Policy Routing using Process-Level Identifiers

IEEE International Symposium on Software Defined Systems
April 4th, 2016, Berlin, Germany

Oliver Michel, Eric Keller
Networking and Security Research Group

University of Colorado Boulder
Network Policies
Network Policies

Load Balancing
Network Policies

Address Translation
Network Policies

Intrusion Detection
Network Policies

Firewalling
Limited Identifiers

- Ethernet
- IP
- TCP/UDP
Limited Identifiers

- **Ethernet**
  - Source
  - Destination
  - EtherType

- **IP**

- **TCP/UDP**
Limited Identifiers

- Ethernet
  - Source
  - Destination
  - EtherType
- IP
  - Source
  - Destination
  - Protocol
  - DSCP
  - ECN
- TCP/UDP
Limited Identifiers

- **Ethernet**
  - Source
  - Destination
  - EtherType

- **IP**
  - Source
  - Destination
  - Protocol
  - DSCP
  - ECN

- **TCP/UDP**
  - Source
  - Destination
Why fine-grained identifiers?

• Uniquely identifying user sessions
Why fine-grained identifiers?

- Uniquely identifying user sessions

MAC, IP
Why fine-grained identifiers?

• Uniquely identifying user sessions
Why fine-grained identifiers?

- Uniquely identifying user sessions
Why fine-grained identifiers?

- Isolating vulnerable software
Why fine-grained identifiers?

• Isolating vulnerable software
Why fine-grained identifiers?

- Isolating vulnerable software

```
$ openssl sha1 /usr/sbin/httpd
SHA1(/usr/sbin/httpd)=5fdbdb587fce265656fd3e2960a6293262efedb7
```
Why fine-grained identifiers?

• Isolating vulnerable software

```
$ openssl sha1 /usr/sbin/httpd
SHA1(/usr/sbin/httpd)=5fdbdb587fce265656fd3e2960a6293262efedb7
```
Why fine-grained identifiers?

- Isolating vulnerable software

```bash
$ openssl sha1 /usr/sbin/httpd
SHA1(/usr/sbin/httpd)=5fdadb587f6565fd3e2960a6293262efed2b7
```
Fine-Grained Information

user space

operating system

Network Interface
IP/MAC

Process

Process
Fine-Grained Information

user space

process

Port

network interface
IP/MAC

operating system
Fine-Grained Information

user space

<table>
<thead>
<tr>
<th>Process</th>
<th>Process</th>
</tr>
</thead>
</table>

operating system

<table>
<thead>
<tr>
<th>Port</th>
<th>PID</th>
</tr>
</thead>
</table>

Network Interface

IP/MAC
Fine-Grained Information

user space

operating system

Network Interface
IP/MAC

Process

PID

GID

Port
Fine-Grained Information

user space

operating system

Process

Port

Process

PID

GID

UID

Network Interface
IP/MAC
**Fine-Grained Information**

- **user space**
  - Process
  - Port

- **operating system**
  - Process
  - PID
  - GID
  - UID
  - cgroups

- Network Interface
  - IP/MAC
Fine-Grained Information

user space

Process

Port

PID

GID

UID

cgroups

open files

operating system

Network Interface

IP/MAC
Fine-Grained Information

user space

- Process

operating system

- Port
  - PID
  - GID
  - UID
  - cgroups
  - open files
  - exe fingerprint

Network Interface
- IP/MAC
Fine-Grained Information

user space

operating system

network

- Process
- Process
- PID
- GID
- UID
- cgroups
- open files
- exe fingerprint
- Network Interface
  - IP/MAC
  - MAC Source
  - MAC Destination
  - EtherType
  - IP Source
  - IP Destination
  - Protocol
  - TCP/UDP Source
  - TCP/UDP Destination

- Network Interface
- IP/MAC
- MAC Source
- MAC Destination
- EtherType
- IP Source
- IP Destination
- Protocol
- TCP/UDP Source
- TCP/UDP Destination
Fine-Grained Information

user space

operating system

network

Process

Process

Network Interface

IP/MAC

MAC Source

MAC Destination

EtherType

IP Source

IP Destination

Protocol

TCP/UDP Source

TCP/UDP Destination

PID

GID

UID

cgroups

open files

exe fingerprint
Benefiting Scenarios

- Uniquely identifying user sessions
- Isolating vulnerable software
Benefiting Scenarios

• Uniquely identifying user sessions
• Isolating vulnerable software
• Identifying services
Benefiting Scenarios

• Uniquely identifying user sessions
• Isolating vulnerable software
• Identifying services
• Quality of Service
Benefiting Scenarios

- Uniquely identifying user sessions
- Isolating vulnerable software
- Identifying services
- Quality of Service
- Forensic Analysis
Policy Routing using Process-Level Identifiers
PRPL Overview
PRPL Overview

controller
PRPL Overview

Distributing and configuring Policy

controller
PRPL Overview

Distributing and configuring Policy

controller
PRPL Overview

Tagging Packets

Distributing and configuring Policy

controller
PRPL Overview

Tagging Packets → Distributing and configuring Policy → Forwarding

controller
Tagging

- Insert a custom header containing a token associated with some policy
Tagging

- Insert a custom header containing a token associated with some policy
Tagging

- Insert a custom header containing a token associated with some policy
Tagging

• Insert a custom header containing a token associated with some policy
Tagging

Host

Policy Controller

User domain

Admin domain

Process
Tagging

Host

Policy Controller

PRPL Agent

Process

user domain

admin domain
Tagging

Host

Policy Controller

PRPL Agent

Process

user domain

admin domain

request communication stream
Tagging

Host

Policy Controller

PRPL Agent

Process

user domain

admin domain

request communication stream

obtain token
Tagging

Host

Policy Controller

user domain

admin domain

request communication stream

obtain token

Process

classify/mark

tag/forward

configure

PRPL Agent
Forwarding

• Programmable Hardware
  [Kangaroo INFOCOM '10, SDN Chip SIGCOMM '13, Intel FM6000 switch silicon]

• Dataplane Forwarding Model in P4 [SIGCOMM CCR 2014]
**Forwarding**

- **Programmable Hardware**
  [Kangaroo INFOCOM ’10, SDN Chip
  SIGCOMM ’13, Intel FM6000 switch
  silicon]

- **Dataplane Forwarding Model**
  in P4  [SIGCOMM CCR 2014]
Forwarding

- **Programmable Hardware**
  [Kangaroo INFOCOM ’10, SDN Chip SIGCOMM ’13, Intel FM6000 switch silicon]

- **Dataplane Forwarding Model in P4** [SIGCOMM CCR 2014]
Implementation
Implementation

- Linux on-board tools: iptables, custom routing, tunnel devices
Implementation

• Linux on-board tools: iptables, custom routing, tunnel devices

• P4: Matching on token
Implementation

• Linux on-board tools: iptables, custom routing, tunnel devices

• P4: Matching on token

• Prototype
  • P4 behavioral model
  • tag based on uid
  • forward or drop
Implementation

- No performance penalty for packets < 200 Bytes
Future Work and Conclusion
Future Work and Conclusion

- Network Management can greatly benefit from fine-grained process-level information
Future Work and Conclusion

• Network Management can greatly benefit from fine-grained process-level information

• System Architecture and Prototype enabling packet processing based on such information
Future Work and Conclusion

• Network Management can greatly benefit from fine-grained process-level information

• System Architecture and Prototype enabling packet processing based on such information

• Future work: expansion beyond current examples, more complex policies
Source Code

https://github.com/nsr-colorado/prpl
Backup Slides
Future Work

- Study feasibility of more complex policy scenarios
- Granularity of Tokens
- Controller - Agent Interface
- Proactive vs. reactive configuration
- Trust in tagging process
Tagging

- Token sizes between 16 bits and 32 bits sufficient even for large networks
Programmable Hardware

• new custom ASICs can achieve such flexibility at terabit speeds [Kangaroo INFOCOM ’10, SDN Chip SIGCOMM ’13, Intel FM6000 switch silicon]

• some switches are more programmable than others:
  • FPGA (Xilinx, Altera, Corsa)
  • NPU (Ezchip, Netronome)
  • CPU (OVS, …)
P4 Language

- P4 program configures forwarding behavior (abstract forwarding model)
- express serial dependencies (e.g. ARP/L3 Routing)
- P4 compiler translates into a target-specific representation
- OF can still be used to install and query rules once forwarding model is defined
P4 Forwarding Model / Runtime
P4 Forwarding Model / Runtime
P4 Forwarding Model / Runtime
P4 Forwarding Model / Runtime
P4 Forwarding Model / Runtime

L2L3.p4

Parsing

Switch

Parse

Egress Queues

Parser

Packet Metadata

Parser

Routing

Firewall

NAT

Controller

COMPILE

VLAN

IP4

IP6

TCP

UDP

Eth
P4 Forwarding Model / Runtime

L2L3.p4

Parser

Egress Queues

Egress Queues

Controller

Switch

Match/Action Tables

Packet Metadata

Routing

Firewall

NAT

parser

Eh

VLAN

IP4

IP6

TCP

UDP

COMPILE
P4 Forwarding Model / Runtime
P4 Forwarding Model / Runtime
P4 Forwarding Model / Runtime

Parser

Switch

Controller

Routing
Firewall
NAT

Egress Queues

Packet Metadata

Match/Action Tables

OF1-3.p4

COMPILE
P4 Forwarding Model / Runtime

OpenFlow 1.3

Controller

Routing  Firewall  NAT

Switch

Parser

Egress Queues

Packet Metadata

Match/Action Tables

Eth

VLAN  IP4  IP6

TCP  UDP

Parser

Egress Queues

Packet Metadata

Match/Action Tables

Controller

Routing  Firewall  NAT

Switch

Compile

OF1-3.p4
P4 Parsing

Incoming Packet

- IP
- Policy Token
- Ethernet

Parsing FSM

- TCP
- UDP
- IP
- Eth
- PRPL

Routing Table

<table>
<thead>
<tr>
<th>IP Address</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.0.1/24</td>
<td>1</td>
</tr>
<tr>
<td>10.0.2/24</td>
<td>2</td>
</tr>
</tbody>
</table>

Policy Table

<table>
<thead>
<tr>
<th>Policy Token</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xd3...42</td>
<td>drop</td>
</tr>
<tr>
<td>0xa4...29</td>
<td>reroute</td>
</tr>
<tr>
<td>0xa6...76</td>
<td>continue</td>
</tr>
</tbody>
</table>
header_type prpl_t {
  fields {
    token : 8;
  }
}

header prpl_t prpl;

parser start {
  return parse_ethernet;
}
parser parse_ethernet {
  extract(ethernet);
  return parse_prpl;
}
parser parse_prpl {
  extract(prpl);
  return ingress;
}

table prpl {
  reads {
    prpl.token : exact;
  }
  actions { _nop; _drop; forward; }
  size : 128;
}

table_set_default prpl _nop
table_add prpl _nop 0x00000001 =>
table_add prpl _drop 0x00000002 =>
table_add prpl forward 0x00000001 => 4
iptables

1003

process

iptables OUTPUT chain
--uid-owner 1003 --set-mark 0xd2

routing rules
from all mark 0xd2 lookup policy_d2

Policy Agent

Network

policy_d2
default dev tun2

policy_54
default dev tun5

/admin domain

/etc/iproute2

/rt_tables