Policy Anomaly Detection for Distributed IPv6 Firewalls

Claas Lorenz and Bettina Schnor

Potsdam University
Institute for Computer Science
Operating Systems and Distributed Systems

Colmar, 2015-07-20
Outline

1. Motivation
2. Challenges and Approach
3. Runtime Measurements
4. Conclusion and Future Work
Figure: Share of IPv6 requests for Google services\(^1\).

\(^1\)cf. [Goo15]
Motivation

- Why IPv6?
  - Steady growth of the IPv6 share
  - Internet of Things works only with enough addresses (estimated: 100 trillion entities\(^2\))

- Often historically grown firewall policies

- Manual migration from IPv4 to IPv6 is challenging for large firewall instances

- IDTv6 project\(^3\):
  - ft6: Tests firewalls for RFC conformity
  - ad6: Finds policy anomalies firewall and network configurations

---

\(^2\)cf. [HFD12] p. 13
\(^3\)cf. [IDS13]
Challenges and Approach

**Challenges for Formal Verification**

- **Goal is the detection of the anomalies:**
  - cyclicity
    
    ip6tables –P OUTPUT DROP
    ip6tables –A OUTPUT –p udp –j ACCEPT
    ip6tables –A OUTPUT –p tcp –j OUTPUT
  
  - unreachability
  - shadowing
  - cross-path

- **Extension of the formalism of Jeffrey and Samak**
  - Larger base header with IPv6 (320 vs. 104 bits) → enlarges search space
  - Extension header chains of arbitrary length

---

4 cf. [JS09]
5 Illustration from [BE06]
Formal Integration - Summary

- Encoding of the firewall rules and the network as Kripke Structure
- Transformation to SAT
- Problem encodings for two additional anomalies: shadowing, cross-path
- For details please refer our paper
Runtime Measurements

2001:abc:def::0/48

gateway

DMZ

clients

prefix:1::0/64

prefix:3::0/64

prefix:2::0/64

prefix:18::0/64

server

clients

prefix:2::1::1

prefix:2::1::2

prefix:18::1

prefix:1018::1

prefix:3::1

prefix:1003::1

client

web

ldap

sub1

sub24

prefix:1::1

prefix:1::2

prefix:1::5

prefix:1::6

prefix:1::8

prefix:1::1

prefix:1::2

prefix:1::4

prefix:1::3

prefix:1::7

prefix:1::8

file

dns

vpn

web

admin

db

Internet

...
Runtime Measurements - Methodology

- Quantification and scalability estimation
- Synthetic network and firewall topology
- Inspired by our campus’ network
- No anomalies inherited (→ worst-case runtime anticipated)
- Processing was single threaded
- Two independent phases with two parameters each:
  - Building phase: First Use vs. Reuse
  - Solving phase: MiniSAT vs. Clasp
- Measurement environment:

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>Intel i7-3630QM</td>
<td>Cores</td>
<td>4</td>
</tr>
<tr>
<td>Freq</td>
<td>2.4GHz</td>
<td>RAM</td>
<td>8GB</td>
</tr>
<tr>
<td>OS</td>
<td>Arch Linux</td>
<td>Kernel</td>
<td>v3.16.2</td>
</tr>
<tr>
<td>Python</td>
<td>v3.4.1</td>
<td>Redis</td>
<td>v2.8.17</td>
</tr>
<tr>
<td>MiniSAT</td>
<td>v2.2.0</td>
<td>Clasp</td>
<td>v3.0.3</td>
</tr>
</tbody>
</table>

Claas Lorenz and Bettina Schnor
Runtime Measurements - Results

- Runtime is superlinear but subquadratic
- \( f(x) = ax^2 + bx + c \) with
  - \( x \) is the number of rules
  - \( a \approx 0.0002, b \approx -0.06 \) and \( c \approx 3.54 \) (Building, First Use)
  - \( a \approx 0.0004, b \approx -0.04 \) and \( c \approx 2.85 \) (Solving, Clasp)
- Longest total runtime of \(~37.2\) minutes
- Memory usage was linear
Conclusion

- Extensions of the formalism of Jeffrey and Samak for:
  - Shadowing and cross-path detection
  - IPv6 base headers
  - Extension header chains
- Runtime does not behave exponential but low quadratic
  \(\rightarrow\) acceptable for the migration scenario
Future Work

- Performance improvement by:
  - Native interfaces for the solver
  - Parallelization
  - Learning from intermediate results
- Expressiveness: Stateful firewalls, VPN-tunnels, etc.
- Applicability for SDN?
Thank you for your attention!
Literature I


Literature II