acl_self-ext
    permit ip 0.0.0.0 255.255.255.255 0.0.0.0 255.255.255.255
!
ip access-list extended acl_self-lan
    permit ip 0.0.0.0 255.255.255.255 10.0.0.0 0.0.0.255
    permit tcp 0.0.0.0 255.255.255.255 10.0.0.10 0.0.0.0 eq 80
!
ip access-list extended acl_self-neighbor
    permit ip 0.0.0.0 255.255.255.255 30.0.0.0 0.0.0.255
!
ip access-list extended acl_lan-ext
    permit ip 10.0.0.0 0.0.0.255 0.0.0.0 255.255.255.255
    permit tcp 10.0.0.10 0.0.0.0 eq 80 0.0.0.0 255.255.255.255
!
ip access-list extended acl_lan-self
    permit tcp 10.0.0.10 0.0.0.0 eq 80 0.0.0.0 255.255.255.255
!
ip access-list extended acl_lan-neighbor
    permit ip 10.0.0.0 0.0.0.255 30.0.0.0 0.0.0.255
    permit tcp 10.0.0.10 0.0.0.0 eq 80 30.0.0.0 0.0.0.255
!
ip access-list extended acl_neighbor-ext
!
ip access-list extended acl_neighbor-self
!
ip access-list extended acl_neighbor-lan
    permit ip 30.0.0.0 0.0.0.255 10.0.0.0 0.0.0.255
    permit tcp 30.0.0.0 0.0.0.255 10.0.0.10 0.0.0.0 eq 80
!
ip access-list extended acl_nat_2
    deny ip 10.0.0.0 0.0.0.255 30.0.0.0 0.0.0.255
    permit ip 10.0.0.0 0.0.0.255 0.0.0.0 255.255.255.255
!
!
!
class-map type inspect match-any ext-self_cmap
```

```
    match access-group name acl_ext-self
!
[... Creating all the class-maps and matching the corresponding ACL ...]
!
class-map type inspect match-any neighbor-lan_cmap
    match access-group name acl_neighbor-lan
!
!
!
policy-map type inspect ext-self_pmap
    class type inspect ext-self_cmap
        inspect
    class class-default
        drop log
!
[... Creating all the policy-maps and matching the corresponding class-map ...]
!
policy-map type inspect neighbor-lan_pmap
    class type inspect neighbor-lan_cmap
        inspect
    class class-default
        drop log
!
!
!
zone-pair security ext-self source ext destination self
    service-policy type inspect ext-self_pmap
!
[... Creating all the zone-pairs and matching the corresponding policy-map ...]
!
zone-pair security neighbor-lan source neighbor destination lan
    service-policy type inspect neighbor-lan_pmap
!
!
!
interface g1/0
    ip access-group acl_intf_g1/0_lan_in in
    ip access-group acl_intf_g1/0_lan_out out
    zone-member security lan
    ip nat enable
!
interface g2/0
    ip access-group acl_intf_g2/0_ext_in in
    ip access-group acl_intf_g2/0_ext_out out
    zone-member security ext
    ip nat enable
!
interface g3/0
    ip access-group acl_intf_g3/0_neighbor_in in
    ip access-group acl_intf_g3/0_neighbor_out out
    zone-member security neighbor
    ip nat enable
!
ip nat source static tcp 10.0.0.10 80 20.0.0.1 80 extendable
!
ip nat source list acl_nat_2 interface g2/0 overload
```

Listing B.2: Abbreviated output generated from the configuration in Listing A.1.