Localizing Security for Distributed Firewalls

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

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

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. Our contributions include a way to specify security goals for how packets traverse the network and an algorithm to distribute filtering functionality to different nodes in the network to enforce a given set of security goals.

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 distinct, for instance those that contain the university's human resources servers. These 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 for 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 addresses to port 80 or 443 on a web server.

Our contribution. In this summary of ongoing work, we describe theories and tools that are under development to protect complex networks by enforcing security policies that control the trajectories of packets through them. Our approach is motivated by some of our own previous work. We have previously studied trajectory-based security goals [3], developing techniques to determine whether existing configurations enforce them correctly. We formalize the network graphs and their possible executions in our frame model [4].

We will here describe the outlines of an approach that starts with a network topology together with a set of security goals to enforce. These security goals constrain which packets may follow a trajectory. The analysis uses the topology to determine what enforcement should be applied at which locations in the network. Our methods are designed to apply also in the case of network operation that transform packets as they pass; we have particularly focused on network address translation (NAT).

In future work we will connect the resulting enforcement strategy to our declarative, order-independent language Mignis which can be compiled to generate concrete implementations under Netfilter [1].

2 Model and Security Goals

2.1 System model

We work within our frame model [4]. Suppose given three domains $\mathcal{LO}, \mathcal{CH}, \mathcal{D}$, to which we will refer as the *locations*, *channels*, and *data*, resp. Each channel will be a unidirectional conduit between the locations that are its endpoints. In a networking context, where flows are frequently bidirectional, these channels can be grouped into pairs.

An *event* e occurs on a channel $\mathsf{chan}(e) \in \mathcal{CH}$ and carries some data $\mathsf{msg}(e) \in \mathcal{D}$. Each channel $c \in \mathcal{CH}$ may be connected to two locations $\mathsf{sender}(c)$ and $\mathsf{rcpt}(c) \in \mathcal{LO}$.

A *frame* is a directed graph where the nodes are locations $\ell \in \mathcal{LO}$, and the arcs are channels $c \in \mathcal{CH}$ connecting $\mathsf{sender}(c)$ to $\mathsf{rcpt}(c)$; moreover, each location is equipped with a labeled transition system. In particular $\ell \in \mathcal{LO}$ has a transition $s \overset{e}{\rightsquigarrow}_\ell s'$ only if $\ell = \mathsf{sender}(c)$ or $\ell = \mathsf{rcpt}(c)$. Thus, every location has a set of traces involving transmissions and receptions on the channels connected to it.

An execution $\mathcal{A} = \mathcal{E}, \preceq$ of a frame is a well-founded partially ordered set of events such that, for every $\ell \in \mathcal{LO}$, the set of events occurring on channels connected to the single location ℓ are in fact linearly ordered by $\preceq$, and form a possible trace of ℓ. That is, the events involving that location are a possible trace, ordered by the sequence in which they occur.

In [4] we argue that frames and their executions allow effective reasoning about information flow and limited disclosure in distributed systems.

There are many ways to represent networks by frames. We will generally assume that the locations represent routers, end hosts, or network regions. We will

generally assume a pair of arcs, and (in the example) write a single undirected edge. Usually the state of a router or network region includes a set of received but not yet forwarded packets. A state transition in a router may add a new packet; remove and discard a packet (filtering); remove and transmit a packet on a particular edge; or remove, transform, and transmit a packet as in a NAT translation. For simplicity in our example, we assume that a router does not change its filtering rules. Network regions transmit packets but do not filter or transform them.

A *trajectory* in an execution $\mathcal{A}$ is a $\preceq$-increasing sequence of events of $\mathcal{A}$ that track a single packet as it is transferred from location to location, and possibly transformed by NAT rules.

By a *property ϕ of packets*, we mean a set of packets. Generally, these are characterized by one or more headers of the packets, i.e. as the set of all packets that have specified values for these headers, or unions of such sets.

2.2 Security Goals

We focus on what we will call *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 a network location, and ψ_B, ψ_E, and ϕ refer to sets of packets. These predicates refer to the header fields of the packet at that step in the trajectory, which may vary from its header fields at other steps, in cases such as NAT:

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 the packet satisfies ψ_B, ψ_E as it exists 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.

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.

Most typical firewall rules are formalized in our framework as two-region rules.

Also of interest are *traversal control statements* $\psi_B @ B \twoheadrightarrow \phi @ R \twoheadrightarrow \psi_E @ E$. The traversal control statement says that every trajectory from B to E must traverse R and packets must satisfy ϕ while at R; its applicability is restricted to packets that satisfy ψ_B and ψ_E at the beginning and end, resp.

Traversal control $\psi_B @ B \twoheadrightarrow \phi @ R \twoheadrightarrow \psi_E @ E$**:** 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 and packets satisfy ϕ while at R.

For instance, 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 graph, 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 enforce a traversal control statement by stipulating the region control statements that for trajectories from B to E: if the packets traverse R they satisfy ϕ; if the packets traverse any member of $C \setminus \{R\}$, then they satisfy the always-false header property **false**. That is, we have the following family of statements:

$$\begin{aligned} &\psi_B @ B \rightarrow \phi @ R \rightarrow \psi_E @ E \\ &\psi_B @ B \rightarrow \mathbf{false} @ R' \rightarrow \psi_E @ E \qquad \forall R' \in C \setminus \{R\} \end{aligned}$$

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.

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*. 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 regard one system as *at least as successful functionally* as another system over the same network graph iff, for every successful trajectory permitted by the latter, the same trajectory is permitted by the former.

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

3 Localizing filtering to enforce goals

Suppose that we are given a set of goal formulas, each a region control statement $\psi_B@B \rightarrow \phi_0@R \rightarrow \psi_E@E$. We assume a bit of bookkeeping for the forms of each statement. Namely, we assume that each statement concerns either only successful trajectories or else only unsuccessful trajectories. The goal concerns only successful trajectories if $\psi_B \Rightarrow \mathtt{sa}(p) \in IP(B)$ and $\psi_E \Rightarrow \mathtt{da}(p) \in IP(E)$, meaning that the source address of any relevant packet at the start is one of the IP addresses of its actual starting point B, and its destination address at the end is one of the IP addresses of its actual endpoint E. That is, it is not created with a spoofed source address, and it is not consumed by a promiscuous interface to which it is not addressed. By replicating rules, we can rewrite them in a form such that any one rule applies only to non-spoofed, non-promiscuous packets, or alternatively only to packets that are either spoofed or promiscuously delivered. We call these *success rules* and *promiscuity rules* resp.

We first assume that the network involves no NAT rules.

We call this process *localizing* the rules, because we determine which locations to use to enforce those rules.

Localizing success rules without NATs. When the network uses no NAT rules, packets remain the same throughout their trajectories. Thus, in any region R, all of the packets that may traverse R as part of a successful trajectory from B_1 to E_1 will have source address in B_1 and destination address in E_1. They can never be confused with any packets that are in a successful trajectory from B_2 to E_2 when $B_1 \neq B_2$ or $E_1 \neq E_2$. This observation allows us to compute which packets are useful to keep (for success trajectories) on R separately for each pair of endpoints B, E; the packets for any other pair are distinguished from them by addresses. Thus, we will do separate computations and then take unions later.

The key intuition is that, for endpoints B, E, we would like to keep those packets at R which:

- are permitted by all the B, E goals to be at R;
- have a path from B to R touching only permissible regions; and
- have a path from R to E touching only permissible regions.

This is the *keep* set for R, given B, E. We can compute *keep* sets using a matrix computation in the ring of sets of packets, where the ring addition is set union and the ring multiplication is set intersection. This computation is tractable

when the sets are represented via Binary Decision Diagrams, as we found previously [3].

The computation reaches a fixed point because a non-simple path never allows more flow than the simple subpath it contains. Each node may decrease the set of surviving packets by filtering, but cannot increase it.

We must consider all B, E pairs, where a pair that lacks goals is understood as permitting any packets at intermediate nodes. When we have completed the pairs, we have computed all of the packets that should be *kept* in each region R, because those packets have a route traversing R from their unspoofed source address to their intended destination. We call this value $\mathsf{KEEP}(R)$. An appropriate filtering rule for an interface $R' \to R$ to R from an adjacent R' may discard any packets not in $\mathsf{KEEP}(R)$. For instance, assuming that the packets in R' will be either generated there or else in its $\mathsf{KEEP}(\cdot)$ set, we may filter $(\mathsf{gen}(R') \cup \mathsf{KEEP}(R')) \setminus \mathsf{KEEP}(R)$ on the interface $R' \to R$.

This computation is optimal in functionality, because it allows all success trajectories that are compatible with the chosen security goals.

Localizing success rules with NATs. Curiously, the computation for the case where the network has NAT rules is very similar, but is performed in a different ring. Namely, we are no longer interested in a set of packets, but in a relation between packets in their state at a previous location and packets in their state at a later location. Thus, the matrices A will contain, in entry $A_{i,j}$, a relation between packets that may have occurred at location i and their resulting state when reaching location j. The addition in this ring is union. The multiplication is relational product. That is, the "product" of the binary relations $S(x, y)$ and $S'(y, z)$ is the binary relation $\exists y \,.\, S(x, y) \wedge S'(y, z)$. Thus, we are no longer working in a commutative ring. However, we have simply lifted the set-based computation to a relation-based computation.

Binary Decision Diagrams may still be used, although the relative product computation for $\exists y \,.\, S(x, y) \wedge S'(y, z)$ requires taking cases on the boolean variables that contributing to the projected variable y. We do not yet have an estimate of the cost of this computation.

There is an assumption to be made here, to ensure the analogue to the independence of the packets for different beginnings and endpoints. When a number of beginnings are behind the same source NAT, the security goals for regions R beyond the NAT should treat them the same way. Nothing else can be enforced, since they will be indistinguishable beyond the NAT. Destinations behind the same destination NAT must be treated uniformly for a similar reason.

A second contrast with the no-NAT case concerns the simple path assumption. NATs can be arranged so that a non-simple path would produce new packets, although this is contrary to the main purposes for which NATs are used. Thus, we terminate when the computation reaches its fixed point. Some modifications would not reach a fixed point in limited time. For instance, suppose a router applied a bizarre form of NAT in which it increments the source address of the packet. If the network might cause that packet to traverse the router repeatedly, then its source address will be incremented repeatedly. However, for reasonable

transformations, although the fixed point is not guaranteed to happen quickly, it is extremely likely in fact to do so.

Simple paths. If one desires, one can adapt the above computations to be based only on simple paths. A classical technique [7], adapted to our context, is to tag sets of packets with the set of locations that have figured in the relevant paths. When combining paths say $B \rightarrow^+ R$ and $R \rightarrow^+ E$, one checks that R is the only location they have both visited. This further complicates the data structure and requires the computation to handle more cases separately, namely all those where paths traverse different set of locations. However, this tagging scheme actually lifts a ring to a more complex ring, thus allowing the same structure for the core computations.

Localizing promiscuity rules. Once we have computed the filtering for the success rules, the promiscuity rules may already be satisfied. This will happen whenever the intermediate locations needed for success trajectories are disjoint from the suspicious locations that the promiscuity rules are protecting against.

Otherwise, there is genuine conflict between functionality for permitted success trajectories and security concerns for messages that might be spoofed in locations they traverse. There is not always a canonical, best-functionality solution to this problem. However, we can always enforce a set of promiscuity rules by using another matrix computation. We find the set of promiscuous trajectories that are permitted by the existing, success-based filtering rules. For all packets that can traverse a trajectory, we then update the filters on a link of the trajectory to discard these packets. An attractive heuristic is that for anti-spoofing goals, we should discard the packets as early as possible: We do not want them to get anywhere. However, for anti-promiscuous delivery goals, we should discard them as late as possible. These packets, if safely routed, will be useful.

Simplifying the network topology. The computations we are describing are tractable, because in fact we can simplfy the topology of networks. In particular, given a large network with a set of filtering locations identified, we can shrink the network by identifying devices that are in similar positions relative to the filtering locations. In particular, two devices should be mapped to the same region when they have simple paths to exactly the same set of filtering locations, where these paths traverse no intermediate filtering locations.

Using this idea, complicated networks actually furnish small graphs for our algorithms to work with.

4 Case study

We consider the case study depicted in Figure 1 composed of three subnetworks: `Sensitive`, `Trusted` and `Untrusted`. `Sensitive` subnetwork contains important servers and data and is connected to the `Internet` through the firewall router `fw1` and then gateway `gw1`; `Trusted` subnetwork is composed of trusted

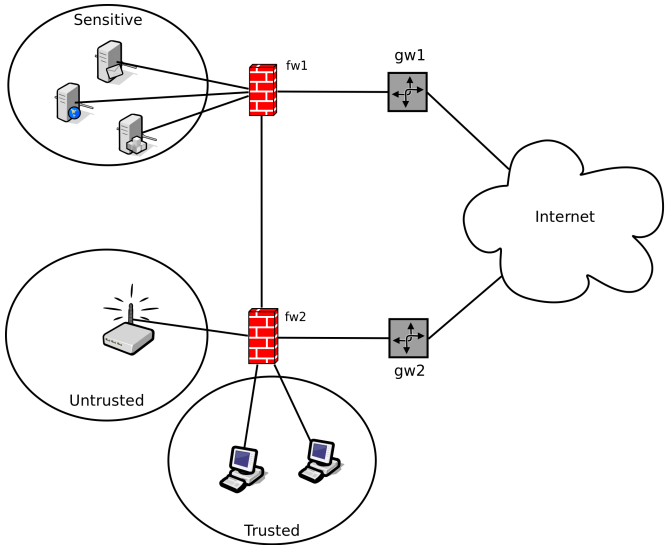

Fig. 1. A simple network with two firewall routers `fw1` and `fw2`.

hosts that, for example, can access services hosted in the `Sensitive` subnetwork; `Untrusted` is a wifi subnetwork providing a 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 firewall router `fw1`, and to the `Internet` through gateway `gw2`.

We now define security goals for the example network.

Success rules. The following rules apply only to non-spoofed, non-promiscuous packets. This can be easily enforced by assuming ψ_B and ψ_E respectively check if $\texttt{sa}(p) \in IP(B)$ and $\texttt{da}(p) \in IP(E)$. For the sake of readability we omit to write these checks in the rules and and we leave them implicit.

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 new connection: it is enough to open the firewall in one direction and let *established* packets come back. In the following we write **est** to note that a packet is established. While specifying rules, we proceed pair by pair so to define rules and their established counterpart (when needed) at the same time.

Hosts in the `Sensitive` and `Trusted` subnetworks should 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 2.2):

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

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. This is achieved by adding a precondition on the start point in the second rule below:

$$\begin{aligned}
\texttt{Trusted} \rightarrow \mathbf{dport} = \mathbf{22}@\texttt{Sensitive} &\rightarrow \texttt{Sensitive} \\
\neg(\mathbf{sport} = \mathbf{22} \wedge \mathbf{est})@\texttt{Sensitive} \rightarrow \quad \mathbf{false}@\texttt{Sensitive} &\rightarrow \texttt{Trusted} \\
\texttt{Trusted} \rightarrow \quad \mathbf{false}@\texttt{gw1}, \texttt{gw2} &\rightarrow \texttt{Sensitive} \\
\texttt{Sensitive} \rightarrow \quad \mathbf{false}@\texttt{gw1}, \texttt{gw2} &\rightarrow \texttt{Trusted}
\end{aligned}$$

`Sensitive` should access the `Internet` only via https (destination port should be 443), while `Internet` hosts should never connect to `Sensitive`:

$$\begin{aligned}
\texttt{Sensitive} \rightarrow \mathbf{dport} = \mathbf{443}@\texttt{Sensitive} &\rightarrow \texttt{Internet} \\
\neg(\mathbf{sport} = \mathbf{443} \wedge \mathbf{est})@\texttt{Internet} \rightarrow \quad \mathbf{false}@\texttt{Sensitive} &\rightarrow \texttt{Sensitive}
\end{aligned}$$

`Trusted` has full access to the `Internet` and from the `Internet` we give access to `Trusted` only via ssh (port 22):

$$\neg\mathbf{est}@\texttt{Internet} \rightarrow \mathbf{dport} = \mathbf{22}@\texttt{Trusted} \rightarrow \texttt{Trusted}$$

`Untrusted` should access the `Internet` exclusively through `gw1` under filter ϕ. This is a form of traversal control that can be compiled into region control rules by taking cut $\{\texttt{gw1}, \texttt{gw2}\}$ and forbidding traversal of everything but `gw1`, i.e., `gw2` (cf. Section 2.2). In a real setting, this might be motivated by the fact `fw1` is more powerful than `fw2` being able to handle complex (stateful) protocols covered by ϕ and offering logging capabilities that are useful to check what the untrusted users do. `Internet` should never access `Untrusted`. We let $\bar{\phi}$ denote the counterpart of ϕ holding on established packets (e.g., swapping source and destination ports):

$$\begin{aligned}
\texttt{Untrusted} \rightarrow \quad \phi@\texttt{gw1} \quad &\rightarrow \texttt{Internet} \\
\neg(\bar{\phi} \wedge \mathbf{est})@\texttt{Internet} \rightarrow \mathbf{false}@\texttt{gw1} &\rightarrow \texttt{Untrusted} \\
\texttt{Untrusted} \rightarrow \mathbf{false}@\texttt{gw2} &\rightarrow \texttt{Internet} \\
\texttt{Internet} \rightarrow \mathbf{false}@\texttt{gw2} &\rightarrow \texttt{Untrusted}
\end{aligned}$$

Promiscuity rules. We assume that subnetworks `Sensitive` and `Trusted` do not spoof or promiscuously deliver packets. Let $\texttt{N} \in \{\texttt{Sensitive}, \texttt{Trusted}, \texttt{Internet}\}$ and $\texttt{N}' \in \{\texttt{Sensitive}, \texttt{Trusted}, \texttt{Untrusted}\}$. The following rules prevent spoofing from `Unstrusted` and `Internet`:

$$\begin{aligned}
\texttt{sa} \notin \mathit{IP}(\texttt{Untrusted})@\texttt{Untrusted} &\rightarrow \mathbf{false}@\texttt{Unstrusted} \rightarrow \texttt{N} \\
\texttt{sa} \notin \mathit{IP}(\texttt{Internet})@\texttt{Internet} \rightarrow \quad \mathbf{false}@\texttt{Internet} \quad &\rightarrow \texttt{N}'
\end{aligned}$$

while the following ones prevent promiscuous deliver to `Unstrusted` and `Internet`:

$$\begin{aligned}
\texttt{N} \rightarrow \mathbf{false}@\texttt{Untrusted} &\rightarrow \texttt{da} \notin \mathit{IP}(\texttt{Untrusted})@\texttt{Untrusted} \\
\texttt{N}' \rightarrow \quad \mathbf{false}@\texttt{Internet} \quad &\rightarrow \texttt{da} \notin \mathit{IP}(\texttt{Internet})@\texttt{Internet}
\end{aligned}$$

Localizing filtering. We show how some of the above goals are localized. We first consider the case of completely forbidden communication from `Sensitive` to `Untrusted`:

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

By performing a matrix computation we easily obtain that $\mathsf{KEEP}(.) = \emptyset$ for each node in the network. Indeed, the rule forbids any packet from `Sensitive` to `Untrusted` directly at the source. Since we have

$$\texttt{gen}(\texttt{Sensitive}) = \{p \mid \texttt{sa}(p) \in \mathit{IP}(\texttt{Sensitive}), \texttt{da}(p) \in \mathit{IP}(\texttt{Untrusted})\}$$

then we only need to filter $(\texttt{gen}(\texttt{Sensitive}) \cup \mathsf{KEEP}(\texttt{Sensitive})) \setminus \mathsf{KEEP}(\texttt{fw1}) = \texttt{gen}(\texttt{Sensitive})$ on the interface $\texttt{Sensitive} \rightarrow \texttt{fw1}$. All packets from `Sensitive` to `Untrusted` will be dropped as early as possible in `fw1` and no filtering will be done in `fw2`.

We now consider the more interesting situation of traversal control from `Untrusted` to the `Internet` imposing packets to go through `fw1`:

$$\begin{array}{l} \texttt{Untrusted} \rightarrow \quad \phi@\texttt{gw1} \quad \rightarrow \texttt{Internet} \\ \texttt{Untrusted} \rightarrow \textbf{false}@\texttt{gw2} \rightarrow \texttt{Internet} \end{array}$$

In this case we obtain that $\mathsf{KEEP}(.)$ is equal to $\texttt{gen}(\texttt{Untrusted})$ for `Untrusted`, `Sensitive`, `fw1`, `fw2`; it is equal to ϕ for `gw1`, `Internet`; it is the emptyset for `gw2`. We thus need to filter $\texttt{gen}(\texttt{Untrusted})$ on the interface $\texttt{fw2} \rightarrow \texttt{gw2}$ and $\texttt{gen}(\texttt{Untrusted}) \setminus \phi$ on the interface $\texttt{fw1} \rightarrow \texttt{gw1}$. Intuitively, `fw1` will filter all generated packets that do not belong to the set of permitted packets ϕ while `fw2` will filter any (generated) packet, enforcing traversal control.

In a similar way, we can filter spoofing and promiscuous delivery. For example:

$$\texttt{sa} \notin \mathit{IP}(\texttt{Untrusted})@\texttt{Untrusted} \rightarrow \textbf{false}@\texttt{Unstrusted} \rightarrow \texttt{Sensitive}$$

will produce a filter on interface $\texttt{Untrusted} \rightarrow \texttt{fw2}$ dropping any packet originated in `Untrusted` with a spoofed IP (not in $\mathit{IP}(\texttt{Untrusted})$) and directed to `Sensitive`. Notice that filtering happens as soon as possible. Dually rule:

$$\texttt{Sensitive} \rightarrow \textbf{false}@\texttt{Untrusted} \rightarrow \texttt{da} \notin \mathit{IP}(\texttt{Untrusted})@\texttt{Untrusted}$$

will produce a filter on interface $\texttt{fw2} \rightarrow \texttt{Untrusted}$ dropping any packet originated in `Sensitive` and promiscuously delivered to `Untrusted` to an IP not in $\mathit{IP}(\texttt{Untrusted})$. Notice that in this case filtering happens as late as possible, only in case the packet is (wrongly) routed to `Untrusted`.

5 Conclusion

Our localization strategy does not generate concrete rule-sets for actual devices. However, we plan to generate rules in the declarative, order-independent language of Mignis [1]; the semantically motivated Mignis compiler then generates

concrete rule-sets for Netfilter. This will complete the task of generating semantically correct network configurations from security goal statements.

Our work is distinguished by its focus on clear behavioral security specifications. In this it contrasts with otherwise very strong work, for instance on security using programming language techniques as in NetKAT [2]. Zhang et al. [8] focus more on the possible conflicts among policies at different organizational levels, and less on their consequences given the topology of the network. Much work has been devoted to firewall analysis, e.g. Margrave [6], which again lacks the distributed behavior of the network.

Kurshid et al. [5] demonstrate that it is possible, in a software defined networking context, to check dynamically to ensure that 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 more systematic way to solve well-defined security problems at design time.

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 Computer Security Foundations*, pages 351–365. IEEE CS Press, July 2014.
2. 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.
3. J.D. Guttman and A.L. Herzog. Rigorous automated network security management. *International Journal for Information Security*, 5(1–2):29–48, 2005.
4. J.D. Guttman and P.D. Rowe. A cut principle for information flow. In *IEEE Computer Security Foundations*, pages 107–121. IEEE CS Press, July 2015.
5. 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.
6. 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.
7. F. Rubin. Enumerating all simple paths in a graph. *IEEE Trans. Circuits and Systems*, 25(8):641–642, 1978.
8. 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.