A declarative based tool for reasoning about CISCO IOS firewall configurations

Shams Karim Valibhai

***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 Ado 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.**

***Index Terms*—Firewall, Cisco IOS, MIGNIS, Network Security, Firewall semantics, Network Address Translation (NAT)**

I. Introduction

The Internet can be defined as a network of networks, many of which are private and should not be accessible to everyone. One popular method to achieve this privacy is the use of 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. It is also possible to implement firewalls directly on Internet hosts, like personal computers and smartphones.

When placing a firewall at the border of a network, the most immediate location is the network gateway, since all traffic leaving the network passes there. A gateway is usually one of two devices, either a router or a server running an operating system like Linux. While the latter has its use cases, the former is the most commonly used.

The configuration of the firewall may not be a simple task. A network 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.

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 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. [2] and [3] are two examples of tools that do so with a graphical approach, while [4] follows a non-visual approach. In [5], 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. The MIGNIS semantic [1] is part of the second family. It offers an order-independent and simple to read syntax which applies to both connectivity and NAT functionalities. This semantic is currently used in the MIGNIS tool [6], offering a translation to *iptables* commands. An indispensable aspect of such a tool is that the translation is theoretically proven to comply with the used semantics. While this tool is compatible with most Linux distributions, it does not serve any purpose for the configuration of firewalls in routers.

A. 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. Our goal will be to translate these semantics into Cisco IOS commands, due to the widespread use of Cisco routers.

B. Related Work

The subject of firewall configuration has been researched through several different approaches. One possible approach is to create a firewall design model that, by design, attenuates or fixes the problems in conventional firewall such as The Structured Firewall Design [7]. Other examples include the Netfilter-like model in [8], the Model Definition Language [9], the High Level Firewall Language [10], Shorewall [11], FLIP [12] and a XML-based language [13] by Cuppens et al.

A different approach consists in using a graphical interface to help visualize and define the firewall policies. *firewalld* [14] is an open-source project that provides a simple way to configure firewall policies (in *iptables* on a Linux machine. It achieves this by combining high-level abstractions like services (which describe a set of ports and protocols) with a clean graphical interface. Another graphical tool is Firewall Builder [15]. This tool targets several different firewalls, from *iptables* to CISCO ASA, including CISCO IOS. The tool also contains features that try to check for redudancy in rules.

II. Firewall and NAT in CISCO IOS

We will start by going over the relevant CISCO IOS features and define an abstraction of a low-level router, that is a simplification of a CISCO IOS router. Considering that this abstraction 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.

A. IP access lists

Access lists (ACL) 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.

Listing 1. Extended access list example.
```
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
```

Each rule in an extended ACL has a well defined syntax, which can be seen in Listing 1. Rules are made of five mandatory parameters: (i) *Entry number:* a positive integer placed at the start of each rule that defines the ordering of rules inside an access list. (ii) *Action:* the action, *permit* or *deny*, to take if a packet matches the rule in question. (iii) *Protocol:* the protocol to match packets against, transport layer UDP, TCP, ..., or network layer IGMP, ICMP, OSPF, ... Keyword *ip* applies to any network protocol. (iv) *Source Address:* a network address or block (defined using a network address followed by a wildcard mask) that matches the source address of the packet being inspected. The keyword *host* can be placed before the address when referring to a single host; 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. (v) *Destination Address:* same as the *source address* parameter, but for the destination fields of the packet.

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.

B. 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.

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, 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 2 shows how the access-list in Listing 1 would be used to control traffic from *zone1* to *zone2*.

Listing 2. ZBFW configuration example..
```
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
```

C. Network Address Translation

IOS provides two mutually exclusive groups of commands to implement address translation: legacy NAT and 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, unlike legacy NAT.

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's first line shows an example of such rule. This translation can be triggered in two different ways. A source NAT from 192.168.1.50:80 to 85.10.10.10:8080 or a destination NAT from 85.10.10.10:8080 to 192.168.1.50:80.

Dynamic rules are source NAT only and mostly used for masquerading purposes. Listing 3 presents a masquerade NAT rule in its second line. This rule requires the creation of an ACL, which will match all traffic that should be translated. The *overload* keyword specifies that interface *g1/0*'s address should be used for the translation.

Listing 3. Static and dynamic NAT entries.
```
ip nat source static tcp 192.168.1.50 80 85.10.10.10 8080
ip nat source list acl_nat_1 interface g1/0 overload
```

D. 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.

E. 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 1 shows how the previous operations act upon packets.

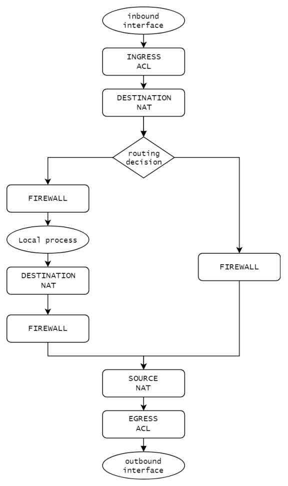

Fig. 1. Order of operations for the relevant features in a Cisco IOS router.

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.

F. 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 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.

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.

2) Access lists and address translation: As we saw in Section II-A, 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 1 and, from its second entry, generate the following basic rule: (192.168.1.0/24:$*$, $*$:$\{21, 23\}$, `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*

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

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. 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 and ϕ is a stateful operation over a packet.*

We can use, as an example, Listing 3 to build the static rule (192.18.1.50:80, 85.10.10.10:8080, `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) $sa(p) \in la \wedge \phi(p, s)$, denoted by $p, s \models_t t_i, snat$;
(ii) $da(p) \in ga \wedge \phi(p, s)$, 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) $\exists_{i \le n} \colon t_i = (la,\ ga,\ \phi) \wedge p, s \models_t t_i, snat \wedge \forall_{j<i}\ p, s \not\models_t t_j$, denoted by $p, s \models^S_T ga$;
(ii) $\exists_{i \le n} \colon t_i = (la,\ ga,\ \phi) \wedge p, s \models_t t_i, dnat \wedge \forall_{j<i}\ p, s \not\models_t t_j$, denoted by $p, s \models^D_T la$.

The case where any of the presented conditions is not satisfied is denoted by, respectively, $p, s \not\models^S_T$ or $p, s \not\models^D_T$.

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:*

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

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^i_d, 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^i_o, where $i, o \in \{1, ..., n\} \cup \{l\}$. These represent the rules between each zone in the firewall. A list F^x_y applies to all traffic entering in interface x and leaving through interface y. The self zone is represented by the letter l.

3) Semantics: In Table I 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 $(s, p) \quad \rfloor^{\delta}_{ll} \quad \tilde{p}$, 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. 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 a match was found in T. Rules in T are of the form $(la,\ ga,\ \phi)$ and relation $p, s \models^D_T t$ means a match was found between the global address, ga, of a rule and the destination address of the packet being

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

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

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

$$\dfrac{p \nvdash_s \quad \tilde{p}, s \models^{S}_{T} t \quad src \in t}{(s,p,\tilde{p}) \ \downharpoonleft^{\sigma}_{ll} \ \tilde{p}[sa \mapsto src]} \ [\text{SNATs}_{ll}]$$

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

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

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

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

TABLE I
SEMANTICS FOR THE LOW-LEVEL ROUTER.

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 $(s,p,\tilde{p}) \ \downharpoonleft^{\sigma}_{ll} \ p'$, 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 II-E. 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:

- $p, s \models_{I^{i}_{in}} accept$, it must be accepted by the inbound ACL placed at the ingress interface;
- $(s,p) \ \downharpoonleft^{\delta}_{ll} \ \tilde{p}$, the packet must go through the pre-routing NAT module. We denote the resulting packet as $\tilde{p}$, even if no translation occurred;
- $p \vdash_s \vee \ \tilde{p}, s \models_{F^{i}_{o}} accept$, it must either belong to an established connection or have its translated version, $\tilde{p}$, be explicitly accepted in F^{i}_{o}, the list of rules applied to traffic flowing from interface i to interface o;
- $(s,p,\tilde{p}) \ \downharpoonleft^{\sigma}_{ll} \ p'$, it must go through the post-routing NAT module. The resulting packet is denoted as $\tilde{p}$;
- $p', s \models_{I^{o}_{out}} accept$, 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 $(s,p) \ \downharpoonleft^{\delta}_{ll} \ p \colon [\text{DNew}_{ll}, \text{DEst}_{ll}]$, 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.

III. MIGNIS

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 existing problems in traditional firewall design. 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.

A. 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 the connection was started by a host in our network. 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.

The semantics for the MIGNIS language are presented in Table II. For simplicity, we use a similar notation as the one in Section II-F. 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 $p, s \models_C \text{RULE}(n_t)$. If a packet p in state s does not match a specific rule, we use the notation $p, s \not\models_C$ RULE.

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

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

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

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

$$\frac{p \not\vdash_s \qquad (s, p) \ \downharpoonleft^{hl} \ p'}{s \xrightarrow{p,p'}_{hl} s \uplus (p, p')} \ [\text{New}_{hl}]$$

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

TABLE II
MIGNIS LANGUAGE SEMANTICS.

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 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.

IV. From MIGNIS to CISCO IOS

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.

A. 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 II-F.*

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 $p, s \models^{il}_S t$. If no match occurs, we use $p, s \not\models^{il}_S$.

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 $p, s \models^{il}_S ga, snat$

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

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

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 $p, s \models^{il}_S t$. If no match occurs, we use $p, s \not\models^{il}_S$.

$$\frac{p \vdash_s src, dst}{(s, p)\ \downharpoonleft^{\text{DNAT}}_{il}\ p[da \mapsto src]}\ [\text{DEst}_{il}] \qquad \frac{p \nvdash_s \quad p, s \not\models^{il}_{S_{STT}}\ dnat}{(s, p)\ \downharpoonleft^{\text{DNAT}}_{il}\ p}\ [\text{DNew}_{il}]$$

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

$$\frac{p \vdash_s src, dst}{(s, p, \tilde{p})\ \downharpoonleft^{\text{SNAT}}_{il}\ \tilde{p}[sa \mapsto dst]}\ [\text{SEst}_{il}]$$

$$\frac{p \nvdash_s \quad p, s \not\models^{il}_{S_{STT}}\ snat \qquad p, s \not\models^{il}_{S_{DYN}}}{(s, p, \tilde{p})\ \downharpoonleft^{\text{SNAT}}_{il}\ \tilde{p}}\ [\text{SNew}_{il}]$$

$$\frac{p \nvdash_s \quad p, s \models^{il}_{S_{STT}} t, snat \quad src \in t}{(s, p, \tilde{p})\ \downharpoonleft^{\text{SNAT}}_{il}\ \tilde{p}[sa \mapsto src]}\ [\text{SNATs}_{il}]$$

$$\frac{p \nvdash_s \quad p, s \not\models^{il}_{S_{STT}}\ snat \quad p, s \models^{il}_{S_{DYN}}\ t \quad src \in t}{(s, p, \tilde{p})\ \downharpoonleft^{\text{SNAT}}_{il}\ \tilde{p}[sa \mapsto src]}\ [\text{SNATd}_{il}]$$

$$\frac{\begin{array}{c} sa(p) \notin \mathcal{L} \quad da(\tilde{p}) \notin \mathcal{L} \\ p, s \not\models^{il}_{S_{D_1}} \quad (s, p)\ \downharpoonleft^{\text{DNAT}}_{il}\ \tilde{p} \quad p \vdash_s \vee\ \tilde{p}, s \not\models^{il}_{S_D} \\ p \vdash_s \vee\ \tilde{p}, s \models^{il}_{S_A} \quad (s, p, \tilde{p})\ \downharpoonleft^{\text{SNAT}}_{il}\ p' \quad p', s \not\models^{il}_{S_{D_2}} \end{array}}{s \xrightarrow{p, p'}_{il} s \uplus (p, p')}\ [\text{Forward}_{il}]$$

$$\frac{\begin{array}{c} sa(p) \notin \mathcal{L} \quad da(p) \in \mathcal{L} \\ p, s \not\models^{il}_{S_{D_1}} \quad (s, p)\ \downharpoonleft^{\text{DNAT}}_{il}\ p \colon [\text{DNew}_{il}, \text{DEst}_{il}] \\ p \vdash_s \vee\ p, s \not\models^{il}_{S_D} \qquad p \vdash_s \vee\ p, s \models^{il}_{S_A} \\ (s, p, \tilde{p})\ \downharpoonleft^{\text{SNAT}}_{il}\ p \colon [\text{SNew}_{il}, \text{SEst}_{il}] \end{array}}{s \xrightarrow{p, p}_{il} s \uplus (p, p)}\ [\text{Input}_{il}]$$

$$\frac{\begin{array}{c} sa(p) \in \mathcal{L} \quad da(p) \notin \mathcal{L} \\ (s, p)\ \downharpoonleft^{\text{DNAT}}_{il}\ p \colon [\text{DNew}_{il}, \text{DEst}_{il}] \\ p \vdash_s \vee\ p, s \not\models^{il}_{S_D} \quad p \vdash_s \vee\ p, s \models^{il}_{S_A} \\ (s, p, \tilde{p})\ \downharpoonleft^{\text{SNAT}}_{il}\ p \colon [\text{SNew}_{il}, \text{SEst}_{il}] \qquad p, s \not\models^{il}_{S_{D_2}} \end{array}}{s \xrightarrow{p, p}_{il} s \uplus (p, p)}\ [\text{Output}_{il}]$$

TABLE III
INTERMEDIATE FIREWALL SEMANTICS.

In Table III 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.

B. Translating from MIGNIS to the intermediate firewall

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

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

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

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

TABLE IV
TRANSLATION FROM A MIGNIS CONFIGURATION TO AN INTERMEDIATE FIREWALL CONFIGURATION.

The translations in Table IV, 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 III, 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). 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.

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 IV. 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.

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.

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.

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 | ϕ') 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 IV 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 $\mathcal{F_I}$ 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 $src \in n_t$.

In Theorem 2, we formally prove the soundness and completeness of the translation in Table IV, 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.

Theorem 2. *Let $\mathcal{F_I}$ be the intermediate firewall obtained from applying the translation in Table IV 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 $s \xrightarrow{p,p'}_{il} s'$.

C. From the intermediate to low-level firewall

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

Definition 21. *Let $\mathcal{F_I}$ 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 $\mathcal{F_I}$ 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 $p, s \models_{\overline{F^i_o}} t$.

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 $\mathcal{F_I}$ 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.*

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 the low-level router generated from the translation of intermediate firewall $\mathcal{F}_\mathcal{I}$. We have, for any state s and packets $p, \tilde{p}, p'$, that:*

(i) $(s,p) \downarrow_{ll}^{\delta} \tilde{p} \Rightarrow / \Leftarrow^{*} (s,p) \downarrow_{il}^{\mathrm{DNAT}} \tilde{p}$;
(ii) $(s,p,\tilde{p}) \downarrow_{ll}^{\sigma} p' \Rightarrow / \Leftarrow^{*} (s,p,\tilde{p}) \downarrow_{il}^{\mathrm{SNAT}} p'$;
Where $\Leftarrow^{*}$ only holds if $\mathcal{F}_\mathcal{I}$ is a deterministic intermediate firewall.

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* $s \xrightarrow{p,p'}_{ll} s'$.

D. 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.

After parsing the configuration file with the firewall rules, 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 III-A. 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* rule into a real IOS command can be found in the Appendix A.

V. Conclusion

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.

References

[1] P. Adão, C. Bozzato, G. Dei Rossi, R. Focardi, and F. L. Luccio, "Mignis: A semantic based tool for firewall configuration," *In CSF14*, pp. 351–365, 2014.
[2] F. Mansmann and W. Cheswick, "Visual analysis of complex firewall configurations," *In VizSec12*, pp. 1–8, 2012.
[3] R. Boutaba, "PolicyVis: Firewall Security Policy Visualization and Inspection," *System*, pp. 1–16, 2008.
[4] 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," *In POLICY07*, pp. 151–160, 2007.
[5] P. Eronen and J. Zitting, "An expert system for analyzing firewall rules," *In NordSec01)*, p. 100107, 2001.
[6] secgroup, "Mignis." [Online]. Available: https://github.com/secgroup/Mignis
[7] M. G. Gouda and A. X. Liu, "Structured firewall design," *Computer Networks*, vol. 51, no. 4, pp. 1106–1120, 2007.
[8] A. Jeffrey and T. Samak, "Model checking firewall policy configurations," *In POLICY 2009*, no. 1, pp. 60–67, 2009.
[9] Y. Bartal, A. Mayer, K. Nissim, and A. Wool, "Firmato," *ACM Transactions on Computer Systems*, vol. 22, no. 4, pp. 381–420, 2004.
[10] "High Level Firewall Language." [Online]. Available: https://www.cusae.com/hlfl
[11] "Shorewall." [Online]. Available: http://www.shorewall.net/
[12] 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," *In SACMAT07*, p. 185, 2007.
[13] 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.
[14] firewalld.org project, "firewalld." [Online]. Available: https://firewalld.org/
[15] NetCitadel, "Firewall Builder." [Online]. Available: http://fwbuilder.sourceforge.net/

APPENDIX

- A **basic rule** (n_1, n_2, ϕ, t) is translated into: $t\ \ \phi^*\ \ n_1^*\ \ n_2^*$ This command is placed inside the context:

  ```
  ip access-list extended acl_[name]
  ```

  where `[name]` depends on whether the basic rule is in I_d^i or F_o^i and specifies the target interfaces. n_1^* is the IOS representation of n_1, which follows the formats shown in Listing 1, likewise for n_2^*. If ϕ is empty, then ϕ^* assumes the value `ip`, otherwise $\phi^* = \phi$.
- A **static translation rule** (la, ga, ϕ), in T, is translated into:

  ip nat source static $\phi\ la\ ga$

  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 also runs commands that apply the translated rules to the respective interfaces or pair of zones.