From Firewalls to Functions and Back

Lorenzo Ceragioli
Università di Pisa, Pisa, Italy
lorenzo.ceragioli@phd.unipi.it

Letterio Galletta
IMT School for Advanced Studies, Lucca, Italy
letterio.galletta@imtlucca.it

Mauro Tempesta
Università Ca’ Foscari, Venezia, Italy
TU Wien, Vienna, Austria
tempesta@unive.it
What is a Firewall?

Inspects the traffic on a node of the network, for each packet

- accepts or drops it
- possibly changes the addresses (NAT)
What is a Firewall?

- Inspects the traffic on a node of the network, for each packet
  - accepts or drops it
  - possibly changes the addresses (NAT)

Based on a configuration
- List of rules
- Possibly using tags
- Procedure-like constructs
- Interaction among rules (Shadowing)
Motivations

**Firewalls** are a basic tool for protecting network

- **Widespread**
- **Configuration-based**
- **Different** configuration languages (iptables, pf, ipfw)
- It’s **Hard** to configure and manage firewalls
- Cross-platform policy porting is **Harder**
Motivations

**Firewalls** are a basic tool for protecting network

- **Widespread**
- **Configuration-based**
- **Different** configuration languages (iptables, pf, ipfw)
- It’s **Hard** to configure and manage firewalls
- Cross-platform policy porting is **Harder**

Misconfigurations cause unintended behaviour

Possible Threats
Previous works:

Transcompilation Pipeline between firewall languages

- Supports iptables, pf, ipfw and (partially) CISCO-ios
- General approach
- Supports NAT
- Formal semantics
Previous works:

**Transcompilation Pipeline** between firewall languages

- Supports *iptables*, *pf*, *ipfw* and (partially) *CISCO-ios*
- General approach
- Supports NAT
- Formal semantics

Why

- For automated policy **porting** (first general approach!)
- For configuration **refactoring**
- Synthesis of a high level **declarative configuration**
- **Basis** for other policy management tasks
Our Goal

Old Legacy Technology

iptables
ipfw
pf
Our Goal

Old Legacy Technology

High level representation

decomilation

iptables
ipfw
pf

Old Legacy Technology
Our Goal

High level representation

configuration update
configuration testing
configuration generation
policy verification
various representation

Old Legacy Technology

iptables
ipfw
pf
Our Goal

High level representation

Old Legacy Technology

compilation

iptables
ipfw
pf
Each firewall system

- Has its own configuration language
- Makes different evaluation steps to process packets
- Lots of low level details
  - First do the NAT, than filtering or vice-versa?
  - How to express complex conditions (negated)?
Each firewall system

- Has its own configuration language
- Makes different evaluation steps to process packets
- Lots of low level details
  - First do the NAT, then filtering or vice-versa?
  - How to express complex conditions (negated)?

**General Model**

\[
\text{Firewall} = \text{set of rules} + \text{the evaluating procedure}
\]
Firewall = set of rules + the evaluating procedure

Control Diagram

$S$ are the addresses of the firewall
Firewall = set of rules + the evaluating procedure

Configuration

Assigns a rule sets to each node

Ruleset : list of rules $r = (\phi, a)$

- $\phi(p)$ : condition
- $a$ : action
  - ACCEPT
  - DROP
  - $\text{NAT}(d_n, s_n)$
  - $\text{MARK}(m)$
  - $\text{GOTO}(R)$
  - $\text{CALL}(R)$
  - RETURN

Control Diagram

$S$ are the addresses of the firewall
Firewall = set of rules + the evaluating procedure

Configuration

Assigns a rulesets to each node

Ruleset : list of rules \( r = (\phi, a) \)

- \( \phi(p) \) : condition
- \( a \) : action
  - ACCEPT
  - DROP
  - NAT\((d_n, s_n)\)
  - MARK\((m)\)
  - GOTO\((R)\)
  - CALL\((R)\)
  - RETURN

Control Diagram

\( S \) are the addresses of the firewall
Transcompilation Pipeline

From Firewalls to Functions and Back
Previous implementation of the pipeline synthesis:

**Compute the models of a predicate (SAT-solver)**
Black-box approach (no fine tuning)
Previous implementation of the pipeline synthesis:

**Compute the models of a predicate (SAT-solver)**
Black-box approach (no fine tuning)

Change of domain:

**Function-based redefinition of the pipeline**
Previous implementation of the pipeline synthesis:

**Compute the models of a predicate (SAT-solver)**
Black-box approach (no fine tuning)

Change of domain:

**Function-based redefinition of the pipeline**

(Firewalls → Functions) :
source configuration ↦ function representing its **meaning**

(Firewalls ← Functions) :
functional representation ↦ target configuration
Previous implementation of the pipeline synthesis:

- **Compute the models of a predicate (SAT-solver)**
- Black-box approach (no fine tuning)

Change of domain:

**Function-based redefinition of the pipeline**

(Firewalls $\rightarrow$ Functions):
- source configuration $\mapsto$ function representing its **meaning**

(Firewalls $\leftarrow$ Functions):
- functional representation $\mapsto$ target configuration

Functions are an **handy domain**:
- They allow **simple and general solutions**
Rule sets and Firewalls as Functions

$$\tau : P \rightarrow \mathcal{T}(P) \cup \{\bot\}$$

where

- $P$: network packets
- $\mathcal{T}(P)$: transformations possibly applied to packets
- $\bot$: discard of a packet
\( \tau : \mathbb{P} \rightarrow \mathcal{T}(\mathbb{P}) \cup \{\bot\} \) where

- \( \mathbb{P} \) network packets
- \( \mathcal{T}(\mathbb{P}) \) transformations possibly applied to packets
- \( \bot \) discard of a packet

New pipeline stages:
- **ruleset synthesis**: rulesets became functions
- **composition**: computes the semantics of the firewall
- **generation**: assign functions to the target nodes
- **translation**: from IFCL to pf configuration language
Rule sets and Firewalls as Functions

\[ \tau : P \rightarrow T(P) \cup \{\perp\} \]

where

- \(P\) network packets
- \(T(P)\) transformations possibly applied to packets
- \(\perp\) discard of a packet

New pipeline stages:

- \textbf{ruleset synthesis}: rulesets became functions
- \textbf{composition}: computes the semantics of the firewall
- \textbf{generation}: assign functions to the target nodes
- \textbf{translation}: from IFCL to pf configuration language

Why:

- \textbf{Parametric} w.r.t. IFCL specification
- Support \textbf{minimal control diagrams} and \textbf{MARK}
- Translation from IFCL to \textbf{target language} is trivial
Function Representation

Functions $\tau : \mathcal{P} \rightarrow \mathcal{T}(\mathcal{P}) \cup \{\bot\}$ as **sets of pairs** $(P, t)$

- $t$ is a transformation
- $P$ is a multi-cube of packets
Function Representation

Functions $\tau : \mathbb{P} \rightarrow \mathcal{T}(\mathbb{P}) \cup \{\bot\}$ as sets of pairs $(P, t)$

- $t$ is a transformation
- $P$ is a multi-cube of packets

Cube:
- Cartesian product of one segment for each dimension

Multi-cube:
- Cartesian product of one union of segments for each dimension
Function Representation

Functions $\tau : \mathbb{P} \rightarrow \mathcal{T}(\mathbb{P}) \cup \{\perp\}$ as sets of pairs $(P, t)$

- $t$ is a transformation
- $P$ is a multi-cube of packets

Cube:
Cartesian product of one segment for each dimension

Multi-cube:
Cartesian product of one union of segments for each dimension

- **succinct** representation
- sets of packets verifying rule conditions
- sets of packets verifying arc conditions
- closed under transformations
Synthesis
From a ruleset to a set of pairs \((P, t)\)
Ruleset Synthesis

From a **ruleset** to a **set of pairs** \((P, t)\)

We scan the ruleset rule-by-rule, keeping track of

- \(P\) packets not managed
- \(t\) transformation assigned to \(P\)
Ruleset Synthesis

From a **ruleset** to a **set of pairs** \((P, t)\)

We scan the ruleset rule-by-rule, keeping track of

- \(P\) packets not managed
- \(t\) transformation assigned to \(P\)

**Base Case:** if \(R = [\ ]\) just returns \(\{(P, t)\}\)
Ruleset Synthesis

From a ruleset to a set of pairs \((P, t)\)

We scan the ruleset rule-by-rule, keeping track of

- \(P\) packets not managed
- \(t\) transformation assigned to \(P\)

**Base Case:** if \(R = []\) just returns \(\{(P, t)\}\)

**Else:** take the first rule \((\phi, action)\)

\[
P = \begin{cases} 
  P_s & \text{packets that verifies } \phi \\
  P_n & \text{packets that do not \& managed by the other rules}
\end{cases}
\]

if \(action\) terminates the packet processing then \((P_s, t')\)
else \(P_s\) also managed by the other rules (updated transformation \(t'\))
Composition

$q \xrightarrow{\Psi} q'$

Ideally, for each $p \in P$ compute $t$ in the first node compute $p'$:

- How $p$ is when exits node $q$
- Check $\psi(p')$...
  - If $t_i$ does then compute $t'$ in the second node

Overall:

$p \xrightarrow{} t$ updated by $t'$

Composition Algorithm:

The same, but with Multi-cubes ...

(Plus details)
**Composition**

Ideally, for each $p \in \mathbb{P}$
Ideally, for each $p \in \mathbb{P}$

- compute $t$ in the first node
Ideally, for each $p \in \mathbb{P}$

- compute $t$ in the first node
- compute $p'$:
  (how $p$ is when exits node $q$)
Ideally, for each \( p \in P \):

- compute \( t \) in the first node
- compute \( p' \):
  (how \( p \) is when exits node \( q \))
- check \( \psi(p') \)
**Composition**

Ideally, for each \( p \in P \)
- compute \( t \) in the first node
- compute \( p' \): (how \( p \) is when exits node \( q \))
- check \( \psi(p') \)... if \( t_i \) does then

\[ \psi(p') \]
Composition

Ideally, for each $p \in \mathbb{P}$

- compute $t$ in the first node
- compute $p'$:
  (how $p$ is when exits node $q$)
- check $\psi(p')$... if $t_i$ does then
  - compute $t'$ in the second node
Composition

Ideally, for each $p \in \mathbb{P}$

- compute $t$ in the first node
- compute $p'$:
  (how $p$ is when exits node $q$)
- check $\psi(p')$... if $t_i$ does then
  - compute $t'$ in the second node
  - **Overall**: $p \mapsto t$ updated by $t'$
Ideally, for each \( p \in \Pi \)

- compute \( t \) in the first node
- compute \( p' \): (how \( p \) is when exits node \( q \))
- check \( \psi(p') \)... if \( t_i \) does then
  - compute \( t' \) in the second node
  - Overall: \( p \mapsto t \) updated by \( t' \)

Composition Algorithm:

The same, but with Multi-cubes ...

(... plus details)
Example from `ipfw` to `pf`: formalization

```bash
ipfw -q nat 1 config ip 151.15.185.183
ipfw -q nat 2 config redirect_port tcp 9.9.8.8:17 17
ipfw -q add 0010 nat 1 tcp from 192.168.0.0/24 to not 192.168.0.0/24
ipfw -q add 0020 nat 2 tcp from 151.15.185.183 to not 192.168.0.0/24 17
ipfw -q add 0030 allow tcp from 151.15.185.183 to not 192.168.0.0/24 out
ipfw -q add 0040 deny all from any to any
```
Example from **ipfw to pf: formalization**

```plaintext
ipfw -q nat 1 config ip 151.15.185.183
ipfw -q nat 2 config redirect port tcp 9.9.8.8:17 17
ipfw -q add 0010 nat 1 tcp from 192.168.0.0/24 to not 192.168.0.0/24
ipfw -q add 0020 nat 2 tcp from 151.15.185.183 to not 192.168.0.0/24 17
ipfw -q add 0030 allow tcp from 151.15.185.183 to not 192.168.0.0/24 out
ipfw -q add 0040 deny all from any to any
```

$R_0: (sIP \in 192.168.0.0/24 \land dIP \notin 192.168.0.0/24,$
\[ \land \text{NAT}(\star : \star, 151.15.15.183 : \star)); \]
\((sIP = 151.15.185.183 \land dIP \notin 192.168.0.0/24 \land dPort = 17,$
\[ \land \text{NAT}(9.9.8.8 : \star, \star : \star)); \]
\((true, \text{ DROP}) \]

$R_1: \ldots$
Example from *ipfw* to *pf*: ruleset synthesis

\[
R_0 : (sIP \in 192.168.0.0/24 \land dIP \notin 192.168.0.0/24, \text{NAT}(\star : \star, 151.15.15.183 : \star));
\]
\[
(sIP = 151.15.185.183 \land dIP \notin 192.168.0.0/24 \land \text{dPort} = 17, \text{NAT}(9.9.8.8 : \star, \star : \star));
\]
\[
(true, \text{DROP})
\]
Example from ipfw to pf: ruleset synthesis

\[ R_0 : (sIP \in 192.168.0.0/24 \land dIP \notin 192.168.0.0/24, \text{NAT}(* : *, 151.15.15.183 : *)); \\
(sIP = 151.15.185.183 \land dIP \notin 192.168.0.0/24 \land dPort = 17, \text{NAT}(9.9.8.8 : *, * : *)); \\
(true, \text{DROP}) \]
Example from ipfw to pf: composition

Given a transition function $\tau$ from states $q_0$ to $q_f$, we can represent the composition as follows:

$\tau_0$

<table>
<thead>
<tr>
<th>Received packets</th>
<th>Accepted packets</th>
</tr>
</thead>
<tbody>
<tr>
<td>source</td>
<td>destination</td>
</tr>
<tr>
<td>192.168.0.0/24</td>
<td>*</td>
</tr>
<tr>
<td>151.15.185.183</td>
<td>*</td>
</tr>
</tbody>
</table>

$\tau_1$

<table>
<thead>
<tr>
<th>Received packets</th>
<th>Accepted packets</th>
</tr>
</thead>
<tbody>
<tr>
<td>source</td>
<td>destination</td>
</tr>
<tr>
<td>192.168.0.0/24</td>
<td>*</td>
</tr>
<tr>
<td>151.15.185.183</td>
<td>*</td>
</tr>
<tr>
<td>151.15.185.183</td>
<td>*</td>
</tr>
</tbody>
</table>
Example from **ipfw** to **pf**: composition

![Diagram showing state transitions](image)

### $\tau_0$

<table>
<thead>
<tr>
<th>Received packets</th>
<th>Accepted packets</th>
</tr>
</thead>
<tbody>
<tr>
<td>source</td>
<td>destination</td>
</tr>
<tr>
<td>192.168.0.0/24</td>
<td>*</td>
</tr>
<tr>
<td>151.15.185.183</td>
<td>*</td>
</tr>
</tbody>
</table>

### $\tau_1$

<table>
<thead>
<tr>
<th>Received packets</th>
<th>Accepted packets</th>
</tr>
</thead>
<tbody>
<tr>
<td>source</td>
<td>destination</td>
</tr>
<tr>
<td>192.168.0.0/24</td>
<td>*</td>
</tr>
<tr>
<td>151.15.185.183</td>
<td>*</td>
</tr>
<tr>
<td>151.15.185.183</td>
<td>*</td>
</tr>
</tbody>
</table>

---

### Received packets

<table>
<thead>
<tr>
<th>source</th>
<th>destination</th>
<th>Accepted packets</th>
</tr>
</thead>
<tbody>
<tr>
<td>151.15.185.183</td>
<td>*</td>
<td>* { 151.15.185.183 192.168.0.0/24 }</td>
</tr>
<tr>
<td>192.168.0.0/24 {192.168.0.1}</td>
<td>*</td>
<td>127.0.0.1 151.15.185.183</td>
</tr>
<tr>
<td>192.168.0.0/24 {192.168.0.1}</td>
<td>*</td>
<td>* { 127.0.0.1 151.15.185.183 192.168.0.0/24 }</td>
</tr>
<tr>
<td>192.168.0.0/24 {192.168.0.1}</td>
<td>*</td>
<td>* { 127.0.0.1 151.15.185.183 192.168.0.0/24 }</td>
</tr>
<tr>
<td>192.168.0.1</td>
<td>*</td>
<td>* { 127.0.0.1 151.15.185.183 192.168.0.0/24 }</td>
</tr>
<tr>
<td>151.15.185.183</td>
<td>*</td>
<td>* { 192.168.0.0/24 }</td>
</tr>
</tbody>
</table>
Generation

\[ \tau \]

\[ \tau_0, \tau_1, \tau_2, \tau_3 \]

\[ \ldots \]
Problem: not every ruleset can be assigned to each node!

- **To guarantee the final translation**
  - Simple targets: ACCEPT, DROP and NAT
  - Assign **Labels** to nodes:
    - DROP
    - SNAT
    - DNAT

- **Different expressive power**
How to generate functions

Problem: not every ruleset can be assigned to each node!

- To guarantee the final translation
  - Simple targets: ACCEPT, DROP and NAT
  - Assign Labels to nodes:
    - DROP
    - SNAT
    - DNAT

- Different expressive power

```
q_0 \rightarrow q_1
\{DNAT\} \rightarrow \{DROP\}
sIP \notin S \quad dIP \notin S
q_i
sIP \in S \quad dIP \in S
{SNAT} \rightarrow {DROP}
q_2 \rightarrow q_3
q_f
```

Algorithm
For each pair \((P, t)\) with \(t \neq \perp\)
Find the path
For each node \(q\) preceding nodes \(\rightarrow P\)
Labels in \(q\) \(\rightarrow t\)
Special management for DROP pairs \((P, \perp)\)
For each node: packets still not managed
Drop as many of these as possible
How to generate functions

Problem: not every ruleset can be assigned to each node!

- To guarantee the final translation
  - Simple targets: ACCEPT, DROP and NAT
  - Assign **Labels** to nodes:
    - DROP
    - SNAT
    - DNAT
- Different expressive power

Algorithm

- **For each** pair \((P, t)\) with \(t \neq \bot\)
  - Find the **path**
  - **For each** node \(q\)
    - Preceding nodes \(\rightarrow P_q\)
    - Labels in \(q \rightarrow t_q\)
- Special management for DROP pairs \((P, \bot)\)
  - For each node: packets **still not managed**
  - Drop as many of these as possible
Conclusion

The presented transcompilation approach

- Is **parametric** w.r.t. the IFCL specification
- Supports the use of **tags**
- Supports firewalls with **minimal control diagram**
- Preserves the **NAT**
- Reveals **different expressive power** of firewall languages
Conclusion

The presented transcompilation approach

- Is **parametric** w.r.t. the IFCL specification
- Supports the use of **tags**
- Supports firewalls with **minimal control diagram**
- Preserves the **NAT**
- Reveals **different expressive power** of firewall languages

Ongoing and Future Works

- **Coding** and **Testing**
- Non-trivial **multi-cube merging** procedure
- Support for **holistic network management**
- **High-level** tools for network management, **compatible with old technology**