Testing iptables firewall rules with scapy

Comply with Cybersecurity Requirements like UNECE R-155

February 3, 2024

Simone Weiß and Michael Estner

Agenda

01 Who are we

02 Why test your firewall rules

03 Netfilter and iptables

04 Scapy

05 Firewall tests with scapy

06 Summary

Who are we

Simone Weiß

Embedded Systems Developer
Elektrobit – Driving the future of software

- Degree in computer science
- Worked mostly on embedded Linux distributions
- Cybersecurity (ISO 21434/UNECE)
- Private: Cats and nature

Michael Estner

Senior Software engineer
Elektrobit – Driving the future of software

- Degree in electrical engineering
- Embedded Linux & Python
- Private: MMA, hiking and cooking

We are Elektrobit

Your software solution provider

35 YEARS AUTOMOTIVE EXPERIENCE

10 YEARS OPEN-SOURCE EXPERIENCE

EMBEDDED LINUX SYSTEM WITH YOCTO

EB CORBOS LINUX BUILT ON UBUNTU

CYBERSECURITY MANAGEMENT SYSTEM (CSMS) COMPLIANT

Why test your firewall rules?

Cybersecurity requirements

February 3, 2024

Cybersecurity requirements

UN R155 and ISO 21434 – Cybersecurity maintenance

UN R155

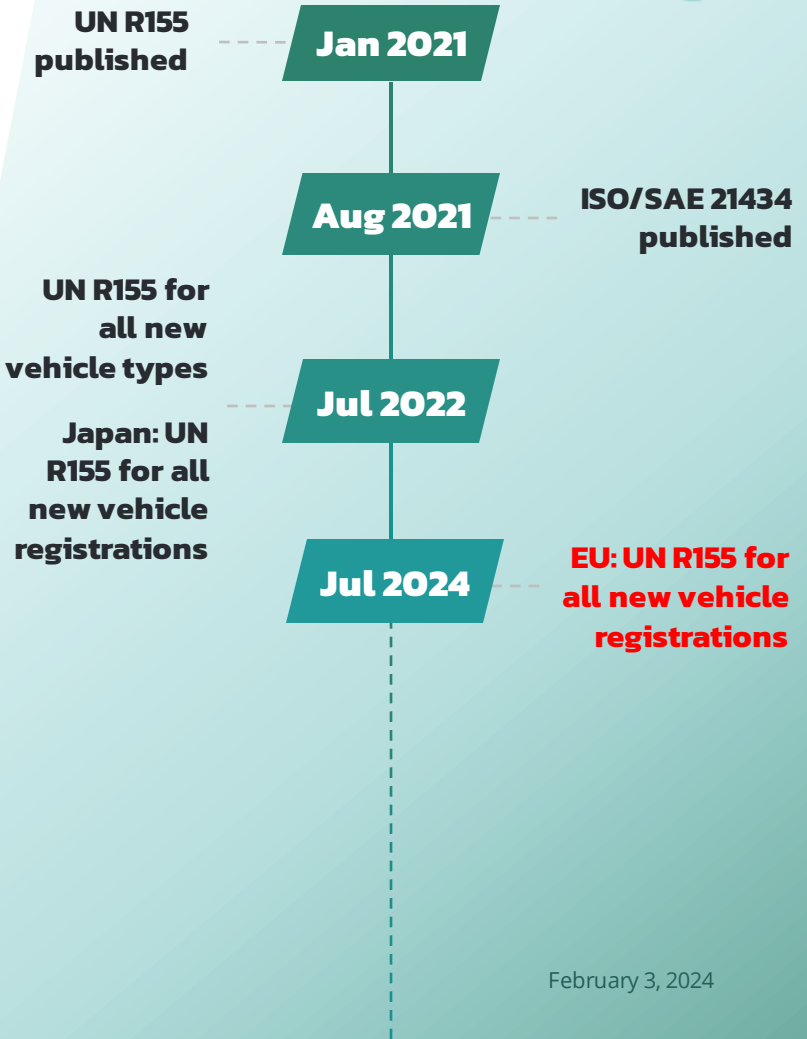
Regulation on cybersecurity

- Mandatory in all UNECE member countries (64)
- Defines requirements for the cybersecurity management system (CSMS) in vehicles
- Ensures that cybersecurity practices and measures are adequately applied across the development process and life cycle of vehicles
- Applies to all software in vehicles

ISO/SAE 21434

Road vehicles – Cybersecurity engineering

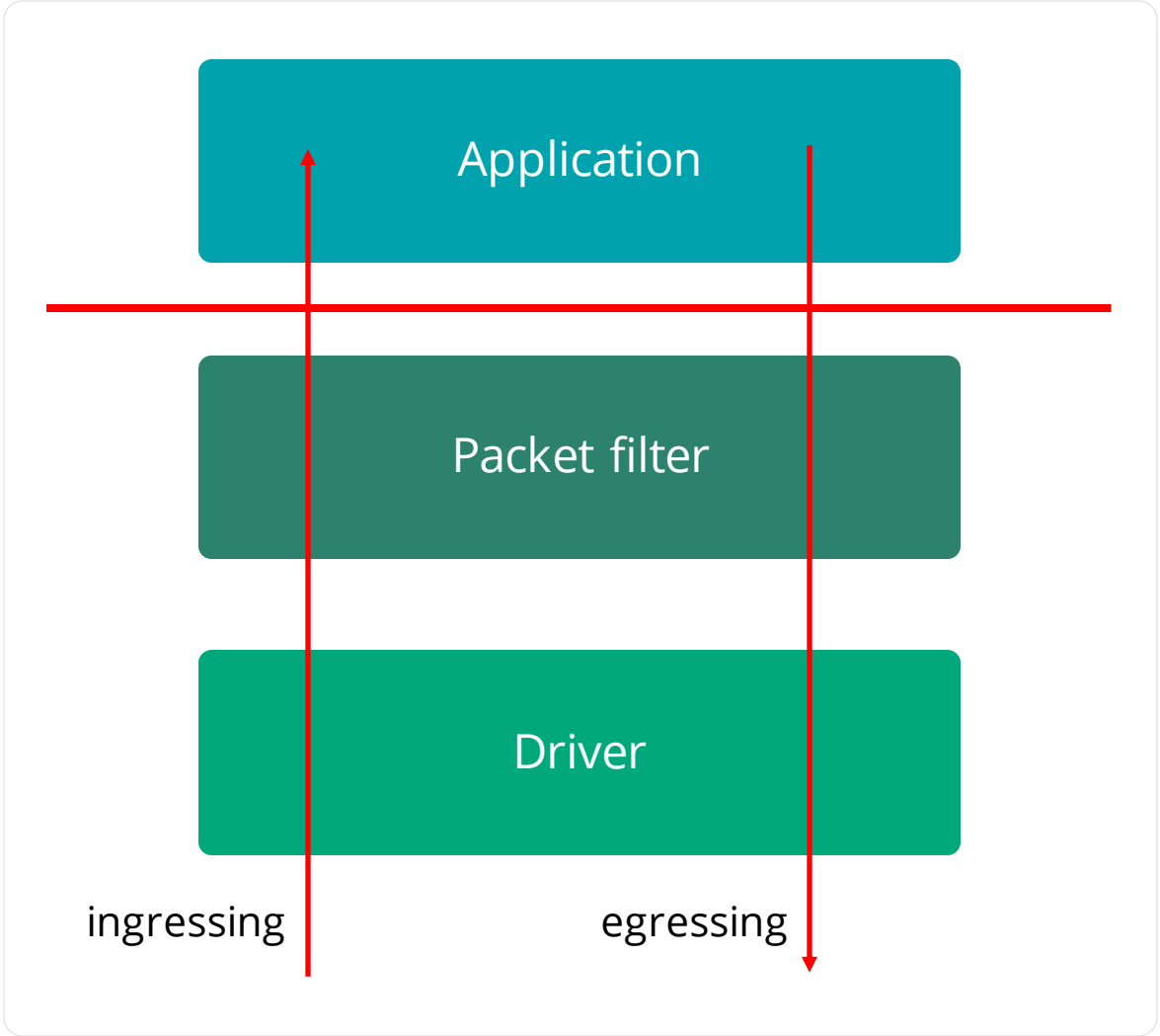
- Requirements and recommendations to develop a cybersecurity product
- Baseline for CSMS



Introduction

Packet filtering

- Packet filter inspects traffic in the network stack
- Use cases
  - Firewall
  - Traffic statistics
  - Logging
  - ...
- Linux packet filter
  - Kernelspace
    - netfilter
  - Userspace
    - iptables, ip6tables, ebtables, arptables
    - nftables



Netfilter

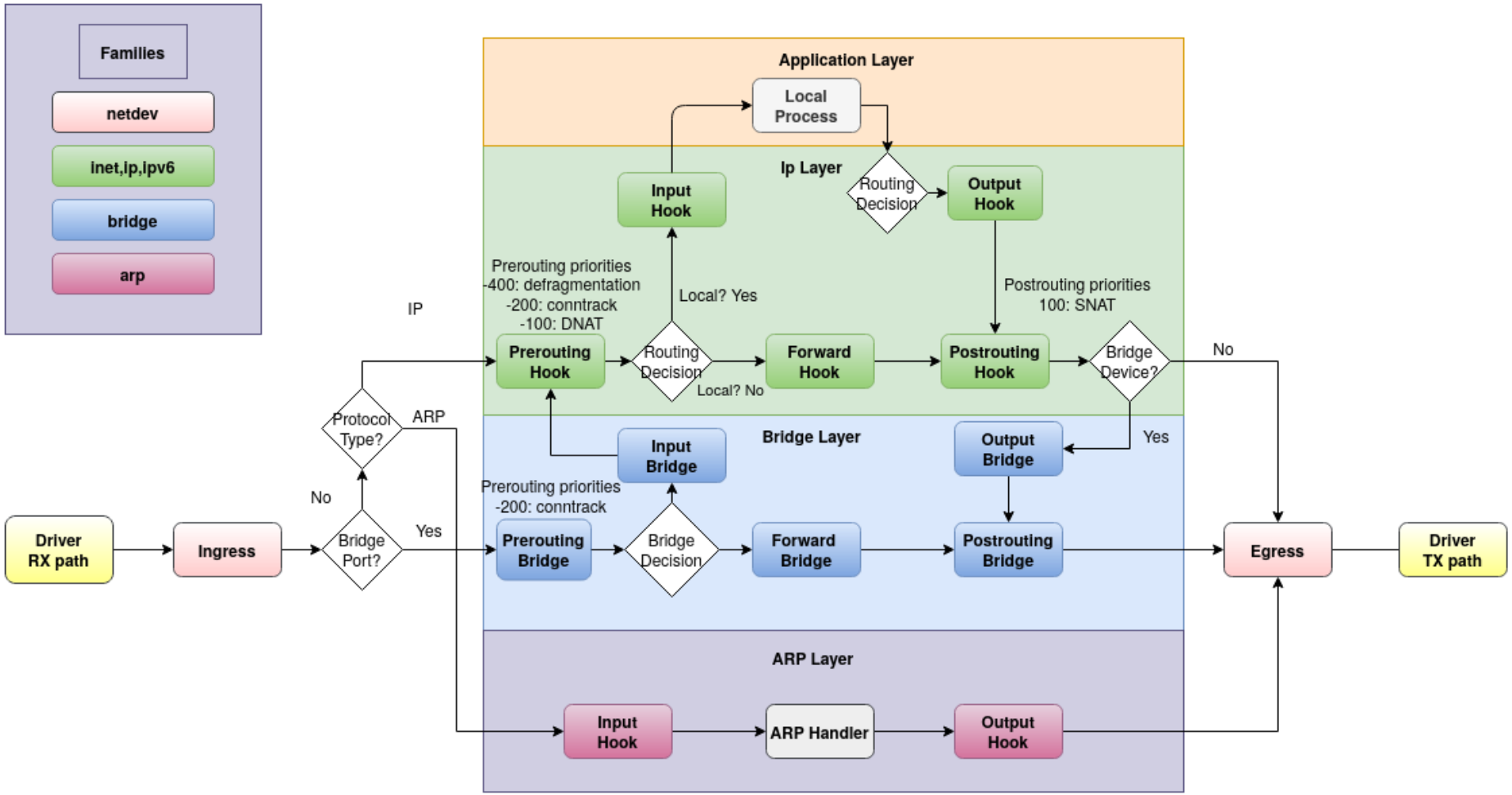
Community-driven FOSS project

Provides packet filtering and network address translation for the Linux kernel

Key components:

- Hooks: Intercept packets at different stages
- Tables: Has a specific packet-handling task
- Chains: Sequences of rules within a table
- Targets: Action when packet matches a rule

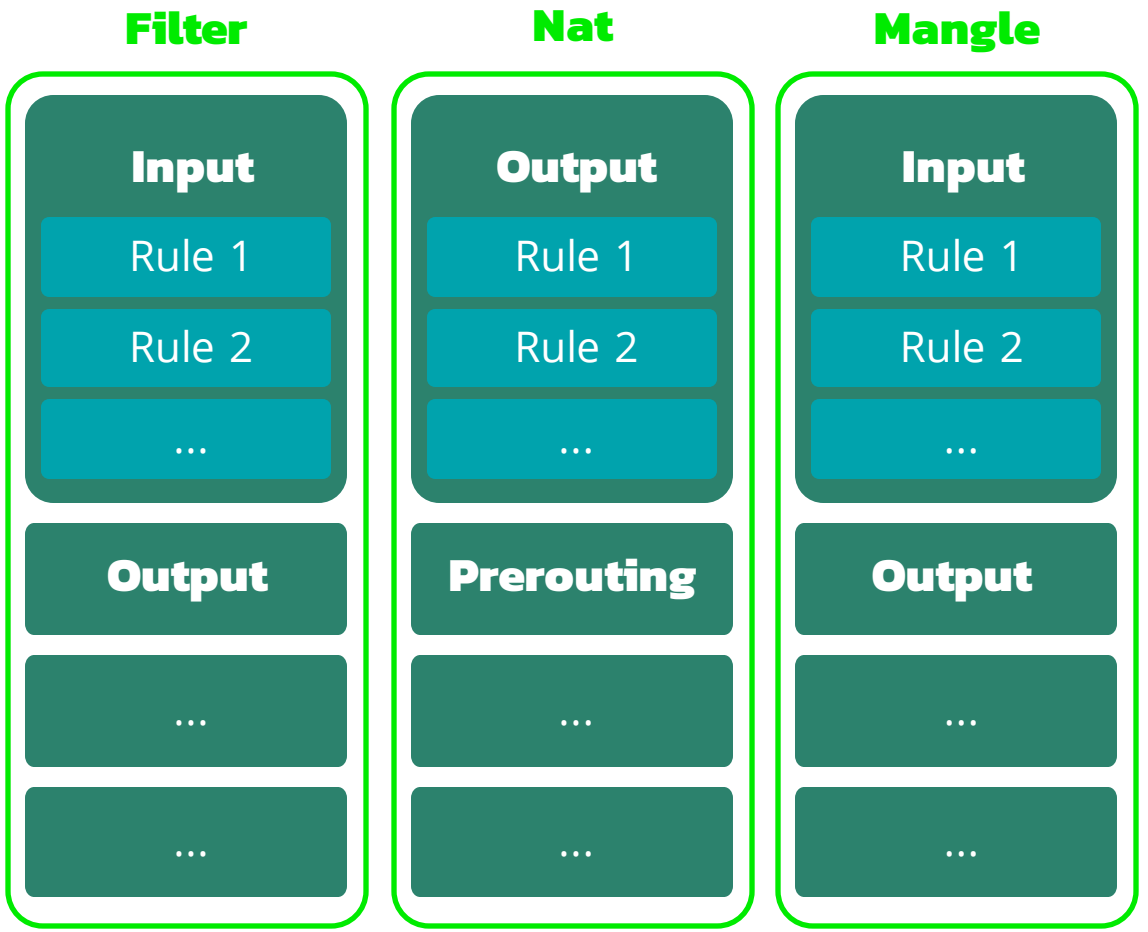
Netfilter



https://wiki.nftables.org/wiki-nftables/index.php/Netfilter_hooks 19.12.2023 15:19

Ip(6)tables

- Userspace program to interact with the netfilter
- Organized in **tables** that contains several chains
- A **chain** is a list of rules which can match a set of packets
- A **rule** specifies what to do with a packet that matches
  - If so, the next rule is the one specified by the target: User-defined, or ACCEPT, DROP, QUEUE, or RETURN
  - If not, next rule is the next in the chain



Ip(6)tables

```
*filter
:INPUT ACCEPT [0:0]
:FORWARD DROP [0:0]
:OUTPUT ACCEPT [0:0]
:DOCKER - [0:0]
:DOCKER-ISOLATION-STAGE-1 - [0:0]
:DOCKER-ISOLATION-STAGE-2 - [0:0]
:DOCKER-USER - [0:0]
-A FORWARD -j DOCKER-USER
-A FORWARD -j DOCKER-ISOLATION-STAGE-1
-A FORWARD -o docker0 -m conntrack --ctstate RELATED,ESTABLISHED -j ACCEPT
-A FORWARD -o docker0 -j DOCKER
-A FORWARD -i docker0 ! -o docker0 -j ACCEPT
-A FORWARD -i docker0 -o docker0 -j ACCEPT
-A DOCKER-ISOLATION-STAGE-1 -i docker0 ! -o docker0 -j DOCKER-ISOLATION-STAGE-2
-A DOCKER-ISOLATION-STAGE-1 -j RETURN
-A DOCKER-ISOLATION-STAGE-2 -o docker0 -j DROP
-A DOCKER-ISOLATION-STAGE-2 -j RETURN
-A DOCKER-USER -j RETURN
COMMIT
*nat
```

- Chain FORWARD
  - Rule 1: All traffic is sent to the DOCKER-USER chain
- Chain DOCKER-USER
  - Rule 1: Target is return for all traffic
- Chain FORWARD
  - Rule 2: All traffic is sent to the DOCKER-ISOLATION-STAGE-1 chain
- Chain DOCKER-ISOLATION-STAGE-1
  - Rule 1: All traffic from docker0 interface to anywhere that is not itself is sent to the DOCKER-ISOLATION-STAGE-2 chain.
  - Rule 2: Return all traffic
- Chain DOCKER-ISOLATION-STAGE-2
  - Rule 1: Drop all traffic to docker0
  - Rule 2: Return else

Ip(6)tables

```
*filter
:INPUT ACCEPT [0:0]
:FORWARD DROP [0:0]
:OUTPUT ACCEPT [0:0]
:DOCKER - [0:0]
:DOCKER-ISOLATION-STAGE-1 - [0:0]
:DOCKER-ISOLATION-STAGE-2 - [0:0]
:DOCKER-USER - [0:0]
-A FORWARD -j DOCKER-USER
-A FORWARD -j DOCKER-ISOLATION-STAGE-1
-A FORWARD -o docker0 -m conntrack --ctstate RELATED,ESTABLISHED -j ACCEPT
-A FORWARD -o docker0 -j DOCKER
-A FORWARD -i docker0 ! -o docker0 -j ACCEPT
-A FORWARD -i docker0 -o docker0 -j ACCEPT
-A DOCKER-ISOLATION-STAGE-1 -i docker0 ! -o docker0 -j DOCKER-ISOLATION-STAGE-2
-A DOCKER-ISOLATION-STAGE-1 -j RETURN
-A DOCKER-ISOLATION-STAGE-2 -o docker0 -j DROP
-A DOCKER-ISOLATION-STAGE-2 -j RETURN
-A DOCKER-USER -j RETURN
COMMIT
*nat
```

- Chain FORWARD
  - Rule 1: All traffic is sent to the DOCKER-USER chain
- Chain DOCKER-USER
  - Rule 1: Target is return for all traffic
- Chain FORWARD
  - Rule 2: All traffic is sent to the DOCKER-ISOLATION-STAGE-1 chain
- Chain DOCKER-ISOLATION-STAGE-1
  - Rule 1: All traffic from docker0 interface to anywhere that is not itself is sent to the DOCKER-ISOLATION-STAGE-2 chain.
  - Rule 2: Return all traffic
- Chain DOCKER-ISOLATION-STAGE-2
  - Rule 1: Drop all traffic to docker0
  - Rule 2: Return else

Ip(6)tables

```
*filter
:INPUT ACCEPT [0:0]
:FORWARD DROP [0:0]
:OUTPUT ACCEPT [0:0]
:DOCKER - [0:0]
:DOCKER-ISOLATION-STAGE-1 - [0:0]
:DOCKER-ISOLATION-STAGE-2 - [0:0]
:DOCKER-USER - [0:0]
-A FORWARD -j DOCKER-USER
-A FORWARD -j DOCKER-ISOLATION-STAGE-1
-A FORWARD -o docker0 -m conntrack --ctstate RELATED,ESTABLISHED -j ACCEPT
-A FORWARD -o docker0 -j DOCKER
-A FORWARD -i docker0 ! -o docker0 -j ACCEPT
-A FORWARD -i docker0 -o docker0 -j ACCEPT
-A DOCKER-ISOLATION-STAGE-1 -i docker0 ! -o docker0 -j DOCKER-ISOLATION-STAGE-2
-A DOCKER-ISOLATION-STAGE-1 -j RETURN
-A DOCKER-ISOLATION-STAGE-2 -o docker0 -j DROP
-A DOCKER-ISOLATION-STAGE-2 -j RETURN
-A DOCKER-USER -j RETURN
COMMIT
*nat
```

- Chain FORWARD
  - Rule 1: All traffic is sent to the DOCKER-USER chain
- Chain DOCKER-USER
  - Rule 1: Target is return for all traffic
- Chain FORWARD
  - Rule 2: All traffic is sent to the DOCKER-ISOLATION-STAGE-1 chain
- Chain DOCKER-ISOLATION-STAGE-1
  - Rule 1: All traffic from docker0 interface to anywhere that is not itself is sent to the DOCKER-ISOLATION-STAGE-2 chain.
  - Rule 2: Return all traffic
- Chain DOCKER-ISOLATION-STAGE-2
  - Rule 1: Drop all traffic to docker0
  - Rule 2: Return else

Ip(6)tables

```
*filter
:INPUT ACCEPT [0:0]
:FORWARD DROP [0:0]
:OUTPUT ACCEPT [0:0]
:DOCKER - [0:0]
:DOCKER-ISOLATION-STAGE-1 - [0:0]
:DOCKER-ISOLATION-STAGE-2 - [0:0]
:DOCKER-USER - [0:0]
-A FORWARD -j DOCKER-USER
-A FORWARD -j DOCKER-ISOLATION-STAGE-1
-A FORWARD -o docker0 -m conntrack --ctstate RELATED,ESTABLISHED -j ACCEPT
-A FORWARD -o docker0 -j DOCKER
-A FORWARD -i docker0 ! -o docker0 -j ACCEPT
-A FORWARD -i docker0 -o docker0 -j ACCEPT
-A DOCKER-ISOLATION-STAGE-1 -i docker0 ! -o docker0 -j DOCKER-ISOLATION-STAGE-2
-A DOCKER-ISOLATION-STAGE-1 -j RETURN
-A DOCKER-ISOLATION-STAGE-2 -o docker0 -j DROP
-A DOCKER-ISOLATION-STAGE-2 -j RETURN
-A DOCKER-USER -j RETURN
COMMIT
*nat
```

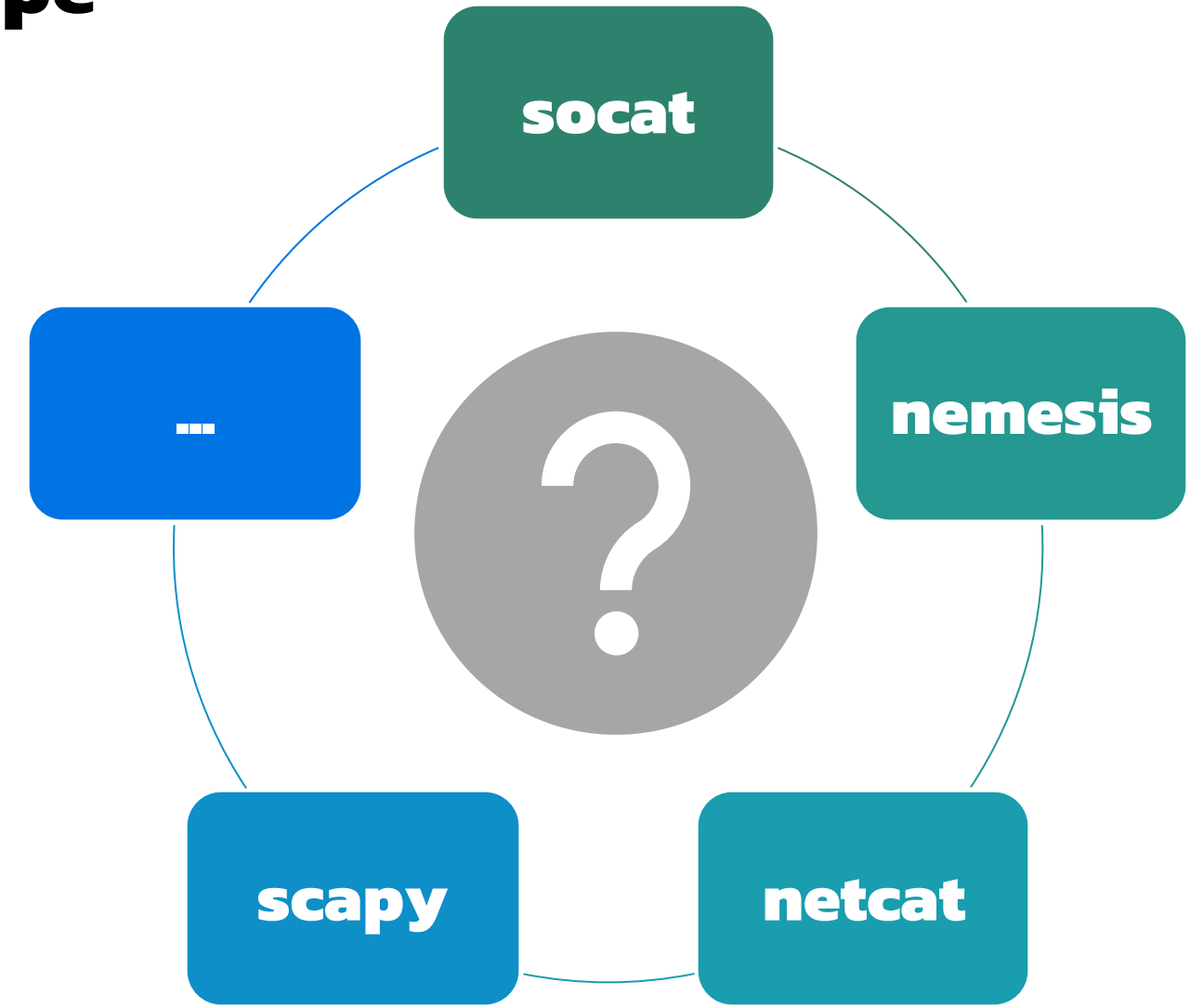
- Chain FORWARD
  - Rule 1: All traffic is sent to the DOCKER-USER chain
- Chain DOCKER-USER
  - Rule 1: Target is return for all traffic
- Chain FORWARD
  - Rule 2: All traffic is sent to the DOCKER-ISOLATION-STAGE-1 chain
- Chain DOCKER-ISOLATION-STAGE-1
  - Rule 1: All traffic from docker0 interface to anywhere that is not itself is sent to the DOCKER-ISOLATION-STAGE-2 chain.
  - Rule 2: Return all traffic
- Chain DOCKER-ISOLATION-STAGE-2
  - Rule 1: Drop all traffic to docker0
  - Rule 2: Return else

Ip(6)tables

```
*filter
:INPUT ACCEPT [0:0]
:FORWARD DROP [0:0]
:OUTPUT ACCEPT [0:0]
:DOCKER - [0:0]
:DOCKER-ISOLATION-STAGE-1 - [0:0]
:DOCKER-ISOLATION-STAGE-2 - [0:0]
:DOCKER-USER - [0:0]
-A FORWARD -j DOCKER-USER
-A FORWARD -j DOCKER-ISOLATION-STAGE-1
-A FORWARD -o docker0 -m conntrack --ctstate RELATED,ESTABLISHED -j ACCEPT
-A FORWARD -o docker0 -j DOCKER
-A FORWARD -i docker0 ! -o docker0 -j ACCEPT
-A FORWARD -i docker0 -o docker0 -j ACCEPT
-A DOCKER-ISOLATION-STAGE-1 -i docker0 ! -o docker0 -j DOCKER-ISOLATION-STAGE-2
-A DOCKER-ISOLATION-STAGE-1 -j RETURN
-A DOCKER-ISOLATION-STAGE-2 -o docker0 -j DROP
-A DOCKER-ISOLATION-STAGE-2 -j RETURN
-A DOCKER-USER -j RETURN
COMMIT
*nat
```

- Chain FORWARD
  - Rule 1: All traffic is sent to the DOCKER-USER chain
- Chain DOCKER-USER
  - Rule 1: Target is return for all traffic
- Chain FORWARD
  - Rule 2: All traffic is sent to the DOCKER-ISOLATION-STAGE-1 chain
- Chain DOCKER-ISOLATION-STAGE-1
  - Rule 1: All traffic from docker0 interface to anywhere that is not itself is sent to the DOCKER-ISOLATION-STAGE-2 chain.
  - Rule 2: Return all traffic
- Chain DOCKER-ISOLATION-STAGE-2
  - Rule 1: Drop all traffic to docker0
  - Rule 2: Return else

Toollandscape



Why scapy?

- Python based interactive packet manipulating library
- With scapy you can define, send and receive complete custom packets
- You can manipulate across different layers
- Low barrier to create custom network packets
- Easy to integrate in the existing test eco system

Transport Layer

- **TCP**
- **UDP**

Network Layer

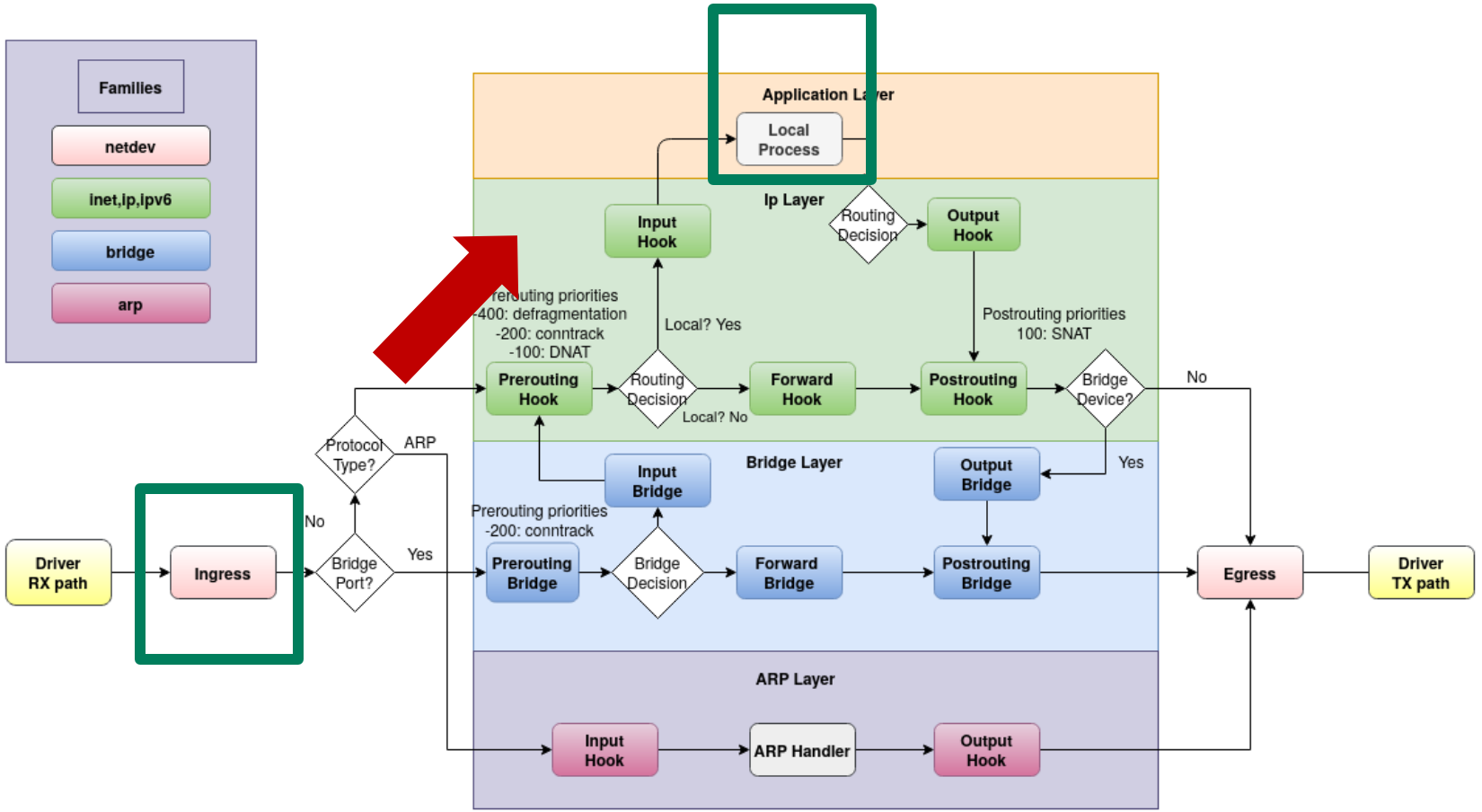
- **IPV4**
- **IPV6**

Link Layer

- **Ethernet**

Scapy and the netfilter

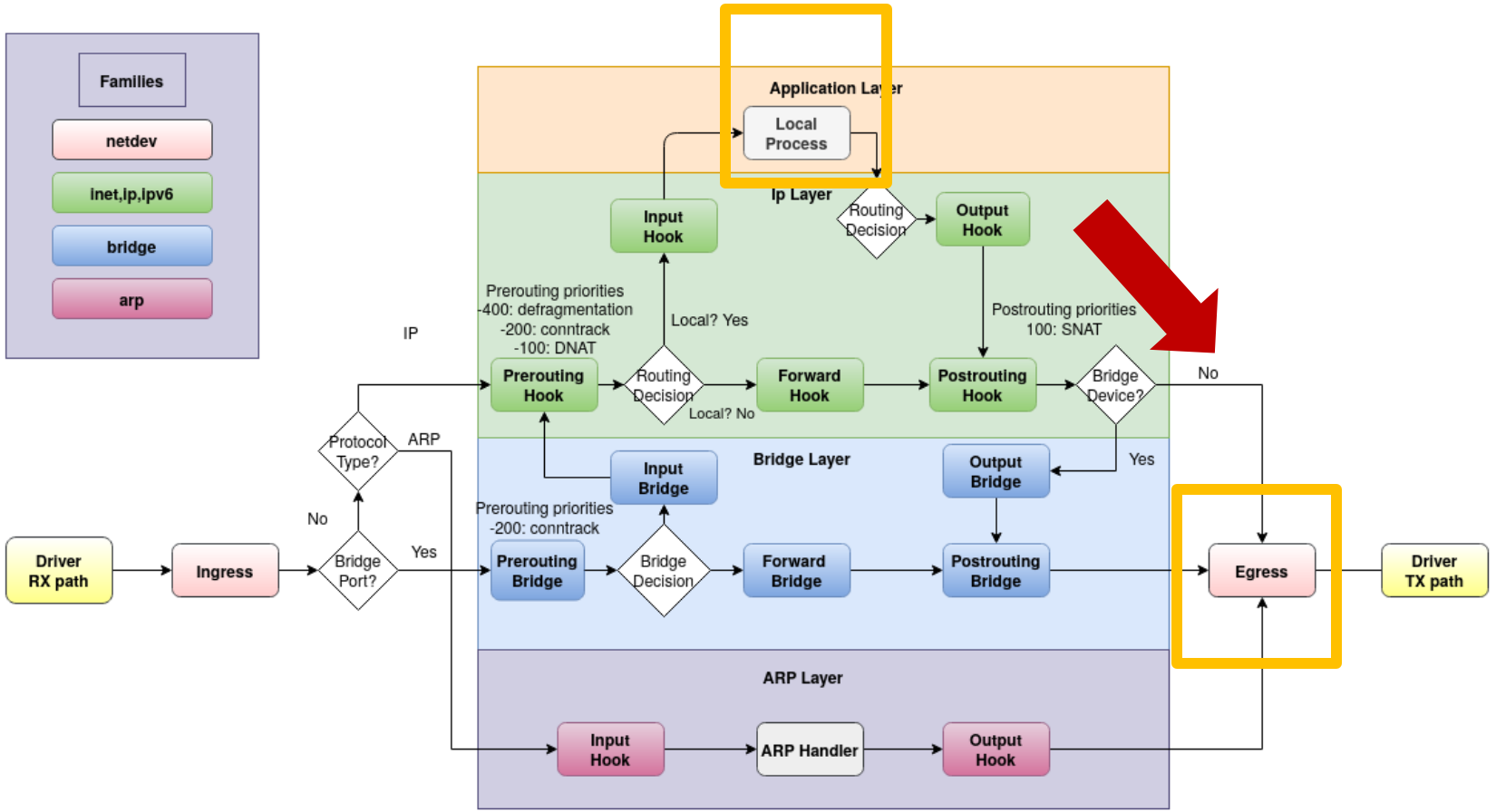
- Ingressing



https://wiki.nftables.org/wiki-nftables/index.php/Netfilter_hooks 19.12.2023 15:18

Scapy and the netfilter

- Egressing



https://wiki.nftables.org/wiki-nftables/index.php/Netfilter_hooks 19.12.2023 15:18

Scapy – Basic examples

TCP upon IP protocol with basic fields

- IP(src="1.2.3.4",dst="1.2.3.5")/TCP(dport=80, flags="S")

UDP upon IP with random sourceport

- IP(src=RandIP(), dst="1.2.3.4")/UDP(sport=RandShort(), dport=80, chksum=0xFFFF)

ICMP

- IP(dst="1.2.3.4") / ICMP(type=3, code=0)

Sending, Receiving

- Available at different layers, in loops…

Sniffing

- sniff(iface='eth3', filter = lambda s: s[TCP].flags == 18, prn = lambda x: x[IP].dst)

Bringing it all together…

```
iptables -A INPUT -p tcp -m ttl --ttl-eq 8 -m tcp --dport 1234 --tcp-flags FIN,SYN,RST,ACK SYN -j DROP
```

Craft a fitting package:

```
packet = IP(ttl=8, dst="192.168.7.2")/TCP(dport=1234, flags=0x02)
```

Send it:

```
send(packet, iface="tap0")
```

Sniff for it:

```
sniff(iface="eth0", filter="tcp and port 1234", count=1, prn=packet1_check )
```

```
def packet1_check(x)
    if x.ttl == 8 and x[TCP].flags == "S":
        print("accepted by FW")
    else:
        print("rejected by FW")
```

Bringing it all together…

Iptables – A INPUT –s 192.168.7.0/24 -I eth0 –p tcp –dport 22 –m state –state NEW,ESTABLISHED –j ACCEPT

Craft a fitting package:

```
packet = IP(src="192.168.7.0, dst="192.168.7.2") /
TCP(dport=22)
```

Send it:

```
send(packet, iface="tap0")
```

Sniff for it:

```
sniff(iface="eth0",  filter="tcp and src
192.168.7.0, count=1, prn=packet2_check )
```

```
def packet2_check(x)
   if x[TCP].dport == 22:
      print("accepted by FW")
   else:
      print("rejected by FW")
```

Bringing it all together…

Iptables –t nat –A OUTPUT –d 192.168.7.1/32 -o eth0 –p tcp --dport 100 –j REDIRECT --to-ports 60001

Craft a fitting package:

```
pkt = TCP(dport=100)
```

Send it:

```
s.setsockopt(socket.SOL_SOCKET, 25, str("eth0"))
s.bind(('192.168.7.2', 0))
s.sendto(bytes(pkt), ("192.168.7.1", 0))
```

Sniff for it:

```
sniff(iface="tap0", filter="tcp and port 60001", count="1", prn=packet3_check)
```

```
def packet3_check(x)
    if x[TCP].dport == 60001:
        print("accepted by FW")
    else:
        print("rejected by FW")
```

Test firewall rules – Demo

DEMO

Summary

Why you have to test your firewall rules

Netfilter basics

Ip(6)tables overview

Toollandscape for network testing

Scapy usage

Test iptables firewall rules

Questions?

Simone Weiß

Embedded Systems Developer, EB-EST-CMS-P-1
Elektrobit – Driving the future of software

simone.weiss@elektrobit.com
elektrobit.com

Michael Estner

Senior Software Engineer, EB-EST-CMS-P-1
Elektrobit – Driving the future of software

michael.estner@elektrobit.com
elektrobit.com

Elektrobit

February 3, 2024