High Level Management of Firewall Configurations

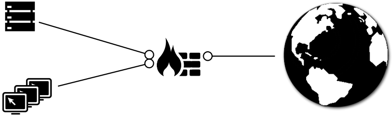
Mauriana Pesaresi Seminar

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

Presentation Outline

- **Introduction and Motivation**
  - What is a **Firewall**
  - Their **configuration** are **difficult** to manage
- **Transcompilation Pipeline**
  - A language-based **Solution**
  - FireWall Synthesizer (**FWS**)
- **Function-Based Redefinition** (Master Thesis)
  - **from Firewalls to Functions and Back**
  - **Composition**
  - **Function Representation**
- **Ongoing and Future Work**
  - **Tag System**
  - **Networks of Firewalls**

What is a Firewall?



Inspects the traffic: for each packet

- **accepts or drops** it
- possibly **modifying** it (NAT)

Based on a **configuration**

- **List of rules**
- Possibly using **tags**
- **Procedure**-like constructs
- **Complex Interaction** among rules (Shadowing)
- **Different** configuration languages
- **Low level** details

Difficult and **error** prone:

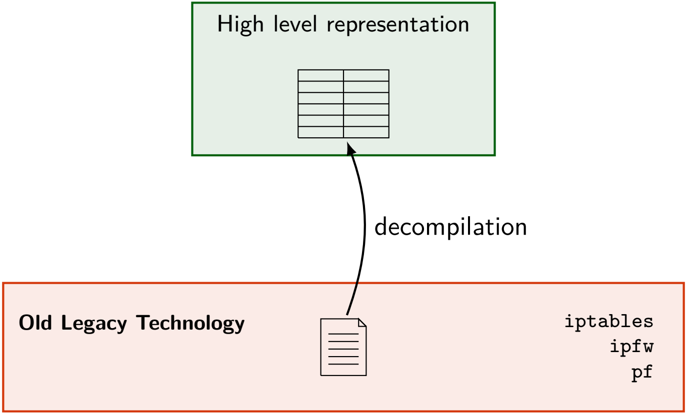
- Configuration
- Cross-system Porting
- Test
- Verification

Our Goal

Old Legacy Technology

`iptables`
`ipfw`
`pf`

Our Goal


High level representation
decompilation
Old Legacy Technology
iptables
ipfw
pf

Our Goal

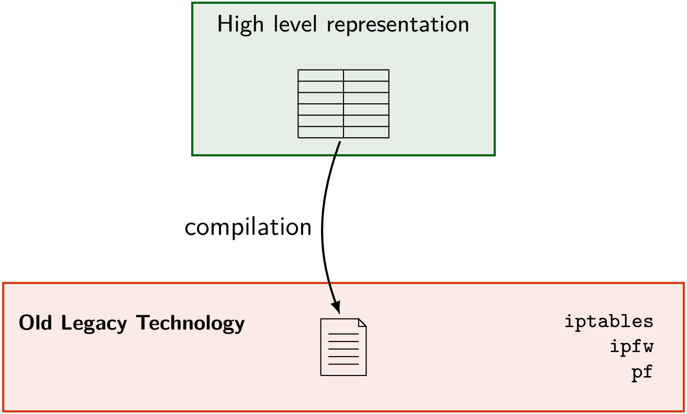
High level representation

update
test
generate
verify
inspect

Old Legacy Technology

`iptables`
`ipfw`
`pf`

Our Goal


High level representation
compilation
Old Legacy Technology
iptables
ipfw
pf

Transcompiltation Pipeline

Transcompilation Pipeline between firewall languages

- Supports `iptables`, `pf`, `ipfw` and (partially) `CISCO`-*ios*
- General approach
- Supports NAT
- Formal semantics
- tool: **FireWall Synthesizer**

IFCL — Intermediate Firewall Configuration Language

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)?

IFCL — Intermediate Firewall Configuration Language

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)?

↓ General Model

Firewall = set of rules + the evaluating procedure

IFCL — Intermediate Firewall Configuration Language

Firewall = set of rules + the evaluating procedure

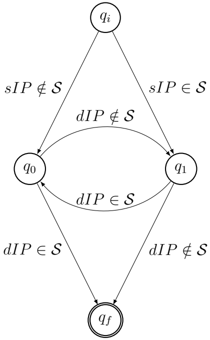
Configuration

Assigns a rulesets to each node

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

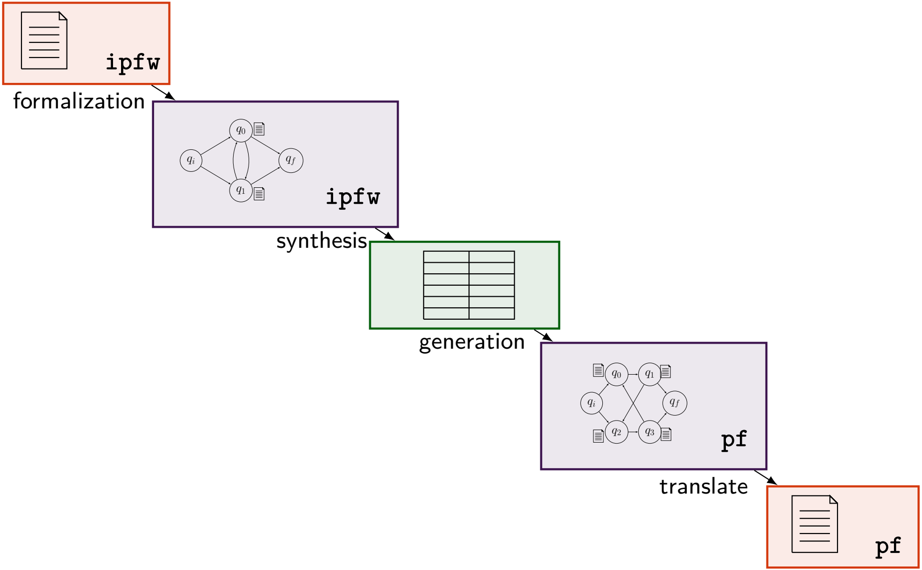
- $\phi(p)$: **condition**
- a : **action**
  - `ACCEPT`
  - `DROP`
  - `NAT`(d_n, s_n)
  - `MARK(m)`
  - ~~`GOTO`(R)~~
  - ~~`CALL`(R)~~
  - ~~`RETURN`~~

Control Diagram



$\mathcal{S}$ are the addresses of the firewall

Transcompilation Pipeline



Rulesets and Firewalls as Functions

$$\tau : \mathbb{P} \to \mathcal{T}(\mathbb{P}) \cup \{\bot\}$$ where

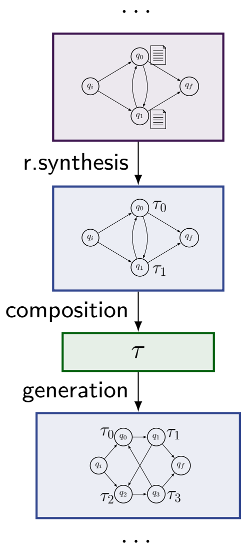
- $\mathbb{P}$ network packets
- $\mathcal{T}(\mathbb{P})$ transformations possibly applied to packets
- $\bot$ discard of a packet

New pipeline stages:

- **ruleset synthesis:** rulesets became functions
- **composition:** computes the semantics of the firewall
- **generation:** assign functions to the target nodes

Why:

- **Parametric** w.r.t. IFCL specification
- Support **minimal control diagrams** and `MARK`
- Translation from IFCL **to target language** is trivial

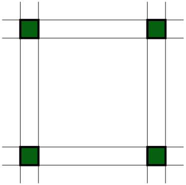


Function Representation

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

t is a transformation

P is a multi-cube of packets

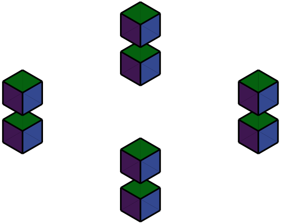


Cube :

Cartesian product of one segment for each dimension

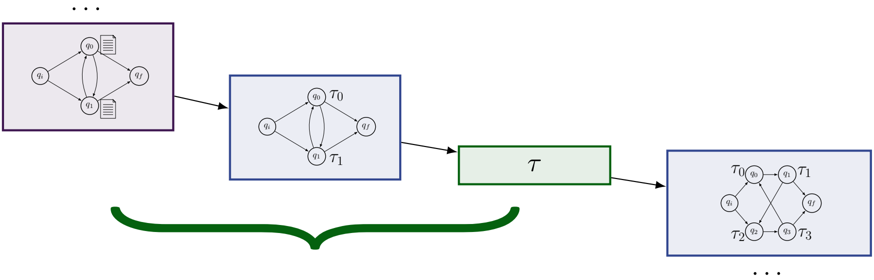
Multi-cube :

Cartesian product of one **union of segments** for each dimension



- **succinct** representation
- sets of packets verifying **rule conditions**
- sets of packets verifying **arc conditions**
- closed under **transformations**

Synthesis



q_0
q_i
q_f
q_1
$\mathcal{T}_0$
$\mathcal{T}_1$
$\mathcal{T}$
$\mathcal{T}_2$
q_2
q_3
$\mathcal{T}_3$

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

Take rule $(\phi, action)$

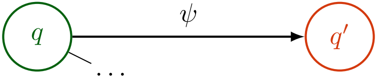
$$P = \begin{cases} P_s & \text{packets that verifies } \phi \\ P_n & \text{packets that do not – managed by the } \textbf{other rules} \end{cases}$$

if $action$ terminates the packet processing **then** (P_s, t')

else P_s also managed by the other rules (updated transformation t')

Composition

τ	
P_1	t_1
P_2	t_2
P_3	t_3



Globally $p \mapsto t$ updated with t'

τ'	
P'_1	t'_1
P'_2	t'_2
P'_3	t'_3

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 ti does then
  - compute t' in the second node
  - **Overall**: $p \mapsto t$ updated by t'

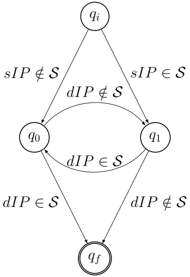
Composition Algorithm:

The same,
but with Multi-cubes ...

(... plus details)

Example from ipfw to pf: formalization

```
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
```



$$
\begin{aligned}
R_0 : &(\texttt{sIP} \in 192.168.0.0/24 \wedge \texttt{dIP} \notin 192.168.0.0/24,\\
&\qquad \texttt{NAT}(\star:\star, 151.15.15.183:\star));\\
&(\texttt{sIP} = 151.15.185.183 \wedge \texttt{dIP} \notin 192.168.0.0/24 \wedge \texttt{dPort} = 17,\\
&\qquad \texttt{NAT}(9.9.8.8:\star, \star:\star));\\
&(true,\ \texttt{DROP})
\end{aligned}
$$

$$R_1 : \ \ldots$$

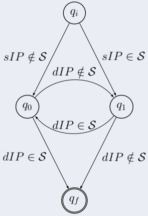
Example from `ipfw` to `pf`: ruleset synthesis

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

τ_0

Received packets				Accepted packets			
source		destination		source		destination	
192.168.0.0/24	*	* \{ 192.168.0.0/24 }	*	151.15.185.183	-	-	-
151.15.185.183	*	* \{ 192.168.0.0/24 }	17	-	-	9.9.8.8	-

Example from `ipfw` to `pf`: composition



τ_0

Received packets				Accepted packets			
source		destination		source		destination	
192.168.0.0/24	*	* \{ 192.168.0.0/24 }	*	151.15.185.183	-	-	-
151.15.185.183	*	* \{ 192.168.0.0/24 }	17	-	-	9.9.8.8	-

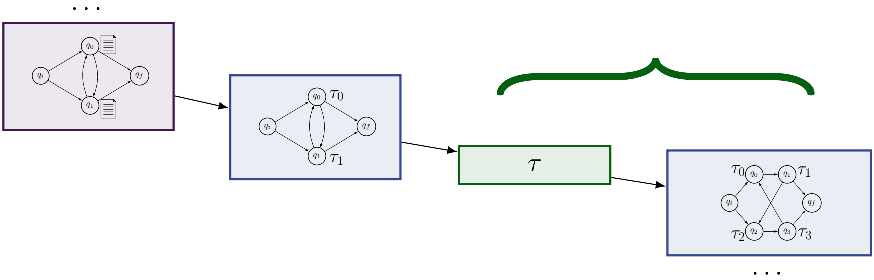
τ_1

Received packets				Accepted packets			
source		destination		source		destination	
192.168.0.0/24	*	* \{ 192.168.0.0/24 }	*	151.15.185.183	-	-	-
151.15.185.183	*	* \{ 192.168.0.0/24 }	17	-	-	9.9.8.8	-
151.15.185.183	*	* \{ 192.168.0.0/24 }	* \{ 17 }	-	-	9.9.8.8	-

⬇

Received packets				Accepted packets			
source		destination		source		destination	
151.15.185.183	*	* \{ 151.15.185.183 192.168.0.0/24 }	* \{17}	-	-	-	-
192.168.0.0/24 \{192.168.0.1}	*	127.0.0.1 151.15.185.183	*	151.15.185.183	-	-	-
192.168.0.0/24 \{192.168.0.1}	*	* \{ 127.0.0.1 151.15.185.183 192.168.0.0/24 }	* \{17}	151.15.185.183	-	-	-
192.168.0.0/24 \{192.168.0.1}	*	* \{ 127.0.0.1 151.15.185.183 192.168.0.0/24 }	17	151.15.185.183	-	9.9.8.8	-
192.168.0.1	*	* \{ 127.0.0.1 151.15.185.183 192.168.0.0/24 }	*	151.15.185.183	-	-	-
151.15.185.183	*	* \{ 192.168.0.0/24 }	17	-	-	9.9.8.8	-

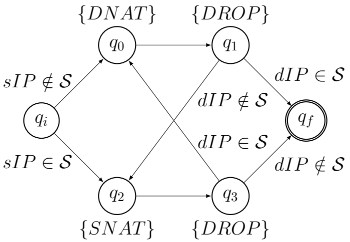
Generation



How to generate functions

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

- Assign **Labels** to nodes
  - `DROP`
  - `SNAT`
  - `DNAT`
- **Different expressive power**



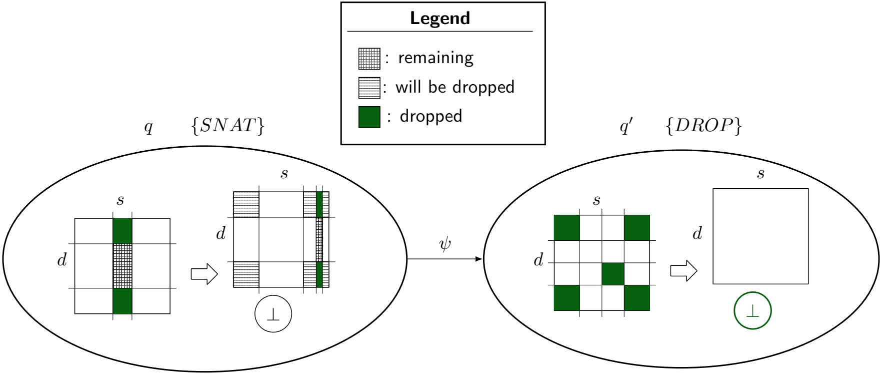
Algorithm

- **For each** pair (P, t) with $t \neq \bot$
  - Find the **path**
  - **For each** node q
    - Preceding nodes $\rightarrow \mathbf{P_q}$
    - Labels in $q \rightarrow \mathbf{t_q}$
- Special management for `DROP` pairs $(P, \bot)$

Management of DROP pairs

Special management for `DROP` pairs $(P, \perp)$

- For each node: packets **still not managed**
- **Drop as many of these as possible**



Recap

The presented transcompilation approach

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

Ongoing and Future Works

Objective

- Preserve the structure of the original configuration: **Refactoring**
- **Lag** between original language and IFCL
- Full support of **tag** system in the original languages
- **Network** with more than one firewall
- Porting to **Software Defined Networks**

Problem with tags: *pf*

PF:

- Rules **read top-down**
- **Last matching** rule is applied
- **Tag** is applied **immediately** (evaluation continues)
- **Quick rules** are applied immediately (evaluation stops)

IFCL:

- Rules read top-down and **applied immediately**
- Tag does not stop the evaluation

Basic solution

Just rewrite **bottom-up** the same list of rules (prepending quick rules)

Example:

$(true,\ \texttt{DROP})$

$(src = 1.2.3.4,\ \texttt{ACCEPT})$

$(dst = 5.6.7.8,\ \texttt{NAT}(1.6.3.8, \star))$ ⚡

$(src = 8.8.8.8,\ \texttt{DROP})$

Basic solution

Just rewrite **bottom-up** the same list of rules (prepending quick rules)

Example:

$(true,\ \texttt{DROP})$
$(src = 1.2.3.4,\ \texttt{ACCEPT})$
$(dst = 5.6.7.8,\ \texttt{NAT}(1.6.3.8, \star))$ ↯
$(src = 8.8.8.8,\ \texttt{DROP})$

become

$(dst = 5.6.7.8,\ \texttt{NAT}(1.6.3.8, \star))$
$(src = 8.8.8.8,\ \texttt{DROP})$
$(src = 1.2.3.4,\ \texttt{ACCEPT})$
$(true,\ \texttt{DROP})$

Basic solution: tag

Divide each rule r into

quick part : r' ($\lightning$ + tag)

slow part : r'' (everything else)

Example:

$$R = \begin{cases} (r_1) \\ (r_2) \\ \ldots \\ (r_n) \end{cases}$$

Basic solution: tag

Divide each rule r into

quick part : r' (↯ + tag)

slow part : r'' (everything else)

Example:

$$R = \begin{cases} (r_1) \\ (r_2) \\ \dots \\ (r_n) \end{cases} \quad \Rightarrow \quad R' = \begin{cases} (r'_1) \\ (r'_2) \\ \dots \\ (r'_n) \end{cases} \qquad reverse(R'') = \begin{cases} (r''_n) \\ \dots \\ (r''_2) \\ (r''_1) \end{cases}$$

Basic solution: tag

Divide each rule r into

- quick part : r' (↯ + tag)
- slow part : r'' (everything else)

Example:

$$R = \begin{cases} (r_1) \\ (r_2) \\ \dots \\ (r_n) \end{cases} \quad \Rightarrow \quad R' = \begin{cases} (r'_1) \\ (r'_2) \\ \dots \\ (r'_n) \end{cases} \qquad reverse(R'') = \begin{cases} (r''_n) \\ \dots \\ (r''_2) \\ (r''_1) \end{cases}$$

The devil is in the detail

Problem with tags: Example

$(true,\ \texttt{DROP})$

$(src = 1.2.3.4 \wedge tag = a,\ \ tag \leftarrow b;\ \texttt{ACCEPT})$

$(dst = 5.6.7.8 \wedge tag = b,\ \ \texttt{NAT}(1.6.3.8, \star))$

Problem with tags: Example

$$(true,\ \text{DROP})$$
$$(src = 1.2.3.4 \wedge tag = a,\ tag \leftarrow b;\ \text{ACCEPT})$$
$$(dst = 5.6.7.8 \wedge tag = b,\ \text{NAT}(1.6.3.8, \star))$$

$$\Downarrow$$

$$(src = 1.2.3.4 \wedge tag = a,\ tag \leftarrow b)$$
$$(dst = 5.6.7.8 \wedge tag = b,\ \text{NAT}(1.6.3.8, \star))$$
$$(src = 1.2.3.4 \wedge tag = a,\ \text{ACCEPT})$$
$$(true,\ \text{DROP})$$

Problem with tags: Example

$(true,\ \text{DROP})$

$(src = 1.2.3.4 \wedge tag = a,\ \ tag \leftarrow b;\ \text{ACCEPT})$

$(dst = 5.6.7.8 \wedge tag = b,\ \ \text{NAT}(1.6.3.8, \star))$

⬇

$(src = 1.2.3.4 \wedge tag = a,\ \ tag \leftarrow b)$

$(dst = 5.6.7.8 \wedge tag = b,\ \ \text{NAT}(1.6.3.8, \star))$

$(src = 1.2.3.4 \wedge tag = \underline{b},\ \ \text{ACCEPT})$

$(true,\ \text{DROP})$

Problem with tags: Example

$(true, \ \texttt{DROP})$

$(src = 1.2.3.4 \wedge tag = a, \ \ tag \leftarrow b; \texttt{ACCEPT})$

$(src = 1.2.3.4 \wedge tag = c, \ \ tag \leftarrow b; \texttt{NAT}(\star, 5.2.7.4))$

$(dst = 5.6.7.8 \wedge tag = b, \ \ \texttt{NAT}(1.6.3.8, \star))$

Problem with tags: Example

$$(true,\ \text{DROP})$$
$$(src = 1.2.3.4 \wedge tag = a,\ \ tag \leftarrow b;\ \text{ACCEPT})$$
$$(src = 1.2.3.4 \wedge tag = c,\ \ tag \leftarrow b;\ \text{NAT}(\star, 5.2.7.4))$$
$$(dst = 5.6.7.8 \wedge tag = b,\ \ \text{NAT}(1.6.3.8, \star))$$

$$\Downarrow$$

$$(src = 1.2.3.4 \wedge tag = a,\ \ tag \leftarrow b)$$
$$(src = 1.2.3.4 \wedge tag = c,\ \ tag \leftarrow b)$$
$$(dst = 5.6.7.8 \wedge tag = b,\ \ \text{NAT}(1.6.3.8, \star))$$
$$(src = 1.2.3.4 \wedge tag = b,\ \ \text{NAT}(\star, 5.2.7.4))$$
$$(src = 1.2.3.4 \wedge tag = b,\ \ \text{ACCEPT})$$
$$(true,\ \text{DROP})$$

Problem with tags: Example

$(true,\ \text{DROP})$

$(src = 1.2.3.4 \wedge tag = a,\ tag \leftarrow b;\ \text{ACCEPT})$

$(src = 1.2.3.4 \wedge tag = c,\ tag \leftarrow b;\ \text{NAT}(\star, 5.2.7.4))$

$(dst = 5.6.7.8 \wedge tag = b,\ \text{NAT}(1.6.3.8, \star))$

⬇

$(src = 1.2.3.4 \wedge tag = a,\ tag \leftarrow b1)$

$(src = 1.2.3.4 \wedge tag = c,\ tag \leftarrow b2)$

$(dst = 5.6.7.8 \wedge tag = b1,\ tag \leftarrow b;\ \text{NAT}(1.6.3.8, \star))$

$(dst = 5.6.7.8 \wedge tag = b2,\ tag \leftarrow b;\ \text{NAT}(1.6.3.8, \star))$

$(src = 1.2.3.4 \wedge tag = b2,\ tag \leftarrow b;\ \text{NAT}(\star, 5.2.7.4))$

$(src = 1.2.3.4 \wedge tag = b1,\ tag \leftarrow b;\ \text{ACCEPT})$

$(true,\ \text{DROP})$

Program the hole network behaviour at high level

NetKAT: Kleene Algebra with Tests for Networks

Kleene Algebra for reasoning about network structure

Boolean Algebra for reasoning about switch behaviour

Packet Algebra for reasoning about packets

	$+$	$\cdot$	$\neg$	0	1
action (policy)	choice	composition		fail	skip
test (predicate)	disjunction	conjunction	negation	false	true

$f = n$ (test on a packet field) $f \leftarrow n$ (modification of a packet field)

Program the hole network behaviour at high level

Network topology : a NetKAT formula

Each Firewall configuration : NetKAT formula

Code Motion & Refactoring : Equational theory

Security property : NetKAT formula

Property verification : Equational theory

We need to implement **Compilation** from real firewall languages to NetKAT

From IFCL to NetKAT is quite simple:

Ruleset : a NetKAT formula

Control Diagram : as Network topology

Non-propagation of Tags : explicitly set to empty in ruleset

Program the hole network behaviour at high level

From IFCL ruleset to NetKAT formula ($(\!|\_|\!)$ is a syntactic translation)

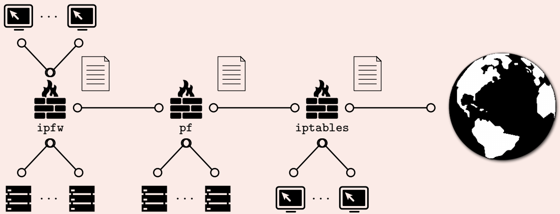
$$[\![(\phi, t); R]\!] = \begin{cases} (\!|\phi|\!) \cdot (\!|t|\!) + (\!|\neg\phi|\!) \cdot [\![\mathtt{R}]\!] & \text{if } t \in \{\text{ACCEPT}, \text{NAT}\} \\ (\!|\neg\phi|\!) \cdot [\![\mathtt{R}]\!] & \text{if } t = \text{DROP} \\ (\!|\phi|\!) \cdot (\!|t|\!) \cdot [\![\mathtt{R}]\!] + (\!|\neg\phi|\!) \cdot [\![\mathtt{R}]\!] & \text{if } t = \text{MARK}(m) \\ (\!|\phi|\!) \cdot [\![\mathtt{R'}]\!] + (\!|\neg\phi|\!) \cdot [\![\mathtt{R}]\!] & \text{if } t = \text{GOTO(R')} \\ (\!|\phi|\!) \cdot [\![\mathtt{R'}]\!] \cdot [\![\mathtt{R}]\!] + (\!|\neg\phi|\!) \cdot [\![\mathtt{R}]\!] & \text{if } t = \text{CALL(R')} \\ (\!|\phi|\!) + (\!|\neg\phi|\!) \cdot [\![\mathtt{R}]\!] & \text{if } t = \text{RETURN} \end{cases}$$

Program the hole network behaviour at high level

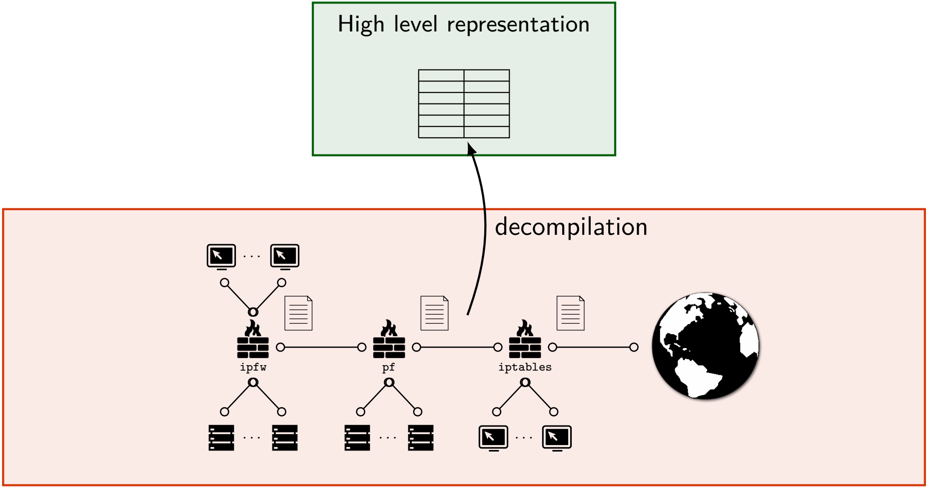
NetKAT for configuring traditional firewalls: NetKAT $\rightarrow$ specific language

- Each language corresponds to a **normal form**
- Equational **reduction** to the specific normal form
- **Compilation** from normal form of NetKAT to target language
- **Preserve the structure** of the original configuration **for free**

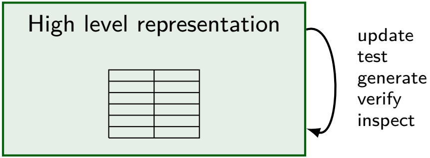
Our NEW Goal

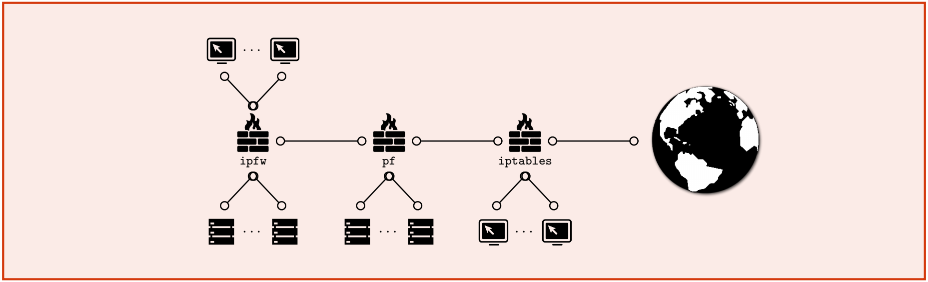


Our NEW Goal


High level representation
decompilation
ipfw
pf
iptables

Our NEW Goal


High level representation
update
test
generate
verify
inspect


ipfw
pf
iptables

Our NEW Goal