# Firewalls

Yannick Chevalier
Université de Toulouse
CSA M1, Security

# Plan

# Context

## Local Area Network (LAN)

- closed network without direct access to Internet
- Most people work only in this kind of networks
- Examples : *eduroam*, ups, etc.

## Border with the Internet

- One computer is tasked with managing the interface between a LAN and :
  - a greater LAN, or
  - the Internet
- The control of the border is the task of a firewall

## Firewall description

- is part of the Operating System
- has direct access to all the communications and all the network interface cards
- issues judgements on what to do to TCP/IP-level messages

# Exercice : Separation in a Network

## Situation

A company has an *e*commerce web site. Some of its employees are connected using an ethernet network, some use wifi, and some are remotely connected. Among the employees, there are developers, administrators, and users. In addition to its locally run Web server, the company has internal servers (DNS, DB server, Business-related application servers). Finally, clients visiting the company may use the wifi to connect to the Web.

## Questions

1. List the components and their network connections. The current network is represented by a single component, a bus.
2. We want to install a firewall to protect the assets of the company. Which components must still be accessible from the outside ?
3. What happens if a hacker can take control of the Web server ?
4. Propose a separation of the components into disjoint zones (disregard wifi for the moment). How can you implement this separation ?
5. What happens when one adds the wifi network ?

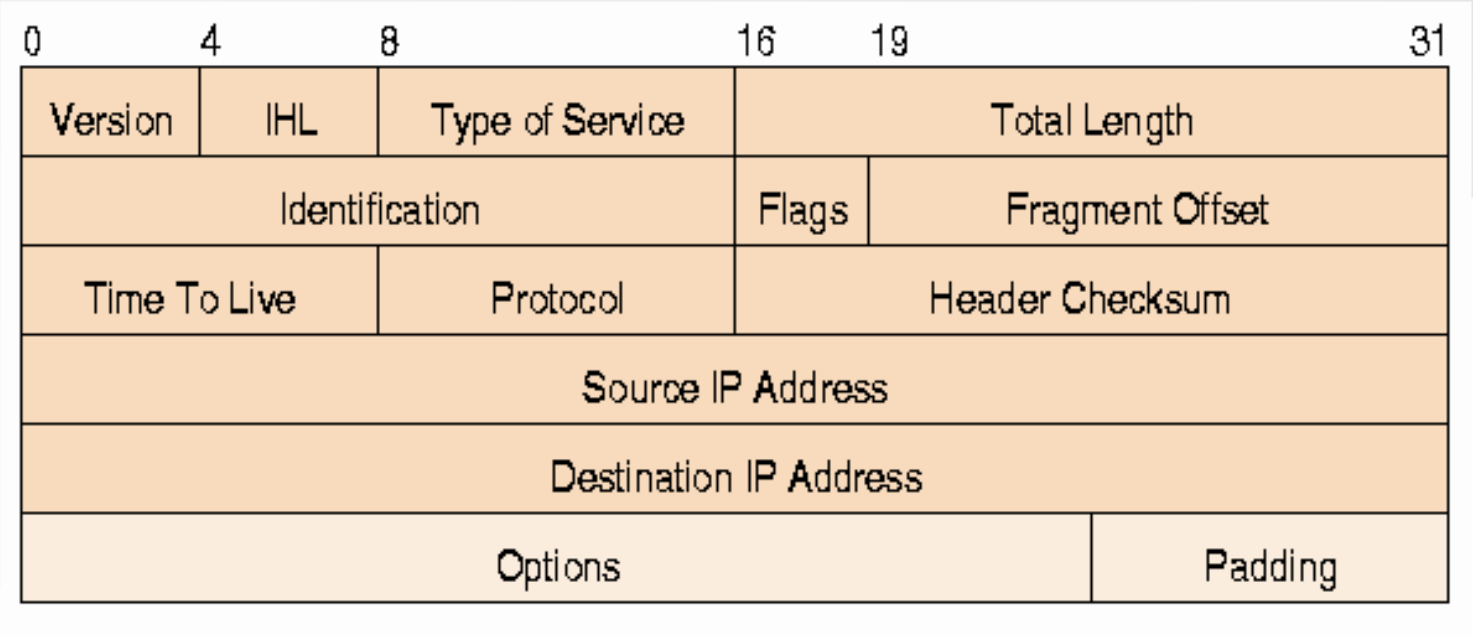
# Plan

# Outline

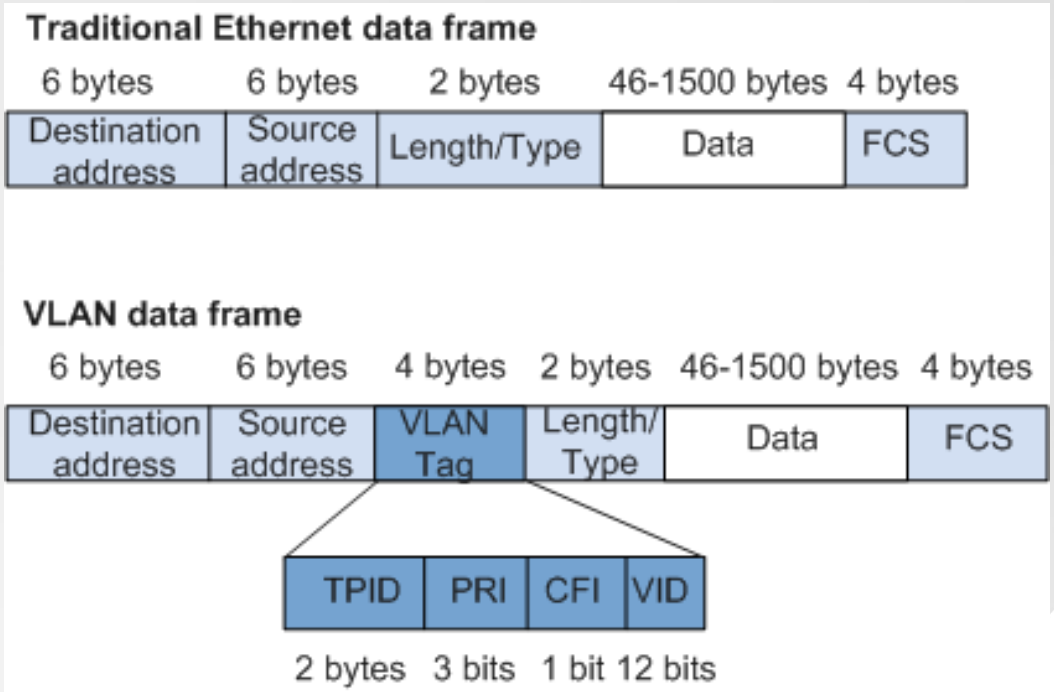
## Border Control with a Firewall

- Packet Selection
- Packet Handling
- IPTables modules

# COMMON STRUCTURE



# ADDRESSES




## IP CRASH COURSE

A packet of the IP protocol has :

- a source IP address, selected by iptables with `-s 192.168.0.3`
- a destination IP address, selected by iptables with `-d 192.168.0.3`

# Interfaces

## Physical devices

- recognized by the OS : `eth0`
- Virtual LAN (VLAN) : peripheral annotated with a mark (a number $0 \leq n < 4096$), *e.g.* `eth0.123`

## Iptables

Refers to a device with its name, and whether it :

- inputs a message : `-i eth0.20` selects the packets with the mark 20 incoming in device `eth0`
- outputs a message : `-o eth1.30` selects the packets with the mark 30 going out through the device `eth1`

# Protocol and Port

## Protocol

- **tcp**, **udp**,icmp, or other listed in (`/etc/protocols`)
- Most often one wants to select tcp : `-p tcp`

## Port

- source port or destination port
- one has to use a protocol that understands ports, like TCP
- for iptables, a port can have a name defined in `/etc/services`
- Examples : `-sport 80`, `-dport smtp`

# Position in request/response

## Underlying issue

- we often want to discard all incoming messages on an interface
- also, we still want to allow responses to a request we have accepted
- Conclusion : remember (in a state) which messages have been sent to whom, and allow the responses

## Module state/conntrack

- `-m state` or `-m conntrack` to be able to specify a state
- `-state` to say which states are selected
- marche pour tcp **et** udp

## Main states

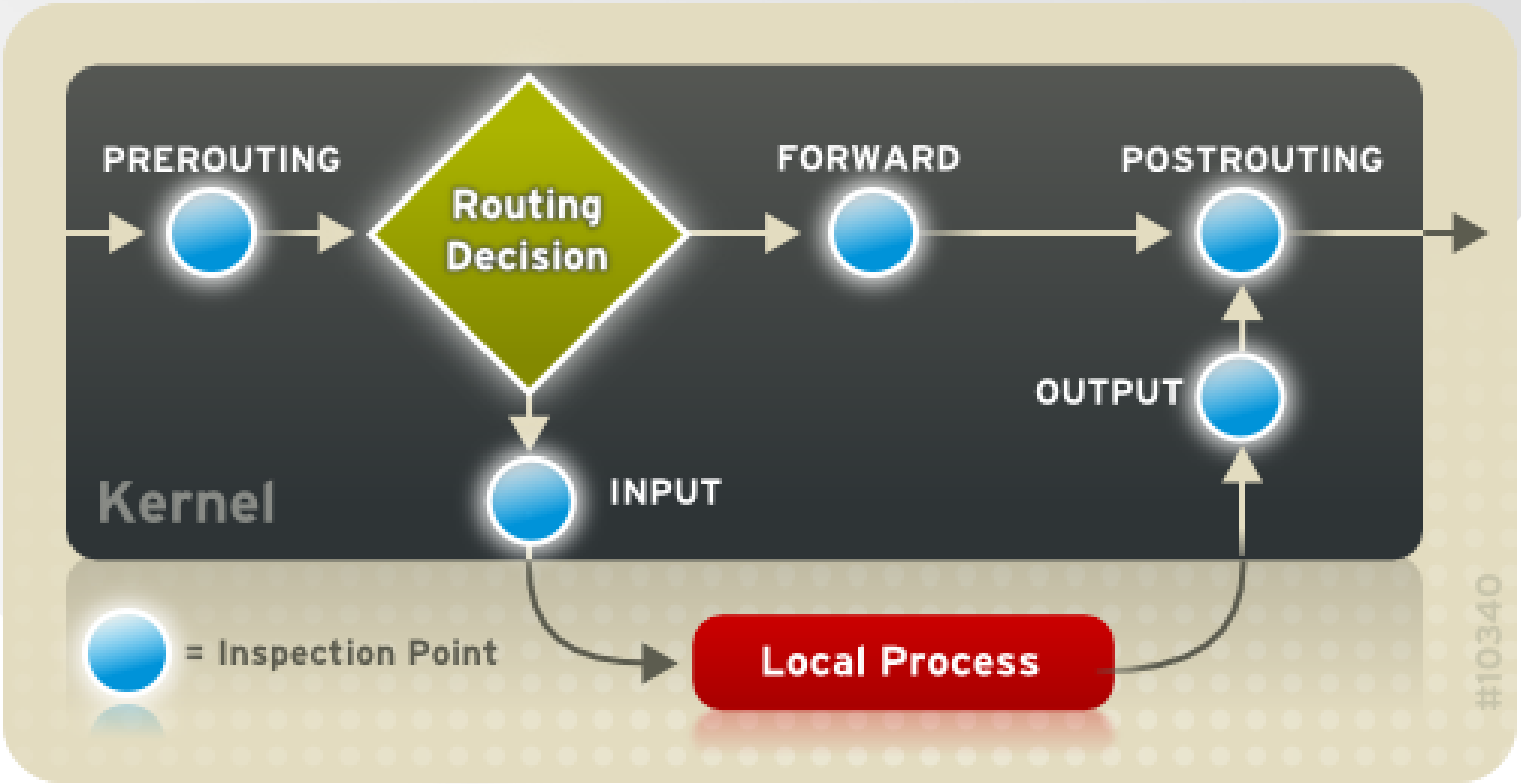
- `new` : new connection
- `established` : response to a previous message
- `related` : related to an existing previous message (e.g. when the destination port change)

# Outline

## Border Control with a Firewall

- Packet Selection
- Packet Handling
- IPTables modules

# ARCHITECTURE



# TABLES (IPTABLES)

## ORGANISATION

- Rules are organized per tables
- The default table is filter, which is employed for access control, to decide whether a packet can go through

## NON-DEFAULT TABLE

to use another table, use `-t name` where `name` is :

MANGLE : to alter packets

NAT : for connecting a LAN with a wider network

SECURITY : for Mandatory Access Control with SELinux

one can also create new tables, and send packets to these tables for closer examination

# Chains

## Chains in the filter table

A chain process a message according to its origin and destination :

INPUT chain : the message is directed to a program on this machine

OUTPUT chain : the message was sent by a program on this machine

FORWARD chain : (border) the message comes from another machine and goes to another machine

## Rules and chains

- A chain is a list of rules
- Rules can be inserted (`-I chain`) or appended (`-A chain`) to the chain
- The first rule matching a packet is applied
- Rules are usually added with `-A`

# Rules

## Rules for rules

- Each rule belongs to a chain
- Each chain belongs to a table
- We have so far seen how to decide whether a rule is applicable on a packet
- When it is, the rule issues a **judgement** on the selected packet is specified with `-j`

## Possible judgements

ACCEPT : stop the filtering and let the packet go

DROP : stop the filtering and discard the packet

REJECT : stop the filtering and inform the source address that the packet has been discarded

CHAIN NAME : the packet is sent to another chain for further processing

# Examples

## Goal

The outside is connected to `eth1` and must be prevented from using the local network (on `eth0`)

## First version

```
iptables -I FORWARD -i eth1 -o eth0 -j REJECT
```

## Second version

```
iptables -I FORWARD -m state --state NEW -i eth1 -o eth0
-j REJECT
```

## Authorize one specific connection (if not rejected before

```
FORWARD -i eth1 -d 10.0.0.2
```

# Examples

## Goal

The outside is connected to `eth1` and must be prevented from using the local network (on `eth0`)

## First version

```
iptables -I FORWARD -i eth1 -o eth0 -j REJECT
```

## Second version

```
iptables -I FORWARD -m state --state NEW -i eth1 -o eth0
-j REJECT
```

## Authorize one specific connection (if not rejected before

```
iptables -I FORWARD -i eth1 -d 10.0.0.2 -j ACCEPT
```

# Examples

## Goal

The outside is connected to `eth1` and must be prevented from using the local network (on `eth0`)

## First version

```
iptables -I FORWARD -i eth1 -o eth0 -j REJECT
```

## Second version

```
iptables -I FORWARD -m state --state NEW -i eth1 -o eth0
-j REJECT
```

## Authorize one specific connection (if not rejected before

```
iptables -I FORWARD -i eth1 -d 10.0.0.2 -j ACCEPT
```

# Examples

## Goal

The outside is connected to `eth1` and must be prevented from using the local network (on `eth0`)

## First version

```
iptables -I FORWARD -i eth1 -o eth0 -j REJECT
```

## Second version

```
iptables -I FORWARD -m state --state NEW -i eth1 -o eth0
-j REJECT
```

## Authorize one specific connection (if not rejected before

```
iptables -I FORWARD -i eth1 -d 10.0.0.2 -j ACCEPT
```

# Logging packets

## Routing to another chain

- A packet can be sent to another chain for further processing
- It can also be sent to a special chain : `LOG`

## Goal and properties

- The routing to `LOG` doesn't stop the processing in the current chain
- The kernel records the event in /var/log/syslog
- Using `-j LOG -log-prefix='[iptables] '` allows for easily finding the events in the log
- One can specify a own file for these (in /etc/rsyslog.d/00-my_iptables.conf) :

```
:msg,contains,"[iptables] " -/var/log/iptables.log
& stop
```

# The NAT Table

## LAN and NAT

- LAN use reserved addressed (defined in the IP protocol)
- These addresses are not valid (cannot be contacted) on the Internet
- Impact on Security :
  - Allow machines on the LAN to connect to the Internet
  - Forbid machines on the Internet to connect to machines in the LAN
- Employed by Eduroam, Universities, ISP, etc.
- Works on the `PREROUTING` (before `INPUT`), `OUTPUT`, and `POSTROUTING` (after `OUTPUT`) chains

## Command (Internet connected to the eth1 interface

```
iptables -t nat -A POSTROUTING -o eth1 -j MASQUERADE
```

# WRITING IPTABLES RULES

## SOLUTION FROM PREVIOUS EXERCICE

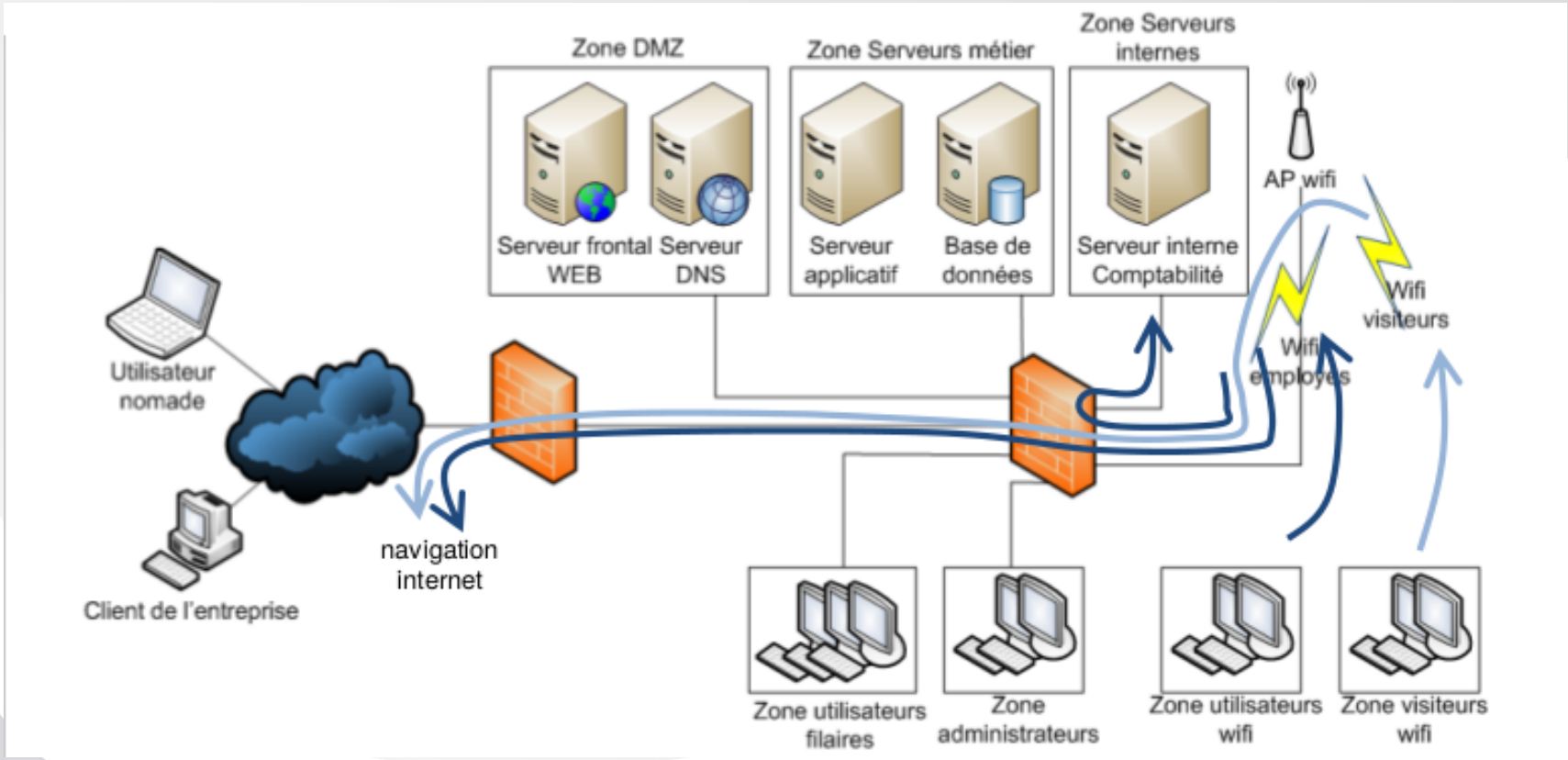



FIGURE – Network separated into different zones

# Questions (1/n)

## Preparation

Prepare a table (on paper) with 1 line and 1 column for each zone (so a $n \times n$ table). On line $i$, column $j$, select whether the communications from zone $i$ to zone $j$ should be $A$ accepted, $D$ denied, or $R$ response only. The wifi iis considered to be a unique zone. You can also specify a protocol if need be.

| | Internet | DMZ | App | internal BP | wifi | admin | Ethernet |
|---|---|---|---|---|---|---|---|
| Internet | | A,web | D | D | R | R | R |
| DMZ | R | | R | D | R | R | R |
| BP | D | A | | D | D | R | R |
| internal BP | D | D | D | | D | R | R |
| wifi | A | A | D | D | | D | D |
| admin | A | A | A | A | D | | A |
| Ethernet | A | A | A | A | D | R | |

# Questions (1/n)

## Preparation

Prepare a table (on paper) with 1 line and 1 column for each zone (so a $n \times n$ table). On line $i$, column $j$, select whether the communications from zone $i$ to zone $j$ should be $A$ accepted, $D$ denied, or $R$ response only. The wifi iis considered to be a unique zone. You can also specify a protocol if need be.

| | Internet | DMZ | App | internal BP | wifi | admin | Ethernet |
|---|---|---|---|---|---|---|---|
| Internet | | A,web | D | D | R | R | R |
| DMZ | R | | R | D | R | R | R |
| BP | D | A | | D | D | R | R |
| internal BP | D | D | D | | D | R | R |
| wifi | A | A | D | D | | D | D |
| admin | A | A | A | A | D | | A |
| Ethernet | A | A | A | A | D | R | |

# Rules

1. Separate the LAN from Internet with a NAT firewall
2. Transfert the http and https requests on port 80 (tcp protocol) and 443 (TLS) of the firewall to the Web server at LAN address `192.168.1.17`.
3. Reject all other communications from the Internet to the LAN, but for those (tcp) that are responses to LAN-issued requests

# Rules

1. Separate the LAN from Internet with a NAT firewall

```
iptables -t nat --flush
iptables -t nat -A POSTROUTING -i eth0 -o eth1 -j MASQUERAD
```

2. Transfert the http and https requests on port 80 (tcp protocol) and 443 (TLS) of the firewall to the Web server at LAN address `192.168.1.17`.
3. Reject all other communications from the Internet to the LAN, but for those (tcp) that are responses to LAN-issued requests

# Rules

1. Separate the LAN from Internet with a NAT firewall
2. Transfert the http and https requests on port 80 (tcp protocol) and 443 (TLS) of the firewall to the Web server at LAN address `192.168.1.17`.

```
iptables -t nat -A PREROUTING -p tcp --dport 80 \
-j DNAT --to 172.31.0.23:80
iptables -t nat -A PREROUTING -p tcp --dport 443 \
-j DNAT --to 172.31.0.23:80
```

3. Reject all other communications from the Internet to the LAN, but for those (tcp) that are responses to LAN-issued requests

# Rules

1. Separate the LAN from Internet with a NAT firewall
2. Transfert the http and https requests on port 80 (tcp protocol) and 443 (TLS) of the firewall to the Web server at LAN address `192.168.1.17`.
3. Reject all other communications from the Internet to the LAN, but for those (tcp) that are responses to LAN-issued requests

```
# Accept redirections to the Web server wherever they come
iptables -A FORWARD -p tcp --dport 80 -d 172.31.0.23 -j ac
# Default rules
iptables -A FORWARD -i eth1 -o eth0 -m state --state NEW -
iptables -A FORWARD -i eth1 -o eth0 -m state \
 --state ESTABLISHED,RELATED -j ACCEPT
iptables -A FORWARD -i eth0 -o eth1 -j ACCEPT
```

# Outline

## Border Control with a Firewall

- Packet Selection
- Packet Handling
- IPTables modules

# Modules

## Context

- iptables also has a number of modules that can be employed for a finer grained access control
- they are automatically loaded when used
- works like the `-m state` (m is for module)

## Examples (-m name)

conntrack : control according to the position in a tcp session

owner : Allow to control the output according to the user running the application that sends the packet

limits : limits the number of packets per second

recent : allows to limit the number of packets per origin (for DoS attacks)

# Plan

# IPTABLES SUMMARY

- Tables : filter, nat, mangle, ...
- Hooks : input, output, forward, ...

Each table has a subset of these

- Chains : DAG of rules

attached to hooks

- Rules : set of matches and a judgement/target

iptables -A INPUT -s 2.2/16 -d 3/8 -p udp -dport 25 -j DROP

# Internal representation

## Internal storage

- Rules are represented by structures, with a flexible array for matches
- Chains represented by structures, with flexible array for rules
- Rules are appended at the end of the array

## Rule compilation and evaluation

- Iptables compiles the set of rules into a structure in a `.o` object file, and loads that file into the kernel on a hook
- The kernel reads these rules one after the other

## Consequence

- The filtering time is linear *wrt* the number of rules
- No way to jump to another case (*e.g.* if the protocol is udp, ignore all tcp rules)

# User mode helper (Linux)

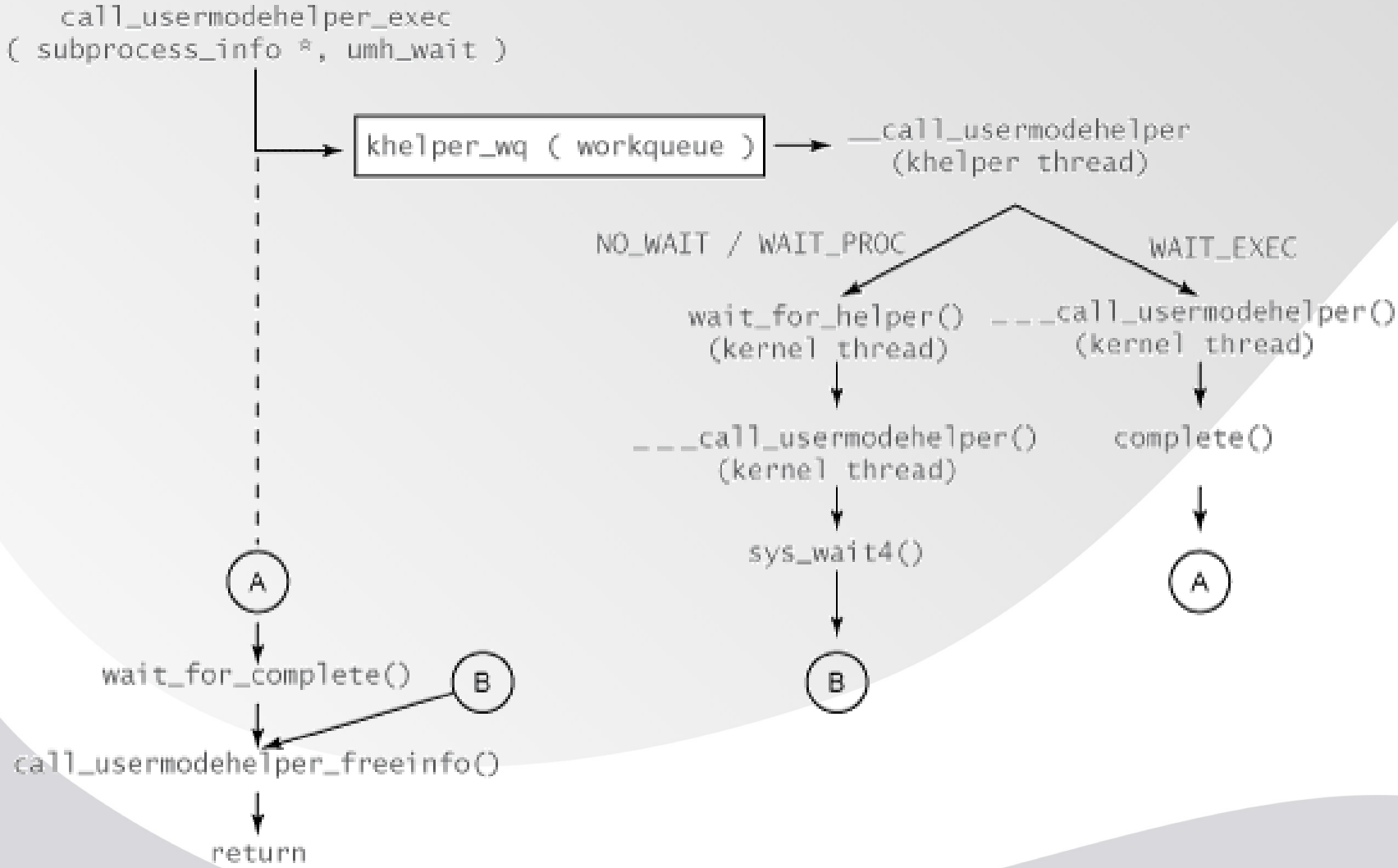
## Overview

- Userspace process : called by the kernel upon some condition, with a return code processed by the kernel
- Communication channel : session key for shared memory, pipe (kernel $\rightarrow$ process, for core dumps)
- IPC

## Example usage

- kernel detects a device is plugged in
- kernel launches a userspace process (modprobe) to load the driver module for that device
- Examples : code to put in a kernel module that will call an external application

# UMH IN THE KERNEL



# UMH IN USER SPACE (FULL)

```
static int umh_test( void )
{
   struct subprocess_info *sub_info;
   char *argv[] = { "/usr/bin/logger", "help!", NULL };
   static char *envp[] = {
         "HOME=/",
         "TERM=linux",
         "PATH=/sbin:/bin:/usr/sbin:/usr/bin", NULL };

   sub_info = call_usermodehelper_setup(
                  argv[0], argv, envp, GFP_ATOMIC );
   if (sub_info == NULL) return -ENOMEM;

   return call_usermodehelper_exec(
                  sub_info, UMH_WAIT_PROC );
}
```

# UMH IN USER SPACE (EASY)

```
static int umh_test( void )
{
   char *argv[] = { "/usr/bin/logger", "help!", NULL };
   static char *envp[] = {
         "HOME=/",
         "TERM=linux",
         "PATH=/sbin:/bin:/usr/sbin:/usr/bin", NULL };

   return call_usermodehelper(
                  argv[0], argv, envp, UMH_WAIT_PROC );
}
```

# UMH for BPFilter

## Overview

- Like loading modules with modproble, loads firewall code instead
- Same structures for compatibility with iptables
- eBPF programs are compiled and inserted into hooks by userspace programs
- Code optimisation by userspace programs (tree instead of list)
- Generic linux kernel infrastructure : eBPF

# Classic BPF

## BPF

- **hooks** on the kernel network processing functions
- BPF code is added to these hooks (on function call or return)

## Modus Operandi

- Create a `RAW` socket (needs to be root)
- Add a filter with setsockopt

```
sock = socket(PF_PACKET, SOCK_RAW, htons(ETH_P_ALL))
...
setsockopt(sock, SOL_SOCKET, SO_ATTACH_FILTER, ...)
```

## Usage

- iptables
- libpcap (at ethernet/layer 2 level)
- tcpdump, wireshark

# Extended BPF

## Extension

- Possibility to attach BPF to any kernel function
- More generic code (some loops are possible)
- Only iptables does not use it, actually

## Applications

- Add code to system calls to monitor applications
- Traffic shaping/Traffic control
- Sandbox applications
- Sandbox protocols : `xt_bpf` iptables module

Recent development, no bounds in sight

# The eBPF Revolution

https://ebpf.io/

## Extended BPF

- More generic language (BPF now called classic BPF)
- Bounded termination of programs has to be proved
- Allows *e.g.* implementation of cryptographic protocols within the kernel

  Falco (monitor), Cilium (Security within the stack for Kubernetes)

## More hooks

- Hooks are now on all system calls
- Allows control, monitoring, and tracing of all applications

  Katran (traffic control), bpftrace, ply (for embedded systems)