Localizing Firewall Security Policies

Pedro Adão[1], Riccardo Focardi[2],
Flaminia L. Luccio[3], and Joshua D. Guttman[4]

[1] SQIG, Instituto de Telecomunicações, and Instituto Superior Técnico, Universidade de Lisboa
[2] University Ca' Foscari, Venice and Cryptosense
[3] University Ca' Foscari, Venice
[4] The MITRE Corporation and Worcester Polytechnic Institute

Abstract. In complex networks, filters may be applied at different nodes to control how packets flow. In this paper, we study how to locate filtering functionality within a network. We show how to enforce a set of security goals while allowing maximal service subject to the security constraints. To implement our results we present a tool that given a network specification and a set of control rules automatically localizes the filters and generates configurations for all the firewalls in the network. These configurations are implemented using an extension of Mignis — an open source tool to generate firewalls from declarative, semantically explicit configurations.
Our contributions include a way to specify security goals for how packets traverse the network; an algorithm to distribute filtering functionality to different nodes in the network to enforce a given set of security goals; and a proof that the results are compatible with a Mignis-based semantics for network behavior.

1 Introduction

Organizations have big and complicated networks. A university may have a network partitioned into dozens of subnets, separated either physically or as VLANs. Although many of those subnets are very similar, for instance in requiring similar protection, others are quite different, for instance those that contain the university's human resources servers. The latter require far tighter protection. As another example, consider a corporation: some subnets contain public-facing machines such as web servers or email servers; others support an engineering department or a sales department; and yet others contain the process-control systems that keep a factory operating. Thus, they should be governed by entirely different policies constraining what network flows can reach them, and from where.

Indeed, a network is a graph, in which the packets flow over the edges, and the nodes may represent routers, end systems, and so forth. The security goals we would like to enforce reflect this graph structure. They are essentially about trajectories, i.e. about where packets travel to get where they are going. For instance, a packet that reaches the process control system in the factory should

not have originated in the public internet. After all, some adversary may use it to insert a destructive command, regardless of how benign its source address header field looks when it arrives. Similarly, a packet that originates in the human resources department should not traverse the public internet en route to the sales department. It could be inspected while there, compromising information about salaries within the company. A security goal may also restrict which packets may take a particular trajectory, for instance only packet addressed to port 80 or 443 on a web server.

In this paper, we introduce an expressive way to state security goals for packet flows. A security goal involves three network locations B, R, E, two assumption properties ψ_B, ψ_E, and one requirement property ϕ. Each of the properties characterizes packets—generally, their header fields—as found at a relevant location. The state of a packet may change as it traverses the network, so the properties apply to the packet as it appears when present at the corresponding locations. Network address translation (NAT) is the packet-changing operation that we consider in this paper. However, the IP security protocols raise similar issues. The semantics of a goal concerns the packet flows that begin at B; end at E; and traverse R. The goal requires that any packet that follows such a path must satisfy predicate ϕ when it is present at location R, assuming that it satisfied ψ_B at the beginning and ψ_E at the end.

We provide an algorithm to determine what filtering to do at the different routers in the network to enforce a given set of security goals. Our algorithm allows maximal functionality, delivering as rich a set of packets as are compatible with the policy, in the relevant case of packets that are not spoofed at their origin and not delivered promiscuously at the end. We also provide a rigorous semantics of network behavior that allows us to prove that our algorithm is sound. Finally, to implement our results we present a tool that, given a network specification and a set of control rules, uses this algorithm to automatically localize the filters and to generate configurations for all the firewalls in the network. These configurations are generated using `Mignis`$^+$, an extension of Mignis — an open-source tool to generate firewalls from declarative, semantically explicit configurations.

Related work Our approach is motivated by some previous work. We are interested in the defining behavioral security specifications in a network. In [6] Guttman and Herzog studied trajectory-based security goals, developing techniques to determine whether existing configurations enforce them correctly. Their goals were a less expressive class of two-region security goals, and they were unable to generate configurations in addition to analyzing given ones. However, this paper follows the lead of [6] in regarding security goals as properties of the trajectories of packets as they traverse the network, formalized as a graph. Our network graphs and their executions are formalized in the frame model of Guttman and Rowe [7].

Zhang et al. [11] focus more on the possible conflicts among policies at different organizational levels, and less on their consequences given the topology of

the network. Our method is more general and is also designed to apply in the case of devices that transform packets as they pass; we have particularly focused on NAT.

Kurshid et al. [8] demonstrate that it is possible, within software defined networking, to check dynamically if global, behavioral properties are maintained as invariants, for instance reachability for certain sorts of packets. We instead make no claims of real-time, on-line feasibility, but we offer a way to solve a broader range of security problems at design time.

Interesting work has also been done from a network programming language point of view. The Frenetic project [5] deals with dynamic policies, but does not address network reachability or cyclicity problems; it is not clear how newly added constructs can interact with old ones. NetKAT [3] is equipped with a sound and complete equational theory. However, while it might be possible to implement our security goals, localization algorithm, and `Mignis`$^+$ processing within the NetKAT framework, doing so would still require the analysis we present here.

Finally, much work has been devoted to firewall analysis, e.g. Margrave [9], which again lacks the distributed behavior of the network. Some automatic tools for testing of the firewall configuration enforcement have been proposed (see, e.g., [2, 10]). These tools are very powerful in static networks, but they do neither prevent consistency problems when new rules are added in the wrong order, nor avoid completeness problems for some undefined packet rules.

Structure of the paper In Section 2 we present a model of our network as frames and executions, together with the different types of nodes (i.e., routers, network regions, and end hosts), and trajectories. In Section 3 we define the security goals and the functionality goals. In Section 4 we describe how to assign filtering functionality to routers so as to enforce a set of region control statements. We first consider the case without NAT. In Section 5 we first present `Mignis`$^+$, an extension of Mignis, and we then present a tool that given a network specification and a set of region control rules automatically localizes the filters and generates `Mignis`$^+$ configurations for all the firewalls in the network, again assuming the NAT-free case. We also describe a case study with automatically generated configurations. In Section 6, we extend localization to cover NATs. We conclude in Section 7.

2 Network model

We model networks by *frames* and *executions* [7]. A *location* $\ell \in \mathcal{LO}$ represents a network node and is equipped with a set of traces $\mathsf{traces}(\ell)$ defining its possible local behaviors. We will use the words *location* and *node* interchangeably. A *channel* $c \in \mathcal{CH}$ allows the synchronous one-way transmission of messages between the locations at its endpoints, $\mathsf{sender}(c)$ and $\mathsf{rcpt}(c)$. Each message also carries some *data* $v \in \mathcal{D}$. An *event* e is an occurrence of a message on a channel, so that $\mathsf{msg}(e)$ is the data transmitted and $\mathsf{chan}(e)$ is the channel on which

it occurs. Events with the same channel and event are indistinguishable, apart from the time at which they occur.

Definition 1 (Frame [7]). *Let $\mathcal{LO}, \mathcal{CH}, \mathcal{D}, \mathcal{E}$ be domains that we will call* locations, channels, data, *and* events, *resp. Assume that $\mathcal{LO}, \mathcal{CH}$ are finite.*

1. *A* trace set *T for a set of channels $C \subseteq \mathcal{CH}$ is a set of sequences of events $\langle e_0, e_1, \ldots, e_k \rangle$ such that:*
   (a) If e_i is an event in $t \in T$, then $\mathsf{chan}(e_i) \in C$; and
   (b) T is prefix-closed.
2. *A* frame *is a structure containing the domains and functions shown in Table 1 such that, for all $\ell \in \mathcal{LO}$, if $\mathsf{traces}(\ell) = T$, then T is a trace set for the*

Domains: $\mathcal{LO}, \mathcal{CH}, \mathcal{D}, \mathcal{E}$

$\mathsf{sender} : \mathcal{CH} \to \mathcal{LO}$	$\mathsf{rcpt} : \mathcal{CH} \to \mathcal{LO}$	$\mathsf{traces} : \mathcal{LO} \to \mathcal{P}(\mathcal{E}^*)$
$\mathsf{chan} : \mathcal{E} \to \mathcal{CH}$	$\mathsf{msg} : \mathcal{E} \to \mathcal{D}$	

Table 1. Signature of frames

set of channels C connected to ℓ, namely $C =$

$$\{c \in \mathcal{CH} : \mathsf{sender}(c) = \ell \text{ or } \mathsf{rcpt}(c) = \ell\}.$$

A trace of ℓ *means a sequence $t \in \mathsf{traces}(\ell)$.* ///

Each $\mathcal{F}$ determines a directed graph.

Definition 2 (Frame graphs [7]). *If $\mathcal{F}$ is a frame, then the* graph of $\mathcal{F}$, *written $\mathsf{gr}(\mathcal{F})$, is the directed graph (V, E) whose vertices V are the locations $\mathcal{LO}$, and such that there is an edge $(\ell_1, \ell_2) \in E$ iff, for some $c \in \mathcal{CH}$, $\mathsf{sender}(c) = \ell_1$ and $\mathsf{rcpt}(c) = \ell_2$.* ///

The execution model for frames relies on partially ordered sets of events. The ordering represents the *occurs before* relation, and events at a certain location ℓ are required to be totally ordered and included in $\mathsf{traces}(\ell)$. Formally:

Definition 3 (Executions [7]). *Let $\mathcal{F}$ be a frame, and $\mathcal{B} = (B, \preceq)$ be a finite, partially ordered set of events, i.e. $B \subseteq \mathcal{E}$ and $\preceq$ is a partial order on B. Define* $\mathsf{proj}(B, \ell) =$

$$\{e \in B : \mathsf{sender}(\mathsf{chan}(e)) = \ell \text{ or } \mathsf{rcpt}(\mathsf{chan}(e)) = \ell\}.$$

$\mathcal{B}$ is an execution of $\mathcal{F}$, *written $\mathcal{B} \in \mathsf{Exc}(\mathcal{F})$, iff*

1. *$\mathsf{proj}(B, \ell)$ is linearly ordered by $\preceq$, and*
2. *$\mathsf{proj}(B, \ell) \in \mathsf{traces}(\ell)$.* ///

A finite linearly ordered set is a sequence. Thus, Clause 1 and the finiteness condition ensure that $\mathsf{proj}(B, \ell)$ is a sequence. Clause 2 adds the requirement that this sequence is a trace of ℓ, for each choice of ℓ.

We often write (c, p) for any event e such that $\mathsf{chan}(e) = c$ and $\mathsf{msg}(e) = p$. Since different events may occur at different times, but involve the same data value p on the same channel c, this is strictly speaking an abuse of notation.

In this paper, the data values $\mathcal{D}$ are packets. We will refer to the source and destination addresses in a packet p as $\mathtt{sa}(p)$ and $\mathtt{da}(p)$. Packets have other familiar properties such as a payload, a protocol and—if the protocol is `tcp` or `udp`—a source port and a destination port.

In networking, physical connections are nearly always bidirectional. A bidirectional connection will be represented by a pair of unidirectional channels. When convenient, we will refer to this pair as if it were a single bidirectional edge.

2.1 Node behaviour

Nodes in a network can be of three types: *(i)* routers, *(ii)* network regions, and *(iii)* end hosts. Routers are connected to network regions (one or more), whereas end hosts are connected to a single network region, or possibly more than one network region when the device has multiple interfaces.

End hosts are either *promiscuous* or *chaste*. A promiscuous host will accept any packet sent to it, and may transmit any packet. A chaste host h has a set of IP addresses $IP(h)$, accepting only packets with destination address $i \in IP(h)$, and transmitting only packets with source address $i \in IP(h)$.

Given a network, one can easily construct a frame: the locations of the frame are the network nodes; for each (undirected) edge of the network, there is a pair of arcs, one in each direction. We provide $\mathsf{traces}(\ell)$ for each location:

Network region: A network region can only forward previously received packets. Formally: $(c_1, p_1), \ldots, (c_k, p_k) \in \mathsf{traces}(\ell)$ iff, $\forall i \in [1, k]$, if $\mathsf{sender}(c_i) = \ell$, then there exists $j < i : \mathsf{rcpt}(c_j) = \ell, p_i = p_j$. Notice that a network region does not guarantee to deliver every packet, nor to deliver it at most once.

End host: We have no constraint on traces for promiscuous hosts. For chaste ones, we require that the source or destination address matches host's addresses for any packet it sends or receives: $(c_1, p_1), \ldots, (c_k, p_k) \in \mathsf{traces}(h)$ iff $\forall i \in [1, k] : \mathsf{sender}(c_i) = \ell$ implies $\mathtt{sa}(p_i) \in h$ and $\mathsf{rcpt}(c_i) = \ell$ implies $\mathtt{da}(p_i) \in h$.

Router: When r is a router, then its behavior is determined by a firewall $\mathcal{FW}_r : \mathcal{T} \times \mathcal{CH} \times \mathcal{CH} \times P \to \mathcal{P}(P)$. Intuitively, $\mathcal{FW}_r$ takes a trace $t \in \mathcal{T}$, representing the history, an input and an output channel c and c' and a packet p and returns a (possibly empty) set of translations of p. Given history t, these translations p' represent all the possible ways in which p is accepted by the firewall when p is received from c and p' is delivered to c'. Given a firewall $\mathcal{FW}_r$ and a routing function ρ, the behaviour of the router is determined as follows:

$$\frac{p' \in \mathcal{FW}(t, c, c', p) \qquad \rho(p') = c'}{\langle t, \sigma\rangle \rightsquigarrow \langle t.(c,p), \sigma \cup (c', p')\rangle} \ [\text{filter}_{\text{in}}]$$

$$\langle t, \sigma \uplus \{(c,p)\}\rangle \rightsquigarrow \langle t.(c,p), \sigma\rangle \ [\text{route}_{\text{out}}]$$

Router configuration is a pair $\langle t, \sigma\rangle$ where trace t represents the history and σ is a buffer containing pairs (c, p) representing a packet p to be delivered over channel c. Intuitively, rule $\text{filter}_{\text{in}}$ accepts packet p from channel c, adding it to t, only if there exists $p' \in \mathcal{FW}(t, c, c', p)$ that will be delivered over channel c'. If this is the case, (c', p') is buffered in σ. Rule $\text{route}_{\text{out}}$ takes a pair (c, p) from the buffer and delivers packet p over channel c.
We let $t \in \mathsf{traces}(r)$ iff $\langle \emptyset, \emptyset\rangle \rightsquigarrow \ldots \rightsquigarrow \langle t, \sigma\rangle$.

Definition 4. *We say that a frame $\mathcal{F}$* tightens *a frame $\mathcal{F}'$ iff they share the same graph, and for all routers r, and all t, c_1, c_2, p, $\mathcal{FW}_r(t, c_1, c_2, p) \subseteq \mathcal{FW}'_r(t, c_1, c_2, p)$.*

2.2 Trajectories

A *trajectory* is a path that a packet may take as it traverses the network. Since the packet itself may change as it passes through a router with Network Address Translation, we need to define which events may belong together, i.e. when the packet in a second event is the result of a NAT operation on the packet in the first event.

Definition 5. *Two events e, e' are* associated *if $\mathsf{rcpt}(\mathsf{chan}(e)) = \mathsf{sender}(\mathsf{chan}(e')) = \ell$ and either*

1. *ℓ is a network region and $\mathsf{msg}(e) = \mathsf{msg}(e')$; or*
2. *ℓ is a router that allows a trace $(c_0, p_0), \ldots, (c_k, p_k)$ and $\exists i, j : i < j$ such that $(c_i, p_i) = e \wedge (c_j, p_j) = e'$, $p_j \in \mathcal{FW}_r(t_{i-1}, c_i, c_j, p_i)$ and $\rho(p_j) = c_j$ where $t_{i-1} = (c_0, p_0), \ldots, (c_{i-1}, p_{i-1})$.* ///

The next result shows that packet association corresponds to the causal relation between inputs and outputs in the network.

Proposition 6. *Let $\ell \in \mathcal{LO}$ be a network region or router.*

1. *If $e = (c, p), e' = (c', p')$ are associated at ℓ then there exists a trace $(c_1, p_1), \ldots, (c_k, p_k) \in \mathsf{traces}(\ell)$ such that $(c_i, p_i) = (c, p) \wedge (c_j, p_j) = (c', p') \wedge i < j$;*
2. *$\forall (c_1, p_1), \ldots, (c_k, p_k) \in \mathsf{traces}(\ell)$, if $\mathsf{sender}(c_j) = \ell$ then $\exists i < j : \mathsf{rcpt}(c_i) = \ell$ and $(c_i, p_i), (c_j, p_j)$ are associated at ℓ.*

Proof. For item (1): Assume ℓ is a network region. Then $\mathsf{msg}(e) = p = p' = \mathsf{msg}(e')$. By the behaviour of network regions (cf. section 2.1) we get that $(c, p), (c', p') \in \mathsf{traces}(\ell)$ which gives the thesis. If ℓ is a router, thesis is a direct consequence of definition 5, item 2.

Item (2) can be trivially derived by the definition of traces for network region and routers given in section 2.1.

A *trajectory* is a path that a packet may take from the point it originates through network regions and routers, possibly reaching an end host at which it is received. A trajectory is *successful* if it originates with an end host whose IP addresses include its (claimed) source address, and terminates with an end host whose IP addresses include its destination address. Thus, at the beginning it is not spoofed, and at the end it is not promiscuously received.

Definition 7. *Let* $\mathcal{B} \in \mathsf{Exc}(\mathcal{F})$ *be an execution and let* $\vec{e} = \langle e_0, \ldots e_k \rangle$ *be a sequence of events in* $\mathcal{B}$*. We say that* $\vec{e}$ *is a* trajectory in $\mathcal{B}$ *iff*

i. $0 \leq i < j \leq k$ *implies* $e_i \prec e_j$*;*
ii. either $\mathsf{sender}(\mathsf{chan}(e_0))$ *is promiscuous, or* $\mathtt{sa}(\mathsf{msg}(e_0)) \in \mathsf{IP}(\mathsf{sender}(\mathsf{chan}(e_0)))$*;*
iii. either $\mathsf{rcpt}(\mathsf{chan}(e_k))$ *is promiscuous, or* $\mathtt{da}(\mathsf{msg}(e_k)) \in \mathsf{IP}(\mathsf{rcpt}(\mathsf{chan}(e_k)))$*;*
iv. if $i < k$*, then* $\mathsf{rcpt}(\mathsf{chan}(e_i))$ *is a network region or router, not an end host;*
v. for all i *such that* $0 \leq i < k$*,* e_i, e_{i+1} *are associated.*

The trajectory $\vec{e}$ *is* successful *iff*

$$\mathtt{sa}(\mathsf{msg}(e_0)) \in \mathsf{IP}(\mathsf{sender}(\mathsf{chan}(e_0)))$$
$$\mathtt{da}(\mathsf{msg}(e_k)) \in \mathsf{IP}(\mathsf{rcpt}(\mathsf{chan}(e_k))) \qquad ///$$

The purpose of course of a network is to allow trajectories to succeed whenever that is compatible with the security goals of the network administrator.

When a packet p will be unchanged throughout a trajectory, for instance because the frame involves no network address translation, we often regard a trajectory as a pair consisting of the packet p together with a path π, where a *path* is a sequence of adjacent locations.

3 Goals for Security and Functionality

Each frame determines a set of trajectories, namely all of those compatible with the router configurations and channels of the frame. In this section we explain how to say which of those trajectories are good and which are bad, i.e. how to express security goals to govern a frame. In Section 4 we will provide an algorithm to determine router configurations that will enforce a given set of these goals.

3.1 Security Goals

We focus on *three-region policy statements* as security goals. We refer to them as *region control statements*. These take the following form, in which the region variables B, E, R each refer to end hosts and network regions, and ψ_B, ψ_E, and ϕ refer to sets of packets, determined by the header fields of the packet at that step in the trajectory:

Region control $\psi_B @ B \to \phi @ R \to \psi_E @ E$**:** For every trajectory τ,
if τ starts at location B with a packet that satisfies ψ_B,
and τ ends at location E with a packet that satisfies ψ_E,

then if location R is traversed in τ, the packet satisfies ϕ while at R.

In these region control statements, the sets ψ_B, ψ_E restrict the applicability of the security goal: They constrain a trajectory only if they are satisfied at the beginning and end respectively. By contrast, ϕ is imposing a requirement, since the network must ensure it is satisfied when the trajectory reaches R. Thus ψ_B, ψ_E are *preconditions* and ϕ is a *postcondition*, even though E is reached last.

We can express many useful properties by suitable choices of ϕ. For instance, we may want to ensure that a packet passing from B to E undergoes network address translation properly, so that its source address at the time it traverses R is a *routable address* rather than a private address. We may want to assure that packets from public regions B to a protected corporate region E have been *properly filtered* by the time they reach the corporate entry network R; thus, they should be `tcp` packets whose destinations are the publicly accessible web and email servers, and whose destination ports are the corresponding well-known ports. These provide examples of region control statements.

We will always assume that $B \neq E$, but there are many useful cases in which the intermediate region R equals one of the endpoints, i.e. $B = R$ or $R = E$. We refer to these as *two-region* statements, since they just restrict the packets that can travel from B to E. When $R = E$, the statement says that whenever a packet travels from B to R, it must satisfy ϕ. Generally speaking, when the purpose of the statement is to protect R from potentially harmful packets from B, this form of the statement is useful; the property ϕ specifies which packets are safe. The two-region formulas may also be used with $R = B$ to protect B against disclosure of certain packets to E. In this case, the property ϕ specifies which packets are non-sensitive.

Consider another type of security goal involving three regions:

Traversal control $\psi_B@B \twoheadrightarrow R \twoheadrightarrow \psi_E@E$**:** For every trajectory τ,
if τ starts at location B with a packet that satisfies ψ_B,
and τ ends at location E with a packet that satisfies ψ_E,
then location R is traversed in τ.

As an example of a traversal control statement, consider a corporate network that has packet inspection in a particular region R. Then we may want to ensure that packets from public sources B to internal destinations E traverse R. The reverse is also important in most cases, i.e. that packets from internal sources to public destinations should traverse R.

Given a particular network, i.e. the graph underlying a frame, one strategy to enforce a traversal control statement is using region control statements. We may select a suitable cut set C of nodes between B and E where $R \in C$. We can then implement the traversal control statement by stipulating the region control statements that for trajectories from B to E, if the packets traverse any member of $C \setminus \{R\}$, then they satisfy the always-false header property **false**. That is, we have the family of statements, one for each $R' \neq R$ in C:

$$\psi_B@B \twoheadrightarrow \mathbf{false}@R' \twoheadrightarrow \psi_E@E.$$

Given this, we will focus our attention on region control statements $\psi_B@B \rightarrow \phi@R \rightarrow \psi_E@E$.

A trajectory violates a region control statement if it has the correct beginning and end points, but violates the property ϕ while at R.

Definition 8. *A trajectory $\vec{e} = \langle e_0, \ldots e_k \rangle$ is a counterexample to the region control statement $\psi_B@B \rightarrow \phi@R \rightarrow \psi_E@E$ iff*

1. $\mathsf{sender}(\mathsf{chan}(e_0)) = B$ *and* $\psi_B(\mathsf{msg}(e_0))$*;*
2. $\mathsf{rcpt}(\mathsf{chan}(e_k)) = E$ *and* $\psi_E(\mathsf{msg}(e_k))$*; and*
3. *for some i such that $0 \leq i \leq k$,* $\neg\phi(\mathsf{msg}(e_i))$ *and either* $\mathsf{sender}(\mathsf{chan}(e_i)) = R$ *or* $\mathsf{rcpt}(\mathsf{chan}(e_i)) = R$*.*

When $\vec{e}$ is not a counterexample we say that $\vec{e}$ satisfies *the region control statement $\psi_B@B \rightarrow \phi@R \rightarrow \psi_E@E$. A frame $\mathcal{F}$* enforces *$\psi_B@B \rightarrow \phi@R \rightarrow \psi_E@E$ iff, whenever $\mathcal{B} \in \mathsf{Exc}(\mathcal{F})$ is an execution of $\mathcal{F}$ and $\vec{e}$ is a trajectory in $\mathcal{B}$, then $\vec{e}$ satisfies $\psi_B@B \rightarrow \phi@R \rightarrow \psi_E@E$.* ///

3.2 Functionality Goals

Unlike security goals, which are mandatory, functionality may be a matter of degree. We choose to measure functionality by the set of packets that have a *successful trajectory* (Def. 7). A successful trajectory is one in which a packet travels from a non-spoofing producer to a consumer actually located at the destination address of the packet. We focus on successful trajectories because we regard spoofing originators as intrinsically hostile, which is also the case for promiscuous hosts that consume packets not addressed to them. We do not care whether exactly the same paths are available for successful trajectories, but only that a successful trajectory should exist for as many packets as possible. Thus, we define:

Definition 9. *Let $\mathcal{F}_1$ and $\mathcal{F}_2$ be two frames with the same underlying graph.*

$\mathcal{F}_2$ is at least as successful functionally *as $\mathcal{F}_1$ iff, for all locations ℓ_0, ℓ_1, and packets p_0, p_1, if there is a successful trajectory in which p_0 starts at ℓ_0 and p_1 ends at ℓ_1 in $\mathcal{F}_1$, then there is also a successful trajectory for p_0 starting at ℓ_0 in which p_1 ends at ℓ_1 in $\mathcal{F}_2$.* ///

Given an underlying network topology, formalized as a graph, and a set of security goals, the acceptable frames are those that allow no counterexamples to the security goals. Among those, one would like to construct a frame that is maximal in the ordering of successful functionality.

3.3 Forms of Goals

In the remainder of this paper, we will make an assumption about the sets of security goals we will consider. It is mere bookkeeping, since any set of goals can be rewritten as a somewhat larger set of goals satisfying this assumption.

Assumption 10. In any set of security goals to be enforced, for any of those goals $\psi_B@B \to \phi@R \to \psi_E@E$, one of the following three cases holds:

1. for all packets $p \in \psi_B$, $\mathtt{sa}(p) \in IP(B)$, and
   for all packets $p \in \psi_E$, $\mathtt{da}(p) \in IP(E)$;
2. for all packets $p \in \psi_B$, $\mathtt{sa}(p) \notin IP(B)$; or
3. for all packets $p \in \psi_E$, $\mathtt{da}(p) \notin IP(E)$.

We refer to these goals as (1) *success goals*, (2) *spoofing goals*, and (3) *promiscuous delivery goals* respectively. We will call goals of the second and third kind jointly *promiscuity goals*.

Success goals concern only successful trajectories in which the packet is not spoofed when created nor delivered promiscuously when consumed. Spoofing goals consider only trajectories with spoofing at creation; promiscuous delivery goals, trajectories with promiscuous delivery. The spoofing and promiscuous delivery goals overlap. Any $\psi_B@B \to \phi@R \to \psi_E@E$ splits into at most three goals, each of which applies in only one of these cases:

$$\begin{aligned}(\psi_B \wedge \mathtt{sa}(p) \in IP(B))@B &\to \phi@R \to (\psi_E \wedge \mathtt{da}(p) \in IP(E))@E\\ (\psi_B \wedge \mathtt{sa}(p) \notin IP(B))@B &\to \phi@R \to \psi_E@E\\ \psi_B@B &\to \phi@R \to (\psi_E \wedge \mathtt{da}(p) \notin IP(E))@E\end{aligned}$$

This decomposition is useful, because when we treat goals of the first kind, we must be careful to enforce them tightly. When preventing all counterexamples, we want to ensure that any non-counterexample that satisfies the assumptions remains possible. These are successful trajectories, and a network that filters them unnecessarily is less successful functionally than it could be.

On the other hand, promiscuity goals, i.e. goals of the second and third kind, may be enforced cavalierly or brusquely, i.e. with less attention to allowing as many trajectories as possible compatible with the goals. A trajectory that satisfies the assumptions but is not a counterexample may be filtered out, since it would not be a successful trajectory. Thus, no functionality is sacrificed if it is discarded.

For this reason, in this paper we will focus on identifying how to enforce the success goals precisely.

3.4 Case Study

We consider the network depicted in Figure 1 composed of three subnetworks `Sensitive`, `Trusted` and `Untrusted`, two internal firewalls `fw1` and `fw2`, and two frontier firewalls `fw3` and `fw4` connecting to the `Internet`. This represents a typical scenario where internal firewalls filter traffic among subnetworks while frontier firewalls filter traffic from/to the Internet.

In our example, `Sensitive` subnetwork contains important servers and data and is connected to the `Internet` through the firewalls `fw1` and `fw3`; `Trusted` subnetwork is composed of trusted hosts that, for example, can access services

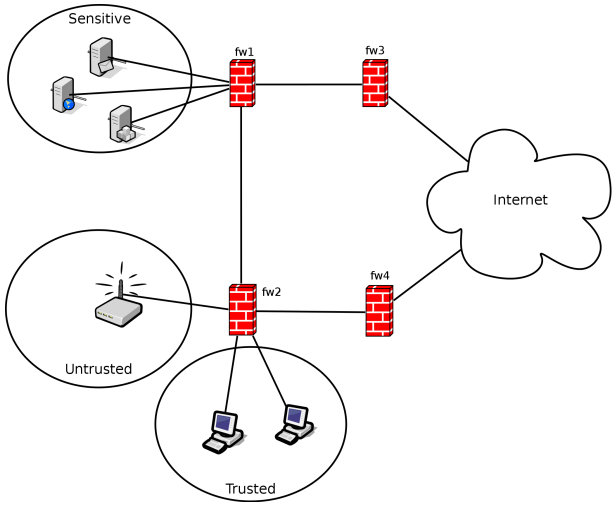


Fig. 1. A simple network with two internal and two frontier firewalls.

hosted in the `Sensitive` subnetwork; `Untrusted` is a wifi subnetwork which provides controlled access to the `Internet` but not to services hosted in `Sensitive`. Both `Trusted` and `Untrusted` are connected to the firewall router `fw2` which in turn is connected to the other internal firewall `fw1`, and to the `Internet` through the frontier firewall `fw4`.

We illustrate how some security constraints for the example network can be naturally specified in the form of security goals. The full specification of security goals will be given in 5.3.

Example 11 (Network isolation). We require that hosts in the `Sensitive` subnetwork never connect to `Untrusted` and vice-versa. This is naturally expressed through two-region statements in which R corresponds to B or E (cf. Section 3.1). We write **false** to denote an unsatisfiable constraint corresponding to the empty set of packets.

$$\texttt{Sensitive} \rightarrow \textbf{false}@\texttt{Sensitive} \rightarrow \texttt{Untrusted}$$
$$\texttt{Untrusted} \rightarrow \textbf{false}@\texttt{Sensitive} \rightarrow \texttt{Sensitive}$$

In the first rule we block packets directly at the source, in order to protect against disclosure of packets starting from `Sensitive` and directed to `Untrusted`. In the second rule, we prevent potentially harmful packets, originated in `Untrusted`, from entering `Sensitive`.

Example 12 (Typical firewall rule). We let `Sensitive` access the `Internet` only via https (destination port 443). We write **dp_443** to denote all packets with destination port 443.

$$\texttt{Sensitive} \rightarrow \textbf{dp\_443}@\texttt{Sensitive} \rightarrow \texttt{Internet}$$

As above, we constrain packets directly at the source in order to prevent disclosure. In particular, notice that packets will never leave `Sensitive` unencrypted.

Example 13 (Traversal control and three region rules). `Untrusted` should access the `Internet` exclusively through `fw3` on port 80. This is a form of traversal control that can be compiled into region control rules by taking cut $\{\texttt{fw3}, \texttt{fw4}\}$ and forbidding traversal of everything but `fw3`, i.e., `fw4` (cf. Section 3.1). In a real setting, we might require this because `fw3` is more powerful than `fw4`; for instance, when it can handle complex (stateful) protocols and can audit what the untrusted users do.

$$\begin{aligned}\texttt{Untrusted} &\rightarrow \mathbf{dp\_80}@\texttt{fw3} \rightarrow \texttt{Internet}\\ \texttt{Untrusted} &\rightarrow \mathbf{false}@\texttt{fw4} \rightarrow \texttt{Internet}\end{aligned}$$

Notice that we cannot express the above goals using only two regions since they specify different requirements on the traversed intermediate nodes.

4 Localizing Security Policies Without Network Address Translation

In this section, we describe how to assign filtering functionality to routers so as to enforce a set of region control statements. For this, we will develop our method based on a well-known matrix-based algorithm. To explain it, we will start from the traditional version, which is applicable in the special case where the network has no nodes that perform NAT. If there are NAT nodes, we need to represent the effect of the various NAT nodes traversed along a path as a relation that composes their individual effects; we will identify in Section 6 some assumptions on their network position and behavior.

When the network has no nodes configured to do network address translation, then every trajectory has the same packet throughout. Thus, only the position of the packet changes as it progresses; its headers remain the same. This leads to two simplifications. First, it is easy to compare a property of headers at one location with the effects it has at other locations; for instance, when packets not satisfying ϕ are discarded at some point of a path, the consequence is that packets reaching some subsequent point along that path satisfy ϕ. Second, non-simple paths, which may revisit the same node more than once, never create any new behavior. Since the only effect of a router may be to discard packets, the set of packets that can traverse a path is a subset of the packets that can traverse any of its subpaths. Hence if any traversal provides a counterexample to a security goal, then we may assume that it is the result of appending two simple paths, one from the beginning region B to the intermediate region R, and another from region R to the end region E. In subsequent sections we will lift these simplifications, but the backbone of our analysis will be clearer in an exposition that can rely on them.

Specifying which packets to keep. We focus on the success goals. Fix a particular pair of endpoints B, E. Thus, we have a collection of statements of the form $\psi_B@B \to \phi_0@R \to \psi_E@E$; because these are success goals, ψ_B, ψ_E ensure that packets contain suitable addresses:

$$\psi_B(p) \Rightarrow \mathtt{sa}(p) \in IP(B) \quad \text{and} \quad \psi_E(p) \Rightarrow \mathtt{da}(p) \in IP(E).$$

Different goals in this collection may have different choices of ψ_B and ψ_E. Since trajectories do not alter packet properties in the no-NAT case, we can equivalently rewrite them to use uniform guards by replacing them with this equivalent form:

$$[\mathtt{sa}(p) \in IP(B)]@B \to [\psi_B \wedge \psi_E \Rightarrow \phi_0]@R \to [\mathtt{da}(p) \in IP(E)]@E$$

Thus, we have essentially moved the variability in ψ_B, ψ_E from the endpoints to R, creating a new formula ϕ_1 at R. Thus, we will now assume that all B, E goals have the same guard formulas at B and E, namely $\mathtt{sa}(p) \in IP(B)$ and $\mathtt{da}(p) \in IP(E)$. We will however keep writing ϕ for the generic form of a formula required at the intermediate location R, even if it has been rewritten as shown above.

Fix a choice of B, E. We will write $P_{B,E}$ for the set of packets with $\mathtt{sa}(p) \in IP(B)$ and $\mathtt{da}(p) \in IP(E)$.

Definition 14. *Suppose given a non-empty collection G of success goals rewritten if necessary to produce uniform ψ_B, ψ_E. We define $\mathsf{Prmt}_{B,E}(\ell)$ to be:*

$$\mathsf{Prmt}_{B,E}(\ell) = P_{B,E} \cap \bigcap_{\phi_\ell} \phi_\ell,$$

taking the intersection over all of the ϕ_ℓ such that a formula $\psi_B@B \to \phi_\ell@\ell \to \psi_E@E$ appears in G. We may write $\mathsf{Prmt}(\ell)$ whenever B, E are clear from the context. ///

Thus, $\mathsf{Prmt}_{B,E}(\ell)$ always includes the packets which are permitted to appear in ℓ, as part of a successful trajectory from B to E. A packet is *worth keeping* at a location if it can use that location to get from B to E along some path that traverses only locations at which it is permitted:

Definition 15 (Keep). *Packet $p \in P_{B,E}$ is* worth keeping along path π *from B to E iff, for every ℓ along π, $p \in \mathsf{Prmt}_{B,E}(\ell)$.*

Packet $p \in P_{B,E}$ is worth keeping from B *to E at ℓ iff there exists some π such that π leads from B to E and traverses ℓ, and p is worth keeping along path π.*

We write $\mathsf{KEEP}_{B,E}(\ell)$ for the set of $p \in P_{B,E}$ worth keeping from B to E at ℓ. We write $\mathsf{KEEP}(\ell)$ whenever B and E are clear from the context. ///

If a packet is permitted at all locations along some path from B to E that passes through ℓ, then it is certainly permitted at location ℓ:

Lemma 16. *If* $p \in \mathsf{KEEP}_{B,E}(\ell)$*, then* $p \in \mathsf{Prmt}_{B,E}(\ell)$.

Notice that a packet going from B to E is permitted at a given location ℓ only if it does not contradict any of the possible region control rules. So, for a packet to be worth keeping from B to E, it is enough if there is a single path π in which p is allowed to traverse all locations of π. However, to ensure that packets use these locations only to follow permissible paths requires the filters we will generate in Def. 18.

By the second of the simplifications mentioned at the beginning of Section 4, this definition is unchanged whether we consider all π or only the paths π in which the part before ℓ is simple, and the part after is too.

The direct way of computing $\mathsf{KEEP}(\ell)$ would thus be to examine every simple path π_1 from B to ℓ, and every path π_2 from ℓ to E, taking an intersection of the $p \in P_{B,E}$ permitted at every step of $\pi_1^\frown\pi_2$, and combining the results via a union over all choices of π_1 and π_2. We present a simpler way using matrix multiplication in Section 4.

Combining the keep sets for different endpoints. When we define $\mathsf{KEEP}_{B,E}$, we work only with packets $p \in P_{B,E}$. We will now assume:

Assumption 17. For all locations ℓ, ℓ', if $\ell \neq \ell'$, then $IP(\ell) \cap IP(\ell') = \emptyset$.

Hence these packets are disjoint from the packets of interest when computing any other $\mathsf{KEEP}_{B',E'}$. Thus, we may simply repeat the computation for each distinct pair B, E, accumulating the union of the KEEP sets for each ℓ.

The resulting union consists of all packets that may traverse ℓ along a successful trajectory from the stated source to the stated destination without encountering a location at which it is not permitted. Thus, we may define, for a given set of goal statements G:

$$\mathsf{KEEP}_*(\ell) = \bigcup_{B,E} \mathsf{KEEP}_{B,E}(\ell) \tag{1}$$

We would like now to filter packets as they are passing from one location to another location at which they should not be kept, i.e., we should discard the complement of $\mathsf{KEEP}_*(\ell')$ along any edge $a\colon \ell \to \ell'$. We summarize this idea in filters for the arcs $a\colon \ell \to \ell'$. Since below we use the complement, the set of packets that should be accepted along a, we formalize that as the accept filter af.

Definition 18. *The* acceptance filter af *is the function from arcs to sets of packets defined* $\mathsf{af}(a) = \mathsf{KEEP}_*(\ell')$*, when* $a : \ell \to \ell'$ *is an arc from* ℓ *to* ℓ'*. We also write* $\mathsf{af}(\ell, \ell')$ *for* $\mathsf{af}(a)$*.*

We define the redundant filter *of an arc* $a : \ell \to \ell'$ *from* ℓ *to* ℓ' *as* $\mathsf{rf}_{\ell,\ell'} = \mathsf{KEEP}_*(\ell) \cap \mathsf{KEEP}_*(\ell')$. ///

Intuitively, af assumes that all firewalls cooperate, while rf redundantly reenforces filtering at each firewall. Thus, when all firewalls behave according to their specifications, af and rf have exactly the same effect. However, rf is more robust if any firewalls may be compromised.

As mentioned before, we always require that $B \neq E$ in any goal statement.

Theorem 19 (Filter security). *Let G be a non-empty collection of success goals, for a* NAT*-free frame. Let $\vec{e} = \langle e_0, \ldots e_k \rangle$ be a success trajectory such that $\mathsf{sender}(\mathsf{chan}(e_0)) = B$ and $\mathsf{rcpt}(\mathsf{chan}(e_k)) = E$. Suppose that, for all $0 \leq i \leq k$ and locations ℓ, ℓ', if $\mathsf{sender}(\mathsf{chan}(e_i)) = \ell$ and $\mathsf{rcpt}(\mathsf{chan}(e_i)) = \ell'$, then*

Case 1. $\mathsf{msg}(e_i) \in \mathsf{af}(\ell, \ell')$
Case 2. $\mathsf{msg}(e_i) \in \mathsf{rf}(\ell, \ell')$.

Then $\vec{e}$ satisfies all of the success goals $\psi_B@B \to \phi@R \to \psi_E@E$ in G.

Proof. First, since we are assuming that there are no NATs, we have a p such that, for all i, $\mathsf{msg}(e_i) = p$. Since $\vec{e}$ is a success trajectory, $\mathsf{sa}(p) \in IP(B)$ and $\mathsf{da}(p) \in IP(E)$.

1. First suppose that $R \neq B$. If $\vec{e}$ never traverses R, then the success goal is vacuously satisfied. So let e_i be the earliest event such that $\mathsf{rcpt}(\mathsf{chan}(e_i)) = R$. By the definition of af, $p \in \mathsf{KEEP}(R)$. By Lemma 16, $p \in \mathsf{Prmt}(R)$. So p satisfies ϕ.

If $R = B$, then we use the fact that $B \neq E$. Thus, p traverses at least one edge to $\mathsf{rcpt}(\mathsf{chan}(e_0))$. Hence, $p = \mathsf{msg}(e_0) \in \mathsf{af}(B, \mathsf{rcpt}(\mathsf{chan}(e_0)))$. By the definition, $p \in \mathsf{KEEP}(\mathsf{rcpt}(\mathsf{chan}(e_0)))$. Hence, there is at least one path π that traverses $\mathsf{rcpt}(\mathsf{chan}(e_0))$ such that $p \in \mathsf{Prmt}(\ell)$ for every ℓ along π. But B appears at the beginning of every path (including π) from B to E. Thus, $p \in \mathsf{Prmt}(B)$, so p satisfies ϕ.

2. The preceding argument applies *a fortiori*, since $\mathsf{rf}(a) \subseteq \mathsf{af}(a)$ for every a.

Moreover, af is maximally successful among all filtering strategies that enforce the success goals. That is, any assignment of filters that permits additional successful trajectories allows counterexamples to some goal. Indeed, we prevent a successful trajectory only if that trajectory is incompatible with the security goals.

Theorem 20 (Maximal success). *Suppose that f is a function from arcs to sets of packets, and for all $a : \ell \to \ell'$, either*

Case (a) $\mathsf{af}(a) \subseteq f(a)$, *or else*
Case (b) $\mathsf{rf}(a) \subseteq f(a)$.

Suppose that τ is a successful trajectory compatible with f but not with the selected filters af *or* rf*. Then τ is a counterexample to some success goal.*

Proof. **Assuming case (a):** Since τ is incompatible with af, it traverses some edge $a : \ell \to \ell'$ such that, letting the packet of τ be p, $p \in \mathsf{KEEP}_*(\ell)$ but $p \notin \mathsf{KEEP}_*(\ell')$. Therefore there is no path π from the start of τ to its endpoint that traverses ℓ' such that $p \in \mathsf{Prmt}(\ell_1)$ for all ℓ_1 along π.

Assuming case (b): Since τ is incompatible with rf, it traverses some edge $a : \ell \to \ell'$ such that, letting the packet of τ be p, $p \notin \mathsf{KEEP}_*(\ell)$ or $p \notin \mathsf{KEEP}_*(\ell')$. Therefore, letting ℓ'' be either ℓ or ℓ', there is no path π from the start of τ to its endpoint that traverses ℓ'' such that $p \in \mathsf{Prmt}(\ell_1)$ for all ℓ_1 along π.

Hence, in either case (a) or case (b), the sequence of locations traversed in τ is not such a path. Thus, τ is a counterexample to some goal between these endpoints.

Computing the sets to keep. Observe that sets of packets form a boolean algebra, and therefore surely a ring where $\cap$ is the multiplication and $\cup$ is the addition. In particular, $\cap$ distributes over $\cup$. Thus, we can form matrices of sets, and matrix multiplication accumulates the $\cup$ of the $\cap$s of corresponding elements. I.e. if we define the inner product $\vec{a} \cdot \vec{b}$ to be:

$$\bigcup_i (a[i] \cap b[i]),$$

then the matrix multiplication $A\,B$ yields C, where $C_{ij} = \vec{a}_i \cdot \vec{b}_j$ using the i^{th} row vector of A and the j^{th} column vector of B. Let:

A be the adjacency matrix for the graph, where if there is an edge from node i to node j, then the entry A_{ij} is the top element, i.e. the set of all packets. $A_{ij} = \emptyset$ if i, j are not adjacent.

K be the diagonal matrix with entries $K_{i,i} = \mathsf{Prmt}(i)$.

We want to compute the matrices Rch^m such that each entry

$\mathsf{Rch}^m_{i,j}$ is the set of $p \in P_{B,E}$ that can reach node j from node i along a path of length $\leq m$ while traversing only locations n such that $p \in \mathsf{Prmt}(n)$.

We claim:

$\mathsf{Rch}^0 = K$**,** since paths of length 0 lead only from i to i, and $K_{i,i}$ is the set of packets permitted there.

$\mathsf{Rch}^1 = K + (K\,A\,K)$**.** A path of length ≤ 1 is either empty or else it takes one step from i to an adjacent location j; moreover, the packet should satisfy $\mathsf{Prmt}(i)$ before the step and $\mathsf{Prmt}(j)$ after it.

$\mathsf{Rch}^{(2m)} = (\mathsf{Rch}^m\,\mathsf{Rch}^m)$**,** since the paths of length $\leq 2m$ are just the paths that divide into two paths of length $\leq m$, respectively ending and beginning at the same node k.

Since every (simple) path visits each node at most once, it is no longer than $|\mathcal{N}|$, the cardinality of the set of nodes. As remarked above, non-simple paths allow no additional packets, since they subject the packets to additional constraints. Thus, the sequence stabilizes by Rch^b where $b = 2^{1+\log_2 |\mathcal{N}|}$, and we define:

$\mathsf{Rch} = \mathsf{Rch}^b$ is the fixed point of Rch^m.

Observe that this computation requires $\mathcal{O}(\log_2 |\mathcal{N}|)$ matrix multiplications to reach its fixed point, and is thus tractable for large $|\mathcal{N}|$, assuming that the underlying ring operations on sets are tractable.

Lemma 21. *For a configuration without* NAT *or other packet transformations,* $\mathsf{KEEP}(i) = \mathsf{Rch}_{B,i} \cap \mathsf{Rch}_{i,E}$.

Proof. Set $\mathsf{Rch}_{B,i} \cap \mathsf{Rch}_{i,E}$ contains all packets $p \in P_{B,E}$ that can reach i from B and then E from i while traversing only locations n such that $p \in \mathsf{Prmt}_{B,E}(n)$. Thus, p belongs to $\mathsf{Rch}_{B,i} \cap \mathsf{Rch}_{i,E}$ if and only if $p \in P_{B,E}$ and there exists some π such that π leads from B to E and traverses i, and for every ℓ along π, $p \in \mathsf{Prmt}_{B,E}(\ell)$. By Definition 15 we directly obtain $\mathsf{Rch}_{B,i} \cap \mathsf{Rch}_{i,E} = \mathsf{KEEP}(i)$.

Tightening given filters. Suppose we want to calculate the success filters relative to some given filters $f(e, d)$, where e is an edge, d is a direction (inbound vs. outbound), and the resulting value $f(e, d)$ is the set of packets we will permit to travel along edge e in direction d. We would like to derive maximally permissive filters that tighten the given ones and enforce the goal statements. To do so, instead of starting with the adjacency matrix A, we define A^f to have the entry $f(e, d)$ in position $A^f_{i,j}$ if edge e leads from location i to location j when traversed in direction d. Like A, it contains the empty set whenever i and j are not adjacent. The successive matrices $\mathsf{Rch}^0, \mathsf{Rch}^1, \ldots, \mathsf{Rch}^m$ are now computed as before, starting with matrix A^f.

For instance, we might like to use this idea to protect terminal networks—meaning a network segment ℓ with just one connection to the remainder of the network—from inappropriate packets. Suppose that a ℓ contains IP addresses $IP(\ell)$. Thus, the remainder of the network has the complementary set of IP addresses $\overline{IP(\ell)}$.

Then we would like to permit packets p outbound only if $\mathtt{sa}(p) \in IP(\ell)$ and $\mathtt{da}(p) \in \overline{IP(\ell)}$. We would like to permit packets p inbound only if $\mathtt{da}(p) \in IP(\ell)$ and $\mathtt{sa}(p) \in \overline{IP(\ell)}$. Edges that do not connect a terminal network retain their original specification of allowing all packets. This leads to a useful policy A^f to start from in computing the sets Rch.

Firewall Configuration. We now define firewall behaviour $\mathcal{FW}$ based on the KEEP_* sets. We consider the case of nodes connected to at least one firewall or, in other words, we assume that any edge in the network has filtering capabilities.

Definition 22. *For each firewall ℓ_f, consider any input and output channels c_i and c_o such that $\mathsf{rcpt}(c_i) = \ell_f$ and $\mathsf{sender}(c_o) = \ell_f$. Let ℓ_i and ℓ_o be the two locations connected to c_i and c_o, i.e., $\ell_i = \mathsf{sender}(c_i), \mathsf{rcpt}(c_o) = \ell_o$. Then we define a firewall as:*

$$\mathcal{FW}_a(t, c_i, c_o, p) \triangleq \begin{cases} \{p\} & \text{if } p \in \mathsf{af}_{\ell_i,\ell_f} \cap \mathsf{af}_{\ell_f,\ell_o} \\ \{\} & \text{otherwise} \end{cases}$$

and, similarly, a redundant *firewall as:*

$$\mathcal{FW}_r(t, c_i, c_o, p) \triangleq \begin{cases} \{p\} & \textit{if } p \in \mathsf{rf}_{\ell_i,\ell_f} \cap \mathsf{rf}_{\ell_f,\ell_o} \\ \{\} & \textit{otherwise} \end{cases} \qquad ///$$

Intuitively, a firewall without network address translation accepts a packet p whenever p is accepted on both the input and the output channels, i.e., whenever p belongs to $\mathsf{af}_{\ell_i,\ell_f} \cap \mathsf{af}_{\ell_f,\ell_o}$, or to $\mathsf{rf}_{\ell_i,\ell_f} \cap \mathsf{rf}_{\ell_f,\ell_o}$ for the redundant firewall.

Theorem 23. *Let G be a non-empty collection of success goals. If each channel is connected to at least one router and the behaviour of all routers is determined by $\mathcal{FW}_a(t, c_i, c_o, p)$ or $\mathcal{FW}_r(t, c_i, c_o, p)$, then frame $\mathcal{F}$ enforces G.*

Proof. Suppose that, in order to get a contradiction, the firewalls are defined as above but frame $\mathcal{F}$ *does not enforce* some goal of G. This means that there is a goal $\psi_B@B \rightarrow \phi@R \rightarrow \psi_E@E$, an execution $\mathcal{B}$ of $\mathcal{F}$, and a trajectory $\vec{e} = \langle e_0, \ldots e_k \rangle$ in $\mathcal{B}$ such that $\vec{e}$ is a counterexample for this goal. By Theorem 19, there exists $0 \leq i \leq k$ such that $\mathsf{msg}(e_i) \notin \mathsf{af}(\ell, \ell')$ and $\mathsf{msg}(e_i) \notin \mathsf{rf}(\ell, \ell')$

Recall that B and E are required to be end hosts and cannot be routers. By hypothesis we have that any edge of $\mathcal{F}$ has at least one firewall so, if the firewall is located at ℓ we consider the *incoming* event e_{i-1} (from ℓ_i to ℓ) and we obtain $\mathcal{FW}_a(t, \mathsf{chan}(e_{i-1}), \mathsf{chan}(e_i), p) = \{\}$, $\mathcal{FW}_r(t, \mathsf{chan}(e_{i-1}), \mathsf{chan}(e_i), p) = \{\}$ as $p \notin \mathsf{af}(\ell_i, \ell) \cap \mathsf{af}(\ell, \ell')$, $p \notin \mathsf{rf}(\ell_i, \ell) \cap \mathsf{rf}(\ell, \ell')$; if the firewall is located at ℓ' we consider the *outgoing* event e_{i+1} (from ℓ' to ℓ_o) and again $\mathcal{FW}_a(t, \mathsf{chan}(e_i), \mathsf{chan}(e_i + 1), p) = \{\}$ and $\mathcal{FW}_r(t, \mathsf{chan}(e_i), \mathsf{chan}(e_i + 1), p) = \{\}$, as $p \notin \mathsf{af}(\ell, \ell') \cap \mathsf{af}(\ell', \ell_o)$, $p \notin \mathsf{rf}(\ell, \ell') \cap \mathsf{rf}(\ell', \ell_o)$.

In both cases, the router behaviour defined in Section 2 enforces that the p is blocked hence we get a contradiction on the existence of the counterexample trajectory.

Example 24 (Localization of traversal control). Consider again the traversal control rules of Example 13:

$$\texttt{Untrusted} \rightarrow \textbf{dp\_80}@\texttt{fw3} \rightarrow \texttt{Internet}$$
$$\texttt{Untrusted} \rightarrow \textbf{false}@\texttt{fw4} \rightarrow \texttt{Internet}$$

We show how they are localized using the redundant firewall $\mathcal{FW}_r(t, c_i, c_o, p)$. We only report cases where we have nonempty set of packets and we write `Un` for `Untrusted` and `Int` for `Internet`:

$$\mathcal{FW}^1_r(t, (\texttt{fw2}, \texttt{fw1}), (\texttt{fw1}, \texttt{fw3}), p) \triangleq \begin{cases} \{p\} & \text{if } \textbf{dp\_80}(p) \\ \{\} & \text{otherwise} \end{cases}$$

$$\mathcal{FW}^2_r(t, (\texttt{Un}, \texttt{fw2}), (\texttt{fw2}, \texttt{fw1}), p) \triangleq \begin{cases} \{p\} & \text{if } \textbf{dp\_80}(p) \\ \{\} & \text{otherwise} \end{cases}$$

$$\mathcal{FW}^3_r(t, (\texttt{fw1}, \texttt{fw3}), (\texttt{fw3}, \texttt{Int}), p) \triangleq \begin{cases} \{p\} & \text{if } \textbf{dp\_80}(p) \\ \{\} & \text{otherwise} \end{cases}$$

$$\mathcal{FW}^4_r(t, c_i, c_o, p) \triangleq \{\}$$

We notice that the above rules enable the path `Untrusted`, `fw2`, `fw1`, `fw3`, `Internet` for packets with destination port 80. Packets trying to reach `fw4` from `fw2` are dropped in `fw2`.

5 Semantic Based Implementation

The theory developed so far considers a very general notion of firewall $\mathcal{FW}$ whose behaviour depends on the firewall history and on the actual input and output channels. We now show how this general notion can be implemented using an extension of Mignis [1], a semantic based tool for firewall configuration in Linux systems. Mignis has a formal semantics that permits us to formally prove correctness with respect to our frame semantics and, at the same time, is provided with an open-source compiler,[5] that will allow us to produce working Linux firewalls, as illustrated in Section 5.2. Notice that Mignis is a general firewall language and is not tailored only to Linux systems. Thus, in principle, Mignis could also generate configurations for other firewall systems.

5.1 Mignis and its Semantics

Mignis [1] is a declarative specification language in which the order of rules does not matter. This makes the specification of a firewall very close to its semantics: a packet goes through (possibly translated) only if it matches a positive rule and is not explicitly denied. This allows administrators to write and inspect rules independently of the order in which they are specified. Moreover, the declarative approach makes it easy to detect possible conflicts and redundancies and to query for a subset of the specification involving specific hosts. Mignis supports Network Address Translation (NAT), i.e., it allows the translation of the source and destination addresses of a packet while it crosses the firewall, and it applies the translation consistently to all packets belonging to the same connection.

Mignis rules are defined in terms of address ranges `n`. An address range `n` is a pair consisting of a set of IP addresses and a set of ports, denoted $IP(\texttt{n})$:$port(\texttt{n})$. Given an address `addr`, we write $\texttt{addr} \in \texttt{n}$ to denote $port(\texttt{addr}) \in port(\texttt{n})$, if $port(\texttt{addr})$ is defined, and $IP(\texttt{addr}) \in IP(\texttt{n})$. Notice that if `addr` does not specify a port (for example in ICMP packets) we only check if the IP address is in the range. We will use the wildcard $*$ to denote any possible address or port or address range, and ϵ to denote a special range consisting of a special address $\epsilon_{\texttt{addr}}$ used to note void translations.

Syntax We present a version of Mignis (that we call $\texttt{Mignis}^+$) that extends the original one in various respects: (*i*) rules are localized on channels $\mathcal{CH}$ in order to allow for packet filtering that depends on the network topology; (*ii*) packets on established connections are not accepted by default; (*iii*) rules are all positive. $\texttt{Mignis}^+$ is slightly more complex to use but strictly more expressive than the

[5] `https://github.com/secgroup/Mignis`

$$[\![\langle\rangle]\!]\, t\, c_i\, c_o\, p \triangleq \{\}$$

$$[\![\mathtt{n_s}\ [\mathtt{n_s^t}]\ @\ \mathtt{c_i} > \mathtt{n_d}\ [\mathtt{n_d^t}]\ @\ \mathtt{c_o} : \phi\ ;\ C]\!]\, t\, c_i\, c_o\, p \triangleq [\![C]\!]\, t\, c_i\, c_o\, p \cup \begin{cases} \{p[\mathtt{sa} \mapsto \mathtt{a_s^t}, \mathtt{da} \mapsto \mathtt{a_d^t}] \mid \mathtt{a_s^t} \in \mathtt{n_s^t}, \mathtt{a_d^t} \in \mathtt{n_d^t}\} \\ \qquad \text{if } \mathtt{c_i} = c_i, \mathtt{c_o} = c_o, \phi_{\mathtt{n_s},\mathtt{n_d}}(p,t) \\ \{\} \qquad \text{otherwise} \end{cases}$$

Table 2. $\texttt{Mignis}^+$ semantics.

original one. Since we will generate $\texttt{Mignis}^+$ configurations automatically we do not consider the increased complexity as a problem. The $\texttt{Mignis}^+$ syntax follows:

$$r ::= \mathtt{n_s}\ [\mathtt{n_s^t}]\ @\ \mathtt{c_s} > \mathtt{n_d}\ [\mathtt{n_d^t}]\ @\ \mathtt{c_d} : \phi$$

where ϕ is a predicate that is checkable over a packet and the firewall state (represented in terms of a trace, as defined in Section 2.1). Intuitively, this rule accepts a packet p from $\mathtt{n_s}$ to $\mathtt{n_d}$ that is received from channel $\mathtt{c_s}$ and is routed to channel $\mathtt{c_d}$ whenever $\phi(p,t)$ holds. Packet p's source and destination addresses are translated into different ones belonging to $\mathtt{n_s^t}$ and $\mathtt{n_d^t}$ in order to support NAT. When $\mathtt{n_s^t}$ and $\mathtt{n_d^t}$ are ϵ the rule leaves the addresses unchanged. When, instead, $\mathtt{n_s^t}$ and $\mathtt{n_d^t}$ are different from ϵ, they respectively correspond to source and destination NATs, and if both source and destination NATs are specified they are combined together.[6] A sequence of these firewall rules is called a *configuration*, $C ::= r; C \mid \langle\rangle$

Semantics Mignis implements NAT by keeping track of the established connections and the relative address translations. In this paper we represent $\texttt{Mignis}^+$ state as a trace representing previous sent and received packets with the respective channels, as defined in Section 2.1. Let $PK = [(\mathcal{CH} \times \mathcal{D})^* \to \mathcal{CH} \to \mathcal{CH} \to P \to \mathcal{P}(P)]$ be the domain of packet transformers. We define $[\![\cdot]\!] : C \to PK$, i.e., a function mapping a $\texttt{Mignis}^+$ configuration C into a packet transformer, representing all the accepted packets with the corresponding translations. $[\![\cdot]\!]\, t\, c_i\, c_o\, p$ is defined inductively in Table 2, where $\phi_{\mathtt{n_s},\mathtt{n_d}}(p,t) \triangleq \mathtt{sa}(p) \in \mathtt{n_s} \wedge \mathtt{da}(p) = \mathtt{n_d} \wedge \phi(p,t)$ and $p[\mathtt{sa} \mapsto \mathtt{a_s^t}, \mathtt{da} \mapsto \mathtt{a_d^t}]$ denotes packet p where $\mathtt{sa}(p)$, respectively $\mathtt{da}(p)$, is replaced by $\mathtt{a_s^t}$, respectively $\mathtt{a_d^t}$, if it is different from the void address $\epsilon_{\mathtt{addr}}$, and left unchanged otherwise.

Intuitively, the empty configuration $\langle\rangle$ corresponds to the empty set. For a configuration $\mathtt{n_s}\ [\mathtt{n_s^t}]\ @\ \mathtt{c_i} > \mathtt{n_d}\ [\mathtt{n_d^t}]\ @\ \mathtt{c_o} : \phi\ ;\ C$ we take all the packets that are received on c_i, routed on c_o and satisfy $\phi_{\mathtt{n_s},\mathtt{n_d}}(p,t)$, whose addresses are translated along non-ϵ NATs $\mathtt{n_s^t}, \mathtt{n_d^t}$, together with the packets returned by the remaining rules in C. By taking this union with the remaining rules in C

[6] Notice that square brackets are used consistently to note the translated addresses. This differs from the original Mignis notation for destination NATs which encloses in square brackets the address before translation, i.e., $\mathtt{n_d}$.

we are indeed considering that there is no ordering in the rules of a $\texttt{Mignis}^+$ configuration.

Definition 25 ($\texttt{Mignis}^+$ firewall). *Given a* $\texttt{Mignis}^+$ *configuration* C*, we let* $\mathcal{FW}(t, c, c', p) \triangleq [\![C]\!]\, t\, c\, c'\, p$.

Example 26. Consider a destination NAT that translates all tcp incoming packets from interface `eth0` directed to 192.168.0.1 port 80 on interface `eth1`, into address 192.168.0.100 on the same port. In $\texttt{Mignis}^+$ this is specified through a configuration C composed of a single rule:

$$*@\texttt{eth0} > \texttt{192.168.0.1:80@eth1}\ [\texttt{192.168.0.100:80}] : \text{tcp}$$

where we omit writing the void $[\epsilon_{\texttt{addr}}]$ source NAT, for readability, and where tcp corresponds to a predicate that holds if and only if p is a tcp packet. We have:

$$[\![C]\!]\, t\, c_i\, c_o\, p = \{p[\texttt{da} \mapsto \texttt{192.168.0.100:80}]\}$$

if $\texttt{da}(p) = \texttt{192.168.0.1:80}, c_i = \texttt{eth0}, c_o = \texttt{eth1}, \text{tcp}(p)$. This firewall will accept any tcp packet from `eth0` with destination `192.168.0.1:80` on `eth1` and will translate its destination to `192.168.0.100:80`. For example, let p be a tcp packet with source `1.2.3.4:5656` and destination `192.168.0.1:80`. We have $[\![C]\!]\, t\, \texttt{eth0}\, \texttt{eth1}\, p = \{p[\texttt{da} \mapsto \texttt{192.168.0.100:80}]\}$. Notice that packet being tcp is independent of the firewall history t, so firewall semantics does not depend on t in this specific example.

5.2 A Tool for Firewall Localization

We have implemented a tool that given a network specification and a set of region control rules automatically localizes the filters and generates $\texttt{Mignis}^+$ configurations for all the firewalls in the network. We have also modified the original Mignis compiler by incorporating the extensions illustrated in section 5.1 in order to produce the actual Linux firewall (Netfilter) configurations. The tool is implemented in Python3.

Network We consider a network consisting of firewalls and end hosts, and we assume that each channel has at least one firewall endpoint. This ensures that all edges can filter packets.

Constraints Given rule $\psi_B@B \rightarrow \phi@R \rightarrow \psi_E@E$, we specify constraints ψ_B, ψ_E and ϕ as Boolean expressions over variables representing the following facts:

Source and destination address whenever a packet has source or destination address of end host `h`, written `sa_h` and `da_h`, respectively;

Source and destination port whenever a packet has a source or destination port `n`, written `sp_n` and `dp_n`, respectively;

Protocol whenever packet protocol is `p`, written `pr_p`;
Established whenever a packet belongs to an established connection, written `est`.

For example, `sp_443` & `est` requires that the source port is 443, and that packets belong to an established connection. This is a typical example of a response from a SSL web server. Notice that we use notations &, | and $\sim$ to represent Boolean AND, OR and NOT, respectively. Rules apply only to non-spoofed, non-promiscuous packets. This is automatically enforced by requiring $\psi_{\mathtt{B}}$&`sa_B` and $\psi_{\mathtt{E}}$&`sa_E` in place of ψ_B and ψ_E.

Localization We compute localization as described in section 4. At the moment the prototype only supports the case without NATs described in section 4. We leave the extension to NATs as a future work. Constraints are represented as reduced, ordered binary decision diagrams (ROBDD) [4]. We base our implementation on the Python EDA library that supports both Boolean algebra and ROBDDs.[7] The advantage of adopting ROBDD representation is that it is a canonical form and makes it very efficient to compute equivalence between Boolean expressions, which is useful to determine when the computation of Rch stabilizes. Set union and intersection used for localization in section 4 naturally become OR and AND Boolean operators over ROBDDs.

Firewall configuration Once we obtain $\mathcal{FW}_a(t, c_i, c_o, p)$ or $\mathcal{FW}_r(t, c_i, c_o, p)$ in the form of a Boolean expression we generate `Mignis`$^+$ configurations by instantiating the expression with the source and destination addresses of any possible end host, and by computing the solutions of the obtained boolean expression. It is worth noticing that Python EDA transparently invokes PicoSAT solver [8] to efficiently solve a Boolean expression. Solutions are then translated into constraints over ports, protocol and established state.

5.3 Case Study in Detail

We now define the security goals for the example network of Figure 1 and discussed in Section 3.4.

As already mentioned, firewalls usually keep track of established connections so that packets belonging to the same connections are not filtered. This is particularly useful to enable bidirectional communication without necessarily opening the firewall bidirectionally to a new connection: it is enough to open the firewall in one direction and let *established* packets come back. We write `est` to note that a packet is established (cf. Section 5.2). While specifying rules, we proceed pair by pair, defining the rules and their established counterpart (when needed) at the same time.

[7] `http://pyeda.readthedocs.org/`
[8] `http://fmv.jku.at/picosat/`

Hosts in the `Sensitive` and `Trusted` subnetworks should never connect to `Untrusted` and vice-versa. This is naturally expressed through two-region statements (cf. Example 11, Section 3.1):

$$\begin{aligned}
\texttt{Sensitive} &\rightarrow \textbf{false}@\texttt{Sensitive} \rightarrow \texttt{Untrusted}\\
\texttt{Untrusted} &\rightarrow \textbf{false}@\texttt{Sensitive} \rightarrow \texttt{Sensitive}\\
\texttt{Trusted} &\rightarrow \textbf{false}@\texttt{Trusted} \rightarrow \texttt{Untrusted}\\
\texttt{Untrusted} &\rightarrow \textbf{false}@\texttt{Trusted} \rightarrow \texttt{Trusted}
\end{aligned}$$

Hosts in the `Sensitive` subnetwork should never connect to `Trusted`, while hosts from `Trusted` network can access `Sensitive` via ssh through `fw1` without passing through the `Internet` as this would unnecessarily expose network connections to attacks. Notice that we filter packets from `Sensitive` to `Trusted` only if they do not belong to established ssh connections:

$$\begin{aligned}
\texttt{Trusted} &\rightarrow \textbf{dp_22}@\texttt{Sensitive} \rightarrow \texttt{Sensitive}\\
\texttt{Sensitive} &\rightarrow (\textbf{sp_22}\&\texttt{est})@\texttt{Sensitive} \rightarrow \texttt{Trusted}\\
\texttt{Trusted} &\rightarrow \textbf{false}@\texttt{Internet} \rightarrow \texttt{Sensitive}\\
\texttt{Sensitive} &\rightarrow \textbf{false}@\texttt{Internet} \rightarrow \texttt{Trusted}
\end{aligned}$$

`Sensitive` should access the `Internet` only via https (destination port 443), while `Internet` hosts should never start new connections towards `Sensitive`:

$$\begin{aligned}
\texttt{Sensitive} &\rightarrow \textbf{dp_443}@\texttt{Sensitive} \rightarrow \texttt{Internet}\\
\texttt{Internet} &\rightarrow (\textbf{sp_443}\&\texttt{est})@\texttt{Sensitive} \rightarrow \texttt{Sensitive}
\end{aligned}$$

`Trusted` has full access to the `Internet` and from the `Internet` we give access to `Trusted` only via ssh (port 22). When full access is granted, we do not specify any region control statement, but we still need to let established packets go through:

$$\texttt{Internet} \rightarrow (\textbf{dp\_22}|\texttt{est})@\texttt{Trusted} \rightarrow \texttt{Trusted}$$

`Untrusted` should access the `Internet` exclusively through `fw3` on port 80. This is a form of traversal control that can be compiled into region control rules by taking cut $\{\texttt{fw3}, \texttt{fw4}\}$ and forbidding traversal of everything but `fw3`, i.e., `fw4` (cf. Example 13, Section 3.1). `Internet` should never access `Untrusted`:

$$\begin{aligned}
\texttt{Untrusted} &\rightarrow \textbf{dp_80}@\texttt{fw3} \rightarrow \texttt{Internet}\\
\texttt{Internet} &\rightarrow (\textbf{sp_80}\&\texttt{est})@\texttt{fw3} \rightarrow \texttt{Untrusted}\\
\texttt{Untrusted} &\rightarrow \textbf{false}@\texttt{fw4} \rightarrow \texttt{Internet}\\
\texttt{Internet} &\rightarrow \textbf{false}@\texttt{fw4} \rightarrow \texttt{Untrusted}
\end{aligned}$$

Localizing filtering Table 3 in the Appendix reports the output of the localization tool that automatically generated the $\texttt{Mignis}^{+}$ configuration for the four firewalls according to $\mathcal{FW}_r(t, c_i, c_o, p)$. Channels/interfaces are named with prefix `if_` to distinguish them from network end hosts. For example, the first group of rules for firewall `fw1`

```
Sensitive@if_Sensitive:22 > Trusted@if_fw2 | -m ...
Sensitive@if_Sensitive    > Internet@if_fw2:443
```

refer to interfaces `if_Sensitive` and `if_fw2`, i.e., packets coming from the channel connecting `Sensitive` to `fw1` and delivered over the channel from `fw1` to `fw2`. Established requirement is translated into the low level syntax of Linux firewall:

```
-m state --state ESTABLISHED,RELATED
```

We can see that, on these two channels, firewall `fw1` only allows packets from `Sensitive` to `Trusted` on established ssh connections and new https connections from `Sensitive` to `Internet`, as required by the region control rules:

$$\begin{aligned}\texttt{Sensitive} &\rightarrow (\textbf{sp\_22}\&\texttt{est})@\texttt{Sensitive} \rightarrow \texttt{Trusted}\\ \texttt{Sensitive} &\rightarrow \quad \textbf{dp\_443}@\texttt{Sensitive} \quad \rightarrow \texttt{Internet}\end{aligned}$$

We illustrate in detail how the following rules, permitting access from `Unstrusted` to `Internet` only on port 80 through `fw3`, are localized in the firewalls:

$$\begin{aligned}\texttt{Untrusted} &\rightarrow \quad \textbf{dp\_80}@\texttt{fw3} \quad \rightarrow \texttt{Internet}\\ \texttt{Internet} &\rightarrow (\textbf{sp\_80}\&\texttt{est})@\texttt{fw3} \rightarrow \texttt{Untrusted}\\ \texttt{Untrusted} &\rightarrow \quad \textbf{false}@\texttt{fw4} \quad \rightarrow \texttt{Internet}\\ \texttt{Internet} &\rightarrow \quad \textbf{false}@\texttt{fw4} \quad \rightarrow \texttt{Untrusted}\end{aligned}$$

The relevant rules are:

```
FIREWALL fw1
Untrusted@if_fw2 > Internet@if_fw3:80
Internet@if_fw3:80 > Untrusted@if_fw2       | -m ...

FIREWALL fw2
Untrusted@if_Untrusted > Internet@if_fw1:80
Internet@if_fw1:80 > Untrusted@if_Untrusted | -m ...

FIREWALL fw3
Internet@if_Internet:80 > Untrusted@if_fw1  | -m ...
Untrusted@if_fw1 > Internet@if_Internet:80
```

Firewall `fw1` allows for traffic between `fw2` and `fw3` on the required ports and firewalls `fw2` and `fw3` enable traffic through `fw1`. Packets from `Untrusted` to `Internet` are dropped in `fw4` (which contains no rule between those two end hosts), and on the link from `fw2` to `fw4`, as required. Recall that Mignis has a default drop policy, so if no rules for two end hosts exist packets will be dropped.

6 Localizing with NATs

In the more general case, we use the same ideas, although with a different ring. In this case, instead of the ring of sets of packets under $\cup$ and $\cap$, we use a ring of relations on packets. The addition-like operator is again $\cup$, but the multiplication-like operator is now the relative product $R \bowtie S$ of binary relations:

$$(R \bowtie S)(x, z) \text{ iff } \exists y \,.\, R(x, y) \wedge S(y, z)$$

We will next verify that these operations do form a ring.

After that, we face two additional hurdles. First, we cannot hope to enforce goals in an exact way if different regions behind the same NAT are subjected to different security policies. By the time that packets emerge through the NAT, we cannot tell in which region they originated, and thus cannot differentiate their filtering according to their origins. Second, we need an analogue to Assumption 17, which ensured that we could compute the KEEP sets for different endpoints B, E separately, and combine the results by taking disjoint unions. Instead, we will assume that the external *IP* addresses of any distinct NAT devices accessible to any location R are disjoint.

Union and relative product form a ring. We will check that this does form a ring, although it is not a commutative ring. This suffices to allow us to use the methods we have just described. We will also point out below that—for the NAT-based packet transformations that interest us—there are simple and efficient ways to represent the relations that arise in our computations.

1. $\cup$ is associative, commutative, and has unit $\emptyset$.
2. $\bowtie$ is associative, $(R \bowtie S) \bowtie T = R \bowtie (S \bowtie T)$; and it has unit the identity relation Id, meaning $R \bowtie \mathsf{Id} = R$. The former is the equivalence:

$$(\exists z \,.\, (\exists y \,.\, R(x,y) \wedge S(y,z)) \wedge T(z,w)) \equiv \\ (\exists y \,.\, R(x,y) \wedge (\exists z \,.\, S(y,z) \wedge T(z,w)))$$

   while the latter says that

$$\exists y \,.\, R(x,y) \wedge y = z \equiv R(x,z).$$

3. $\bowtie$ distributes over $\cup$:

$$R \bowtie (S \cup T) = (R \bowtie S) \cup (R \bowtie T) \quad \text{and} \\ (S \cup T) \bowtie R = (S \bowtie R) \cup (T \bowtie R).$$

   We check the former, via the definitions, distributing $\wedge$ over $\vee$:

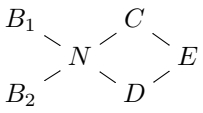
$$\begin{aligned} &(R \bowtie (S \cup T))(x,z) \\ &\equiv \exists y \,.\, R(x,y) \wedge (S(y,z) \vee T(y,z)) \\ &\equiv \exists y \,.\, (R(x,y) \wedge S(y,z)) \vee (R(x,y) \wedge T(y,z)) \\ &\equiv (\exists y \,.\, R(x,y) \wedge S(y,z)) \vee (\exists y \,.\, R(x,y) \wedge T(y,z)) \\ &\equiv ((R \bowtie S) \cup (R \bowtie T))(x,z) \end{aligned}$$

   To check the latter, we use almost the same argument, but inverting the arguments; the commutativity of $\wedge$ and $\vee$ justifies the way we write this:

$$\begin{aligned} &((S \cup T) \bowtie R)(z,x) \\ &\equiv \exists y \,.\, R(y,x) \wedge (S(z,y) \vee T(z,y)) \\ &\equiv \exists y \,.\, (R(y,x) \wedge S(z,y)) \vee (R(y,x) \wedge T(z,y)) \\ &\equiv (\exists y \,.\, R(y,x) \wedge S(z,y)) \vee (\exists y \,.\, R(y,x) \wedge T(z,y)) \\ &\equiv ((R \bowtie S) \cup (R \bowtie T))(z,x) \end{aligned}$$

We may also regard each set of packets $\phi(p)$ as "lifting" to the binary relation $\phi(p) \wedge p = p'$, i.e. the lifted version of ϕ is the intersection $\uparrow \phi = \mathsf{Id} \cap (\phi \times \phi)$.

Agreeing on goals across NATs. Consider the network:



(2)

where we are interested in the packets that are permitted to travel from either B_1 or B_2 through the source NAT device N to E. However, they must satisfy different properties depending on which intermediate node C, D they traverse. In particular, successful trajectories from B_1 to E that traverse C must have property ϕ, while successful trajectories from B_2 to E that traverse C must have property $\neg\phi$. On the other hand, successful trajectories from B_1 to E that traverse D must have property $\neg\phi$, while successful trajectories from B_2 to E that traverse D must have property ϕ.

Unfortunately, we do not want to enforce this requirement before traversing the NAT N, because we do not yet know whether the packets will traverse the route through C or through D. We would need to discard all the packets for E. And we do not want to enforce this requirement after traversing the NAT N, because we do not know whether the packets originated at B_1 or B_2. We would again have to discard them all, since N has rewritten their source addresses to its own external address.

Since the goals differentiate the packets by the source behind the NAT, we cannot enforce them tightly. To prevent non-ϕ packets that originated at B_1 from traversing C, we must also discard non-ϕ packets that originated at B_2.

Hence, all origins beyond a source NAT must be subject to the same policy. Indeed, a purpose of the NAT is to make those packets indistinguishable.

In particular, consider triples of locations B_1, R, E and B_2, R, E, where R is separated from B_1, B_2 by at least one NAT. Then for every goal for B_1, R, E, there should be a goal for B_2, R, E that is at least as restrictive, and conversely. In this case, we can say that B_1, B_2 are *region of origin equi-goals* for R, E. A similar notion of *destination equi-goals* applies to E_1, E_2. We can expect to enforce a policy tightly only when it ensures equi-goals across NATs both for regions of origin and destinations.

Networks with NATs: Assumptions. In real life, NATs are used in very specific ways. First, because the source NAT functionality must always be applied to outgoing packets, and reversed when packets return in response—and conversely for destination NATs—the NAT must separate the portion of the network using private addresses from the remainder of the network. We will here also assume an all-or-nothing NAT policy, namely that if a router provides NAT service for one host that reaches it on a particular interface, then it does so for all hosts reaching it on that interface.

Assumption 27. If a router ℓ provides NAT functionality, then the singleton $\{\ell\}$ is a cut set partitioning $\mathcal{LO} \setminus \{\ell\}$ into an *inside* using private addresses and an *outside*.

NATs can be arranged in several layers, so that addresses used outside an "inner" NAT may include private addresses served by a subsequent "outer" NAT. Nevertheless, at any point of the network, the IP addresses that are locally visible should be disjoint, in an analogue of Assumption 17:

Assumption 28. Suppose that paths π_1, π_2 start from B_1 and B_2 respectively, reaching a shared endpoint R. Then $IP(B_1) \cap IP(B_2) = \emptyset$ unless $B_1 = B_2$ or:

1. π_i traverses a NAT device properly between B_i and R, where $i = 1$ or 2; or
2. R is a NAT device and π_1, π_2 reach it on different interfaces.

We now can define the equi-goal idea:

Definition 29. *A set G of goals has* source equi-goals *iff, for every* NAT *device d, if B_1, B_2 lie on the inside of d and R lies on the outside, then for all goals $\psi_B@B_1 \to \phi@R \to \psi_E@E$, this goal is in G iff $\psi_B@B_2 \to \phi@R \to \psi_E@E$ is in G.*

G has destination equi-goals *iff, for every* NAT *device d, if E_1, E_2 lie on the inside of d and R lies on the outside, then for all goals $\psi_B@B \to \phi@R \to \psi_E@E_1$, this goal is in G iff $\psi_B@B \to \phi@R \to \psi_E@E_2$ is in G.*

G has equi-goals *iff both conditions are met.* ///

Keep sets with NATs. With NATs, packets are no longer unchanged throughout a trajectory. We need to account for these changes, as the packets have different fields at each node, and those fields define if the node keeps or discards the packet.

Fix a choice of endpoints B, E. Unlike the NAT-free case, we cannot simply move the preconditions ψ_B, ψ_E of a success goal from the beginning and end of a trajectory to the region R. After all, NAT transformations may have altered the packet headers between B and R, or R and E. Instead, we will use the relations to keep track of both the state of the packet at the earlier point and the later point. For this reason, we will now alter the set $\mathsf{Prmt}_{B,E}(\cdot)$ to a pair of relations on packets. $\mathsf{Prmt}^{\ell}_{B,R}$ concerns the "left" half of the trajectory from B to R; $\mathsf{Prmt}^{r}_{R,E}$ concerns the "right" half from R to E. Since B, E are fixed, we do not indicate them here; we focus on R.

If G is a set of success goals, we will write $G(B, R, E)$ for the subset of G concerning B, R, E. If $g \in G(B, R, E)$ is a goal $\psi_B@B \to \phi@R \to \psi_E@E$, we will write $\textsc{Pre}_B(g)$, $\textsc{Post}(g)$, and $\textsc{Pre}_E(g)$ for ψ_B, ϕ, and ψ_E, resp.

$$
\begin{aligned}
&\mathsf{Prmt}^{\ell}_{B,R} = \\
&\quad \{(p,q) \colon \textstyle\bigwedge_{g \in G(B,R,E)} \mathsf{sa}(p) \in IP(B) \wedge (\textsc{Pre}_B(g)(p) \\
&\qquad\qquad \Rightarrow \textsc{Post}(g)(q))\} \\
&\mathsf{Prmt}^{r}_{R,E} = \\
&\quad \{(q,p) \colon \textstyle\bigwedge_{g \in G(B,R,E)} \mathsf{da}(p) \in IP(E) \wedge (\textsc{Pre}_E(g)(p) \\
&\qquad\qquad \Rightarrow \textsc{Post}(g)(q))\}.
\end{aligned}
$$

$\mathsf{Prmt}^{\ell}_{B,R}$ contains a pair (p_0, q_0) when p_0 satisfies the precondition $\textsc{Pre}_B$ at the beginning B, and q_0 satisfies the claim $\textsc{Post}(g)$ property at R, meaning that

if a packet could follow some path that would transform it from p_0 to q_0, then this would be allowed by the goal statements. Conversely for $(q_1, p_1) \in \mathsf{Prmt}^r_{R,E}$, if q_1 could follow some path to E that would transform it to p_1, then the goal statements would allow this. A packet q is worth keeping at a given location if there is some real path that leads through R which does transform it in ways that remain permissible:

Definition 30 (nat-Keep). *If $\vec{e} = \langle e_0, \ldots e_k \rangle$ is a trajectory, let $\mathsf{locs}(\vec{e})$ be the sequence of length $k+2$ such that $\mathsf{locs}(\vec{e})[0] = \mathsf{sender}(\mathsf{chan}(e_0))$ and, for all i such that $1 \leq i \leq k+1$, $\mathsf{locs}(\vec{e})[i] = \mathsf{rcpt}(\mathsf{chan}(e_{i-1}))$.*

*Trajectory $\vec{e} = \langle e_0, \ldots e_k \rangle$ is B, E-*worthy *iff $\vec{e}$ is a success trajectory, $\mathsf{locs}(\vec{e})[0] = B$, $\mathsf{locs}(\vec{e})[k+1] = E$, and, for every i where $0 \leq i \leq k+1$,*

$$(\mathsf{msg}(e_0), \mathsf{msg}(e_i)) \in \mathsf{Prmt}^{\ell}_{B,\mathsf{locs}(\vec{e})[i]} \text{ and}$$
$$(\mathsf{msg}(e_i), \mathsf{msg}(e_k)) \in \mathsf{Prmt}^{r}_{\mathsf{locs}(\vec{e})[i],E}.$$

Packet $\mathsf{msg}(e_i)$ is worth keeping from *B to E at R iff for some B, E-worthy trajectory $\vec{e} = \langle e_0, \ldots e_k \rangle$, $R \in \mathsf{locs}(\vec{e})$.*

Let $\mathsf{KEEP}_{B,E}(R)$ be the set of packets worth keeping from B to E at R. ///

In case there are no NATs, then $\mathsf{KEEP}_{B,E}(R)$ receives the same meaning as previously. By the definition:

Lemma 31. *For all q, $q \in \mathsf{KEEP}_{B,E}(R)$ iff there exist p_1, p_2 such that $(p_1, q) \in \mathsf{Prmt}^{\ell}_{B,R}$ and $(q, p_2) \in \mathsf{Prmt}^{r}_{R,E}$.* ///

Combining the keep sets for different endpoints. The equi-goal property and the IP disjointness assumption (Assumption 28) allow us to combine the keep sets for different endpoints.

Lemma 32. *Suppose that the goals G have the equi-goal property, and that $B_1 \neq B_2$ but $\mathsf{KEEP}_{B_1,E}(R) \cap \mathsf{KEEP}_{B_2,E}(R) \neq \emptyset$. Then*

1. *There are success trajectories from B_1 through R to E or from B_2 through R to E that traverse NATs between B_i and R; and*
2. *If B_1, B_2 have source equi-goals for R, E, then $\mathsf{KEEP}_{B_1,E}(R) = \mathsf{KEEP}_{B_2,E}(R)$.*

Suppose that $E_1 \neq E_2$ but $\mathsf{KEEP}_{B,E_1}(R) \cap \mathsf{KEEP}_{B,E_2}(R) \neq \emptyset$. Then

1. *There are success trajectories from B through R to E_1 or from B through R to E_2 that traverse NATs between R and E_i; and*
2. *If E_1, E_2 have destination equi-goals for B, R, then $\mathsf{KEEP}_{B,E_1}(R) = \mathsf{KEEP}_{B,E_2}(R)$.*

Proof. By the non-emptiness condition, there is a B_1, E-worthy $\vec{e_1}$ and a B_2, E-worthy $\vec{e_2}$ both traversing R and with the same packet form p there. If $\vec{e_1}, \vec{e_2}$ have not yet traversed a NAT, then p is in its original form; since these are success trajectories, this contradicts $B_1 \neq B_2$. Thus claim 1 holds.

Since both trajectories provide p at R, the most recent NATs d_1, d_2 that rewrote their source addresses both have $\mathtt{sa}(p) \in IP(d_i)$. Hence by Assumption 28, $d_1 = d_2$. Thus we may apply the definition of equi-goals; so the same conditions occur in the goal statements for B_1, B_2, and the KEEP sets are the same. The claims for E_1, E_2 are symmetric.

That is, equi-goals on NATs ensure that KEEP sets are either equal or disjoint. The information in the headers $\mathtt{sa}(p)$ and $\mathtt{da}(p)$ tells us enough to enforce the goals G tightly. As before, we define, for a given set of goal statements G:

$$\mathsf{KEEP}_*(\ell) = \bigcup_{B,E} \mathsf{KEEP}_{B,E}(\ell) \tag{3}$$

We would like now to filter packets as they are passing from one location to another location at which they should not be kept, i.e., we should discard the complement of $\mathsf{KEEP}_*(\ell')$ along any edge $a\colon \ell \to \ell'$. We do this as before:

Definition 33. *The* acceptance filter af *maps arcs to sets of packets, where* $\mathsf{af}(a) = \mathsf{KEEP}_*(\ell')$*, for each arc* $a : \ell \to \ell'$ *from* ℓ *to* ℓ'*. We write* $\mathsf{af}(\ell, \ell')$ *for* $\mathsf{af}(a)$. ///

We again want to configure our network to discard all packets traversing the edge a that do not belong to the acceptance filter $\mathsf{af}(a)$.

Computing the matrices Rch^m. We regard source NAT operations as occurring outbound on an edge e, and as defined by the relation $R_e(p, p')$. Here, typically, this holds if p has a local source address; p' has the NAT device's address as its source address; and p' agrees with p for destination port and address. The translation for returning packets flowing inbound on the edge is dual.

Suppose now that $F(e, d)$ expresses both the packet rewriting and the filtering that occurs on edge e in direction d (inbound vs. outbound). In particular, $F(e, d)$ is a relation that holds on a pair of packets (p, q) when p is permitted to pass over e in direction d, but is rewritten by the sender to q before transmission. If an entry reflects a filter retaining packets satisfying ϕ without rewriting, then its entry $F(e, d)$ is the lifted relation $\uparrow \phi$, namely the identity relation restricted to ϕ. We now construct, for any pair of endpoints B, E:

A^F contains these relations $A^F_{i,j} = F(e, d)$ in entries i, j, when edge e in direction d leads from i to j. If there is no edge from i to j, then $A^F_{i,j} =\uparrow \emptyset$ is the empty relation.

K is the diagonal matrix where $K_{i,j} =\uparrow \emptyset$ when $i \neq j$, and

$$K_{i,i} =\uparrow \{q\colon \exists p_B, p_E \,.\, \mathsf{Prmt}^{\ell}_{B,i}(p_B, q) \wedge \mathsf{Prmt}^{r}_{i,E}(q, p_E)\}.$$

Thus, $K_{i,i}$ contains the pairs (q, q) where q is permitted to pass through i on a trajectory from B to E, for any forms of the packet at the endpoint.

We now use the same matrix operations to compute Rch^m for the given F. Each Rch^m will contain in position $\mathsf{Rch}^m{}_{i,j}$ the relation which contains (q_i, q_j)

iff there is a path from location i to j of length $\leq m$ such that q_i is transformed to q_j along that path, and each intermediate q_k (for $i \leq k \leq j$) is permitted to pass through location k on a trajectory from B to E.

$\mathsf{Rch}^0 = K$**,** since paths of length 0 lead only from i to i, and $K_{i,i}$ reflects the packets permitted there.

$\mathsf{Rch}^1 = K + (K\, A^F\, K)$**.** A path of length ≤ 1 is either empty or else it takes one step from i to an adjacent location j.
F determines whether a packet can traverse this edge and how it is rewritten. The multiplication by K on the left restricts the results to the packets that could permissibly be present at i before traversing the edge, while the multiplication by K on the right restricts them to packets permissibly at j after the step.

$\mathsf{Rch}^{(2m)} = (\mathsf{Rch}^m\, \mathsf{Rch}^m)$**,** since the paths of length $\leq 2m$ are just the paths that divide into two paths of length $\leq m$, respectively ending and beginning at the same node k.

$\mathsf{Rch} = \mathsf{Rch}^b$ is the fixed point of Rch^m.

When the only packet transformations are NATs, this recursion will again reach its fixed point Rch after $\mathcal{O}(\log_2 |\mathcal{LO}|)$ matrix multiplications. Although non-simple paths can bring transformed packets back to NATs they have already traversed, NAT outputs are essentially idempotent: They simply select fixed addresses and (for source NATs) new undistinguished ports. Thus, these packets will not lead to a larger set of output, NAT-transformed results. Packet transformations could be conceived that would converge to a fixed point only after large numbers of multiplications, possibly of the order of the number of distinct packets. However, non-simple paths do not delay the fixed point for NATs.

Lemma 34. $\mathsf{KEEP}(i) = \{q\colon \exists p_b, p_E\,.\,(p_B, q) \in \mathsf{Rch}_{B,i} \wedge (q, p_E) \in \mathsf{Rch}_{i,E}\}$.

Security and Success Functionality. We justify security for this filtering posture much as before, but using the equi-goals properties. For given endpoints B, E, the matrix Rch^m summarizes what permissible trajectories of length $\leq m$ can deliver from B and to E. When we define $\mathsf{KEEP}_*(\ell)$ as the union of the sets $\mathsf{KEEP}_{B,E}(\ell)$, we rely on Lemma 32 to justify that this does not erase the differences between KEEP sets for the different endpoints.

As for its maximality, we observe as before that we filter out only packets that have no permissible success trajectories.

Firewall Configuration. As in the plain case, we define the firewall behaviour $\mathcal{FW}$ based on the KEEP_* sets and the function $F(e, d)$. Consider the subgraph $\cdot \xrightarrow{c_i} \ell \xrightarrow{c_o} \ell'$ where ℓ and ℓ' may have NAT behavior defined by the function F used to define A^F. Then $\mathcal{FW}(t, c_i, c_o, p) =$

$$\{q_o \in \mathsf{KEEP}_*(\ell')\colon \begin{array}{l} \exists q_i \in \mathsf{KEEP}_*(\ell)\,. \\ (p, q_i) \in F(c_i, \mathtt{in}) \wedge \\ (q_i, q_o) \in F(c_o, \mathtt{out})\}. \end{array}$$

That is, ℓ may queue q_o for delivery over c_o if there is a q_i into which it can translate p along channel c_i, and q_i is a packet it can keep, and it can translate q_i into q_o outbound on c_o, and moreover ℓ' can keep q_o. The router ℓ must do filtering on both sides, because ℓ' may be a network region or end host, or a router without filtering functionality.

As with Thm. 23 and 20:

Theorem 35. *Let G be a non-empty collection of success goals, and let the frame $\mathcal{F}$ obey $\mathcal{FW}$ and have every channel connected to at least one router. $\mathcal{F}$ enforces G. If $\mathcal{F}'$ (with the same graph and* NAT *behaviour) enforces G, then $\mathcal{F}$ is as successful functionally as $\mathcal{F}'$.* ///

The complexity of generating filters using this method depends on several considerations. One is the complexity of the underlying ring operations set intersection and relative product. Although these are potentially computationally hard, they appear to be easy in practice. In particular, operations relevant to network filtering rarely mix many different bits. For instance, the low order bit of a source address will probably never be combined with the third bit of a destination port.

The complexity also depends on the number of locations in the network. While this would seem large, in fact, we do not need to represent every network device as a separate location. Only the ones that are relevant to the security policy—by requiring some distinctive protection—or to the security processing—by being a location at which we will filter or route—need to be represented. The remaining devices may be coalesced with their neighbors, and represented by a single location. This abstraction leads to fairly small graphs.

If we regard a ring operation as a unit of computational effort, and let k be the number of locations in the abstracted graph, then each matrix multiplication is below $\mathcal{O}(k^3)$, and computing Rch requires $\log_2 k$ multiplications. This must be done k^2 times, performing the computations for $\mathsf{KEEP}_{B,E}(\cdot)$ as B, E vary. Thus, the full computation is $\mathcal{O}(k^5 \log k)$.

7 Conclusion

This paper develops specifications for security goals governing how packets traverse the network, and shows how to distribute filtering functionality to different nodes. We have proved that the resulting filtering implements the security constraints while providing maximal service functionality. We have moreover implemented and validated our results using a tool that automatically localizes the filters and generates configurations for all the firewalls in the network.

As a future work, from a theoretical point of view we plan to give an enforcement of the spoofing goals (that consider only trajectories with spoofing at creation), and promiscuous delivery goals (i.e., trajectories with promiscuous delivery). From a practical point of view we plan to extend Mignis to localize filters also when NATs are present.

Acknowledgements This work was partially supported by FCT projects Confident PTDC/EEI-CTP/4503/2014 and FCT project UID/EEA/50008/2013.

References

1. P. Adao, C. Bozzato, R. Focardi G. Dei Rossi, and F.L. Luccio. Mignis: A semantic based tool for firewall configuration. In *IEEE 27th Computer Security Foundations Symposium (CSF 2014), Vienna, Austria*, pages 351–365. IEEE CS Press, July 2014.
2. E. Al-Shaer, A. El-Atawy, and T. Samak. Automated Pseudo-Live Testing of Firewall Configuration Enforcement. *IEEE Journal on selected areas in communication*, 27(3):302–314, 2009.
3. C. J. Anderson, N. Foster, A. Guha, J.-B. Jeannin, D. Kozen, C. Schlesinger, and D. Walker. NetKAT: Semantic foundations for networks. In *Proc. of the 41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL 2014)*. ACM, 2014.
4. Randal E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Trans. Comput.*, 35(8):677–691, August 1986.
5. Frenetic, a family of network programming languages. `http://www.frenetic-lang.org/`, 2013.
6. J.D. Guttman and A.L. Herzog. Rigorous automated network security management. *International Journal for Information Security*, 5(1–2):29–48, 2005.
7. J.D Guttman and P.D. Rowe. A cut principle for information flow. In *IEEE 28th Computer Security Foundations Symposium (CSF 2015), Verona, Italy*, pages 107–121. IEEE CS Press, July 2015.
8. Ahmed Khurshid, Wenxuan Zhou, Matthew Caesar, and PB Godfrey. Veriflow: verifying network-wide invariants in real time. *ACM SIGCOMM Computer Communication Review*, 42(4):467–472, 2012.
9. T. Nelson, C. Barratt, D.J. Dougherty, K. Fisler, and S. Krishnamurthi. The margrave tool for firewall analysis. In *Proceedings of the 24th International Conference on Large Installation System Administration (LISA'10)*, pages 1–8, Berkeley, CA, USA, 2010. USENIX Association.
10. J. Walsh. Icsa labs firewall testing: An in depth analysis. `http://bandwidthco.com/whitepapers/netforensics/penetration/Firewall\%20Testing.pdf`, 2004.
11. 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 *Proc. of ACM Symposium on Access Control Models and Technologies (SACMAT 2007)*. ACM, 2007.

Appendix

Table II reports the output of the localization tool that automatically generated the `Mignis`$^+$ configuration for the four firewalls.

```
FIREWALL fw1

   Sensitive@if_Sensitive   : 22        >     Trusted@if_fw2                | -m state --state ESTABLISHED,RELATED
   Sensitive@if_Sensitive               >    Internet@if_fw2        : 443

   Sensitive@if_Sensitive               >    Internet@if_fw3        : 443

     Trusted@if_fw2                     >   Sensitive@if_Sensitive   : 22
    Internet@if_fw2         : 443       >   Sensitive@if_Sensitive            | -m state --state ESTABLISHED,RELATED

   Untrusted@if_fw2                     >    Internet@if_fw3        : 80
     Trusted@if_fw2                     >    Internet@if_fw3

    Internet@if_fw3         : 443       >   Sensitive@if_Sensitive            | -m state --state ESTABLISHED,RELATED

    Internet@if_fw3         : 80        >   Untrusted@if_fw2                 | -m state --state ESTABLISHED,RELATED
    Internet@if_fw3                     >     Trusted@if_fw2        : 22
    Internet@if_fw3                     >     Trusted@if_fw2                | -m state --state ESTABLISHED,RELATED


FIREWALL fw2

     Trusted@if_Trusted                 >   Sensitive@if_fw1        : 22
     Trusted@if_Trusted                 >    Internet@if_fw1

     Trusted@if_Trusted                 >    Internet@if_fw4

   Untrusted@if_Untrusted               >    Internet@if_fw1        : 80

   Sensitive@if_fw1         : 22        >     Trusted@if_Trusted            | -m state --state ESTABLISHED,RELATED
    Internet@if_fw1                     >     Trusted@if_Trusted    : 22
    Internet@if_fw1                     >     Trusted@if_Trusted            | -m state --state ESTABLISHED,RELATED

    Internet@if_fw1         : 80        >   Untrusted@if_Untrusted          | -m state --state ESTABLISHED,RELATED

   Sensitive@if_fw1                     >    Internet@if_fw4        : 443

    Internet@if_fw4                     >     Trusted@if_Trusted    : 22
    Internet@if_fw4                     >     Trusted@if_Trusted            | -m state --state ESTABLISHED,RELATED

    Internet@if_fw4         : 443       >   Sensitive@if_fw1                | -m state --state ESTABLISHED,RELATED


FIREWALL fw3

    Internet@if_Internet    : 443       >   Sensitive@if_fw1                | -m state --state ESTABLISHED,RELATED
    Internet@if_Internet    : 80        >   Untrusted@if_fw1                | -m state --state ESTABLISHED,RELATED
    Internet@if_Internet                >     Trusted@if_fw1        : 22
    Internet@if_Internet                >     Trusted@if_fw1                | -m state --state ESTABLISHED,RELATED

   Sensitive@if_fw1                     >    Internet@if_Internet    : 443
   Untrusted@if_fw1                     >    Internet@if_Internet    : 80
     Trusted@if_fw1                     >    Internet@if_Internet


FIREWALL fw4

    Internet@if_Internet    : 443       >   Sensitive@if_fw2                | -m state --state ESTABLISHED,RELATED
    Internet@if_Internet                >     Trusted@if_fw2        : 22
    Internet@if_Internet                >     Trusted@if_fw2                | -m state --state ESTABLISHED,RELATED

   Sensitive@if_fw2                     >    Internet@if_Internet    : 443
     Trusted@if_fw2                     >    Internet@if_Internet
```

Table 3. `Mignis`$^+$ rules automatically generated for the four firewalls of the example network.