MASTER OF SCIENCE
IN ENGINEERING

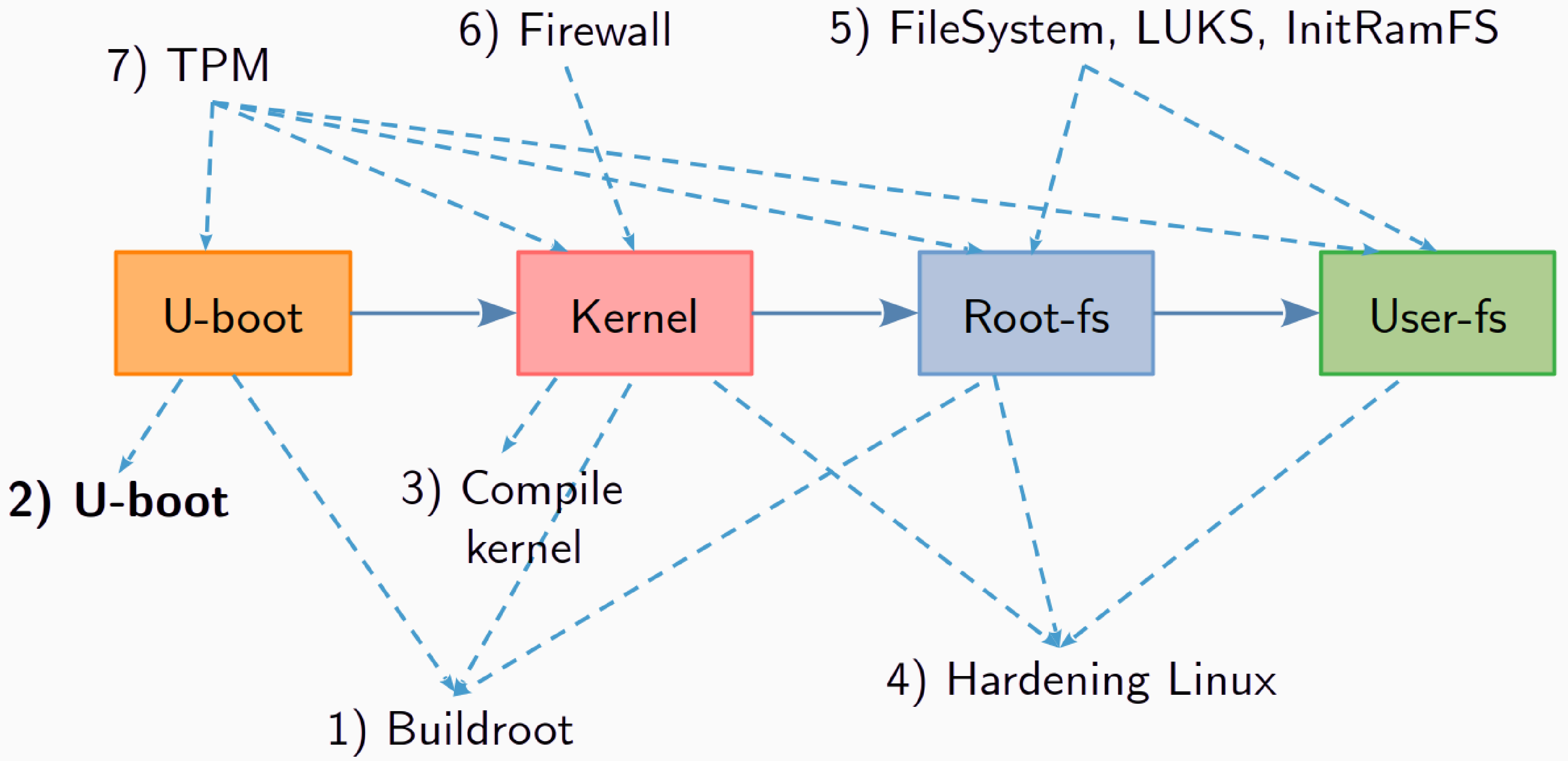
netfilter and iptables

Luca Haab, Jean-Roland Schuler

November 24, 2025

swissuniversities

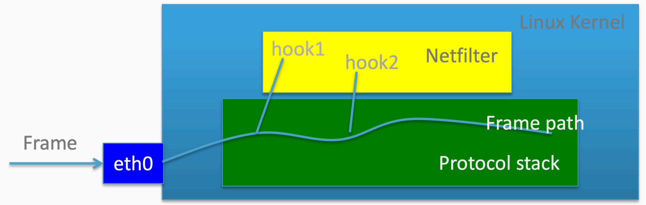
Overview


7) TPM
6) Firewall
5) FileSystem, LUKS, InitRamFS
U-boot
Kernel
Root-fs
User-fs
2) U-boot
3) Compile kernel
4) Hardening Linux
1) Buildroot

Introduction

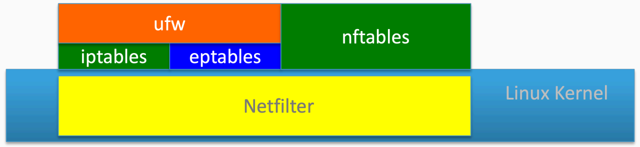
What is netfilter?

- `netfilter` is a framework for packet mangling, outside the normal Berkeley socket interface.
- each protocol defines “hooks” (chain-table) which are well-defined points in a packet’s traversal of that protocol stack. At each of these points, the protocol will call the netfilter framework with the packet and the hook number.



iptables/ufw/eptables/... what are all these really?

- iptables is a user-space application program that allows to configure the tables provided by the Linux kernel firewall (Netfilter)
- ebtables is a user-space program that allows to configure Netfilter only for the **layer 2 of the OSI model** (Linux bridge)
- The Uncomplicated Firewall (ufw) is a frontend for iptables and is particularly well-suited for host-based firewalls.
- nftables is the project that aims to replace the existing {ip,ip6,arp,eb}tables framework.



swissuniversities

iptables Main Features

- stateless packet filtering (IPv4 and IPv6)
- stateful packet filtering (IPv4 and IPv6)
- all kinds of network address and port translation, e.g. NAT (IPv4 and IPv6)
- flexible and extensible infrastructure
- multiple layers of API's for 3rd party extensions
- fix a bug
- ...

What can one do with `iptables`?

- Build internet firewalls based on stateless and stateful packet - filtering
- Deploy highly available stateless and stateful firewall clusters
- Use NAT and masquerading for sharing internet access if you don't - have enough public IP addresses
- Use NAT to implement transparent proxies
- Support the `tc` and `iproute2` systems used to build sophisticated QoS - and policy routers
- Do further packet manipulation (*mangling*) like altering the *TOS* / *DSCP*/ *ECN* bits of the *IP* header
- Receiving in user-space queued packets from the kernel - nfnetlink_queue subsystem
- ...

netfilter hooks

There are **five netfilter hooks**. As packets progress through the kernel stack, they will trigger kernel modules having registered with these hooks. Which ones will be triggered depend on whether the packet is incoming/ outgoing, its destination, and whether the packet was dropped or rejected at a previous point.

The following hooks represent these well-defined points in the networking stack:

- `NF_IP_PRE_ROUTING`: triggered by any incoming packet after entering the network stack
- `NF_IP_LOCAL_IN`: triggered after an incoming packet has been routed if the packet is destined for the local system
- `NF_IP_FORWARD`: triggered after an incoming packet has been routed if the packet is to be forwarded to another host
- `NF_IP_LOCAL_OUT`: triggered by any locally created outbound traffic as soon as it enters the network stack
- `NF_IP_POST_ROUTING`: triggered by any outgoing or forwarded traffic after routing has taken place and just before being sent out

netfilter in-depth

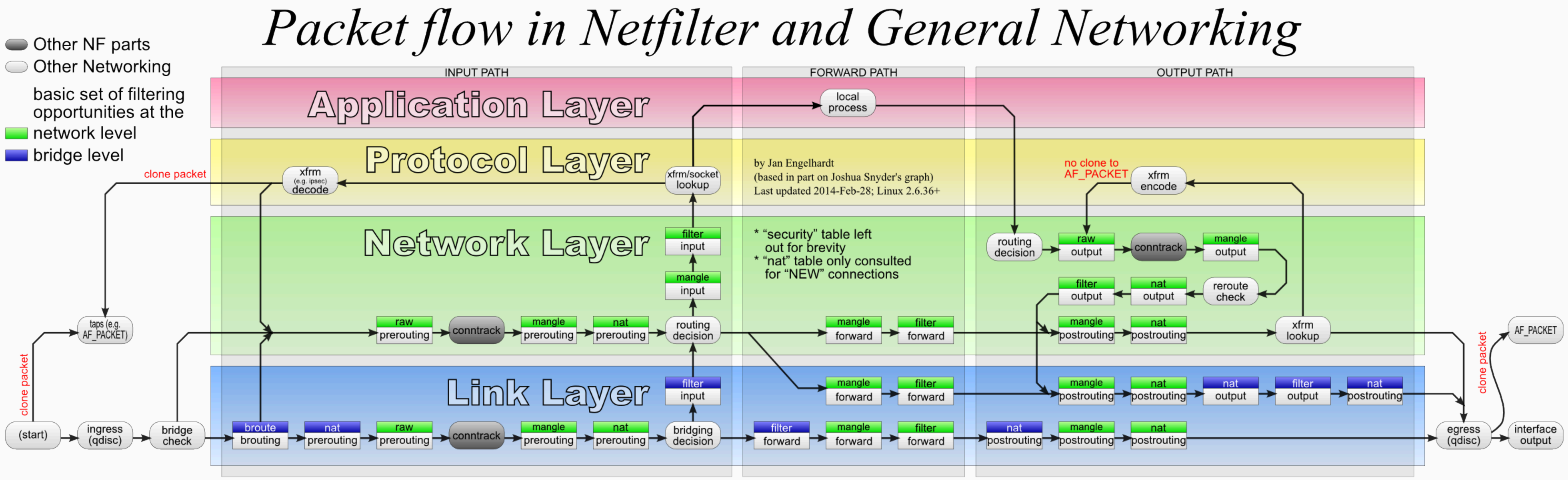
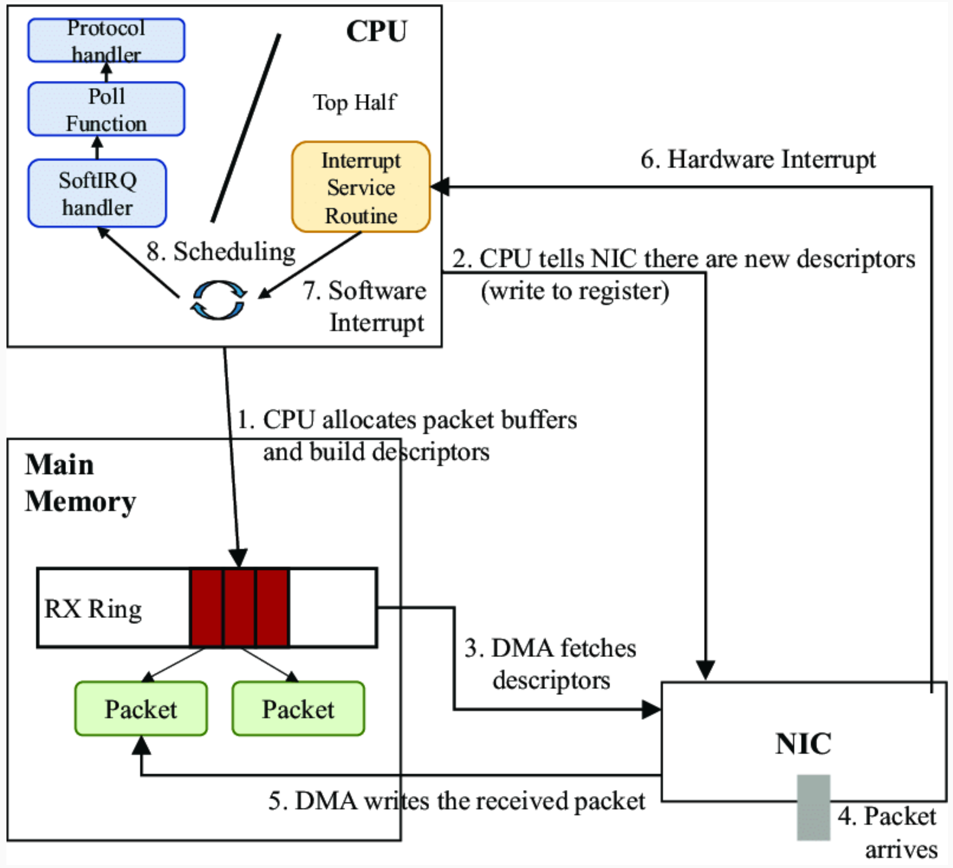


Figure source: https://upload.wikimedia.org/wikipedia/commons/3/37/Netfilter-packet-flow.svg (Author: J. Engelhardt)

netfilter packet processing (I)

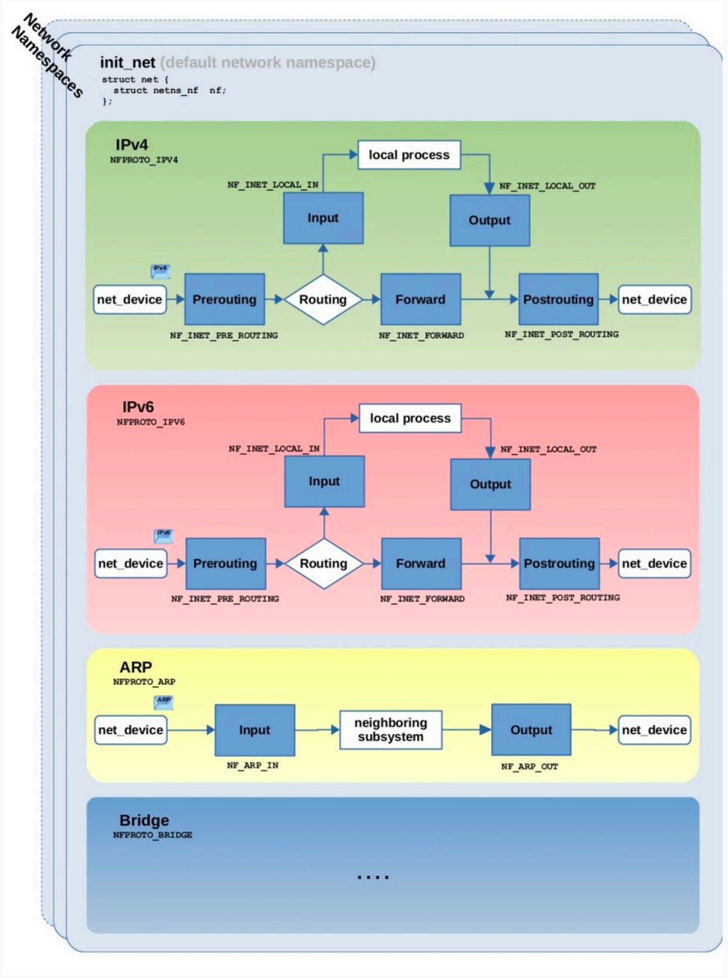
At the physical-network layer, the NIC writes the received packet to a ring buffer in memory via DMA. Afterwards, the OS kernel calls `__skb_dequeue` to add the packet to the processing queue of the corresponding device and converts it into a `sk_buffer` type (i.e. socket buffer - the `__skb_dequeue`). The packets can thus be considered as `sk_buffer` and finally the `netif_receive_skb` function is called to sort the packets by protocol type and jump to the corresponding processing function.

Source: https://www.sobyte.net/post/2022-04/understanding-netfilter-and-iptables/ (Author: So Byte)


CPU
Protocol handler
Poll Function
SoftIRQ handler
Top Half
Interrupt Service Routine
8. Scheduling
7. Software Interrupt
6. Hardware Interrupt
2. CPU tells NIC there are new descriptors (write to register)
1. CPU allocates packet buffers and build descriptors
Main Memory
RX Ring
Packet
Packet
3. DMA fetches descriptors
NIC
5. DMA writes the received packet
4. Packet arrives

netfilter packet processing (II)

For different protocols (IPv4, IPv6, ARP etc.), the Linux kernel network stack triggers the corresponding hooks at predefined locations along the packet processing path of the protocol stack. On the image the trigger points in the different protocol processing flows and the corresponding hook names (in **bold**) can be seen on the picture.

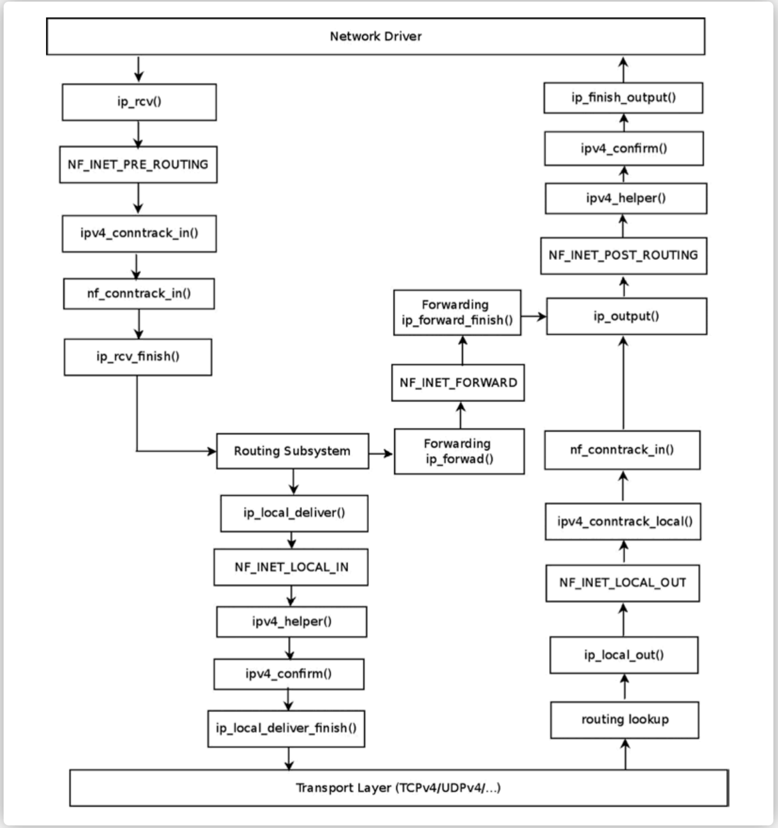


netfilter packet processing (III)

The so-called hooks are essentially enumerated objects in the code (integers with values incrementing from 0).

```
enum nf_inet_hooks {
NF_INET_PRE_ROUTING,
NF_INET_LOCAL_IN,
NF_INET_FORWARD,
NF_INET_LOCAL_OUT,
NF_INET_POST_ROUTING,
NF_INET_NUMHOOKS
};
```

Each hook corresponds to a specific trigger point location in the kernel network stack. On the right, for example, the IPv4 netfilter hooks definition can be observed.



netfilter packet processing (IV)

The macro NF_HOOK (see code on the right) is called uniformly for all trigger point locations to trigger the hook. The parameters received by NF-HOOK are as follows:

- pf: protocol family of the packet, NFPROTO_IPV4 for IPv4.
- hook: netfilter hook enumeration object shown in the above figure, such as NF_INET_PRE_ROUTING or NF_INET_LOCAL_OUT.
- skb: SKB object indicating the packet being processed.
- in: input network device for the packet.
- out: output network device for the packet.
- okfn: pointer to a function that will be called when this hook is about to terminate, usually passed to the next processing function on the packet processing path.

Info

You can check the details on your installationunder BUILDROOT_ROOT/output/build/linux-6.3.6/net/.

```
static inline int NF_HOOK(uint8_t pf, unsigned int
hook, struct sk_buff *skb,
    struct net_device *in, struct net_device *out,
    int (*okfn)(struct sk_buff *))
{
    return NF_HOOK_THRESH(pf, hook, skb, in, out,
okfn, INT_MIN);
}
```

The return value of NF_HOOK is one of the following netfilter vectors with a specific meaning.

- NF_ACCEPT: Continue normally on the processing path (actually the last incoming okfn is executed in NF_HOOK).
- NF_DROP: Discard the packet and terminate processing.
- NF_STOLEN: Packet forwarded, terminate processing.
- NF_QUEUE: Queue the packet for other processing.
- NF_REPEAT: Re-call the current hook.

`iptables`: Tables and Chains

The `iptables` firewall uses *tables* to organize its rules. These *tables* classify rules according to the type of decisions they are used to make. For instance, if a rule deals with *Network Address Translation*, it will be put into the *nat* table. If a different rule is used to decide whether to allow the packet to continue to its destination, it likely resides in the *filter* table.

Within each `iptables` *table*, rules are further organized within separate *chains*. While *tables* are defined by the general aim of the rules they hold, the built-in chains represent the `netfilter` hooks which trigger them. *Chains* determine when rules will be evaluated.

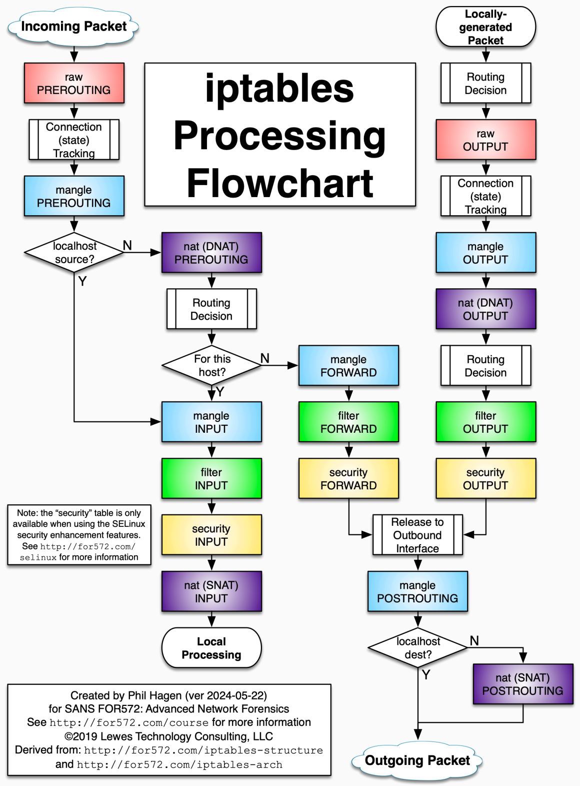
iptables: Chain

- When a packet arrives, it traverses different chains. Theses chains are contained in different tables.
- The combination chain-table are the hooks
  - Chains: INPUT, OUTPUT, FORWARD, PREROUTING, POSTROUTING
  - Tables: raw, mangle, nat, filter

Chain explanation:

```
PREROUTING:  Triggered by the NF_IP_PRE_ROUTING hook. Used typically by NAT.
INPUT:       Triggered by the NF_IP_LOCAL_IN hook. The packet is for the
             local host.
FORWARD:     Triggered by the NF_IP_FORWARD hook. The packet arrives to an
             interface (e.g. eth0) and it is forwarded to another interface
             (e.g. eth1).
OUTPUT:      Triggered by the NF_IP_LOCAL_OUT hook. The packet is sent from
             the local host.
POSTROUTING: Triggered by the NF_IP_POST_ROUTING hook. Used typically by NAT.
```

Figure source: https://stuffphilwrites.com/wp-content/uploads/2024/05/FW-IDS-iptables-Flowchart-v2024-05-22.png (Author: P. Hagen)



Note: the "security" table is only available when using the SELinux security enhancement features. See http://for572.com/selinux for more information

Created by Phil Hagen (ver 2024-05-22)
for SANS FOR572: Advanced Network Forensics
See http://for572.com/course for more information
©2019 Lewes Technology Consulting, LLC
Derived from: http://for572.com/iptables-structure
and http://for572.com/iptables-arch

iptables: filter *table*

This table is mainly used for the firewalls.

This is the default table (if no `-t` option is passed). It contains the built-in chains

- INPUT (for packets destined to local sockets),
- FORWARD (for packets being routed through the box),
- OUTPUT (for locally-generated packets).

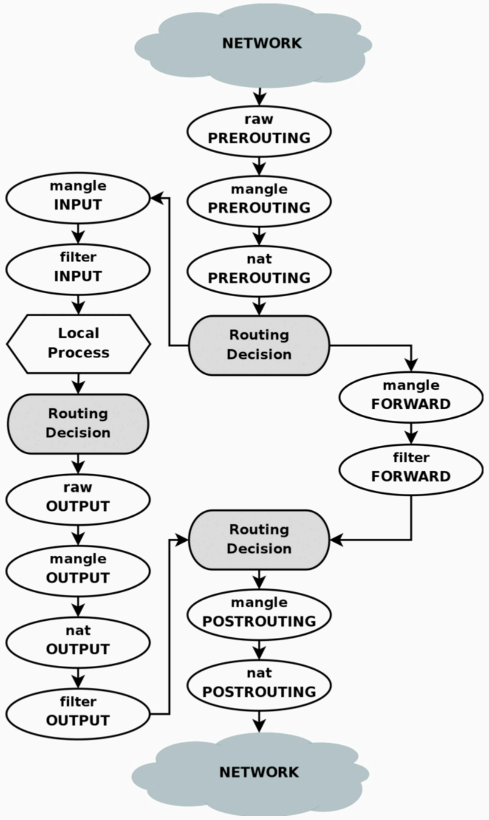


Figure source: https://commons.wikimedia.org/wiki/File:Tables_traverse.jpg (Author: O. Andreasson)

iptables: nat *table*

This table is consulted when a packet that creates a new connection is encountered. This table is consulted by the NAT (Network Address Translation) process. It consists of three built-ins chains:

- PREROUTING (for altering packets as soon as they come in),
- OUTPUT (for altering locally-generated packets before routing),
- POSTROUTING (for altering packets as they are about to go out).

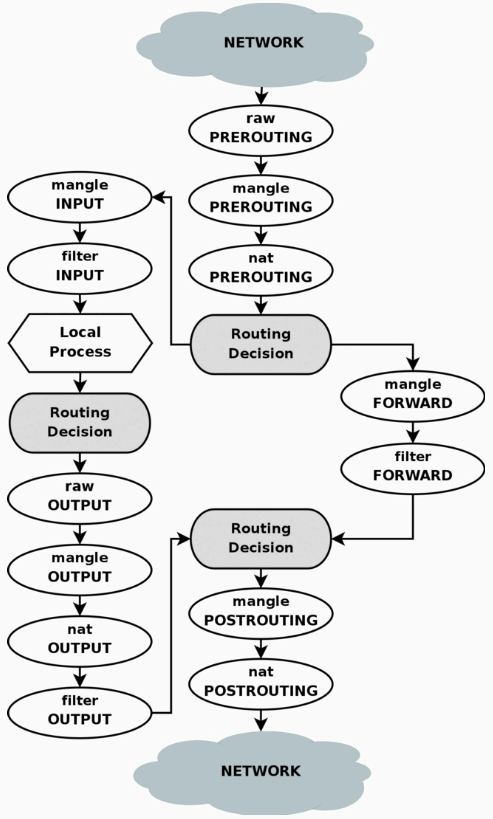


Figure source: https://commons.wikimedia.org/wiki/File:Tables_traverse.jpg (Author: O. Andreasson)

iptables: mangle *table*

This table is used for specialized packet alteration-modification. Example modify the TOS value (Type of Service) It consists of five built-ins chains

- INPUT (for packets coming into the box itself),
- FORWARD (for altering packets being routed through the box),
- POSTROUTING (for altering packets as they are about to go out),
- PREROUTING (for altering incoming packets before routing)
- OUTPUT (for altering locally-generated packets before routing)

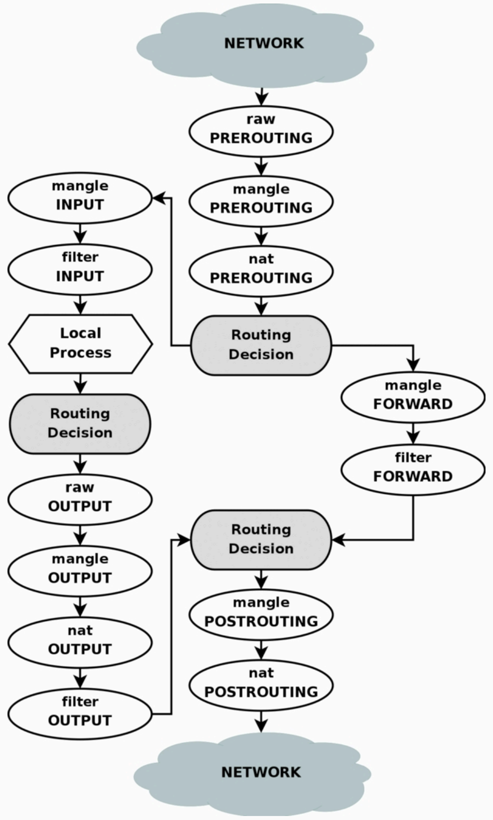


Figure source: https://commons.wikimedia.org/wiki/File:Tables_traverse.jpg (Author: O. Andreasson)

iptables: what a command looks like

An iptables command has this general form:

```
iptables -t table -COMMAND chain … -j TARGET
where
-t table:
table=FILTER, MANGLE, NAT, …    // FILTER is the default table
-COMMAND chain:
COMMAND: A=append, I=insert, D=delete, C=check, R=replace, L=list, F=flush, P=policy, ..
Chain: INPUT, OUTPUT, FORWARD
-j TARGET:
TARGET: ACCEPT, DROP, REJECT, NFQUEUE
```

swissuniversities

iptables: saving (and restoring) a configuration

Save the firewall rules:

```
iptables-save > /etc/iptables.rules

cat /etc/iptables.rules
# Generated by iptables-save v1.8.8
*filter
:INPUT ACCEPT [0:0]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [0:0]
-A INPUT -p tcp --dport 80 -j ACCEPT
COMMIT
```

Configuration on startup:

```
cat /etc/network/interface
auto eth0
iface eth0 inet static
    pre-up iptables-restore    < /etc/iptables.rules
    post-down iptables-restore < /etc/iptables.down
    address 192.168.0.14
    netmask 255.255.255.0
    gateway 192.168.0.4
```

`iptables`: stateless vs. stateful *firewall*

Stateless firewall :

> Stateless Packet Filtering: Stateless packet filters block according to content-neutral rules, e.g., blocking all inbound connections or outbound connections on certain ports, protocols, or network-layer addresses. For example, blocking outbound connections to port 25
>
> — IETF RFC7754

Stateful firewall :

> Stateful Packet Filtering: More advanced configurations require keeping state used to enforce flow-based policies, e.g., blocking inbound traffic for flows that have not been established.
>
> — IETF RFC7754

`iptables`: stateless vs. stateful - pros

Stateless firewall :

+ Efficient
+ Simple to set-up and administer as they rely on fixed set of filtering rules, such as allowing or blocking packets based on IP addresses, ports, or protocols.
+ Less vulnerable (as they do not need to recognize behaviours).

Stateful firewall :

+ Monitors the entire session for the state of the connection, while also checking IP addresses and payloads for more thorough security
+ Offers a high degree of control over what content is let in or out of the network
+ Does not need to open numerous ports to allow traffic in or out
+ Delivers substantive logging capabilities

iptables: stateless vs. stateful - cons

Stateless firewall :

- Little inspection/behaviour capabilities
- Hard to scale as the number of connections to your network increases, so does the number of rules in your firewall.
- Require initial effort to make them work correctly as settings require careful configuration to ensure that legitimate traffic is allowed through while malicious traffic is blocked.

Stateful firewall :

- Resource-intensive and interferes with the speed of network communications
- More expensive than other firewall options
- Doesn't provide authentication capabilities to validate traffic sources are not spoofed
- Doesn't work with asymmetric routing (opposite directions use different paths)
- Can lead to unexpected disconnections or half-open connections if connections - are idle for longer than the time-out

Info

Check IETF RFC 2647 and RFC 9099 (and others) for more information.

iptables: stateless *firewall*

The stateless packet firewall use the `FILTER` table

There are mainly three types of operations:

- `ACCEPT`: Accept the packet
- `DROP`: Discard the packet silently
- `REJECT`: Discard the packet and inform the source with an `ICMP` message

All the rules are consulted until the first rule matching the packet is located If no rules match the packet, the kernel looks at the chain policy (default: `INPUT-OUTPUT-FORWARD`: `ACCEPT`)

Info

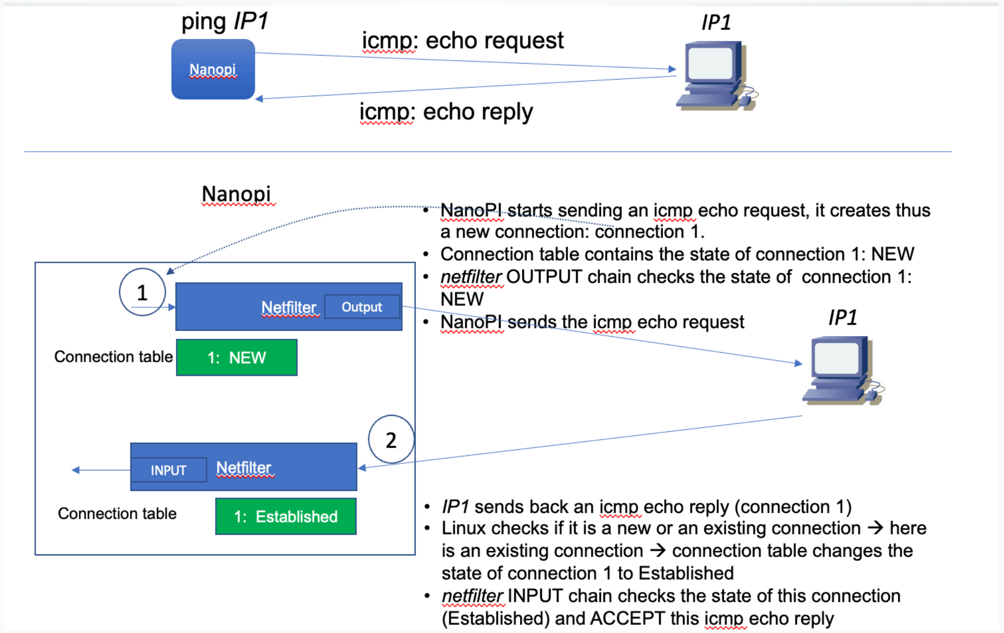
`ICMP`: Internet Control Message Protocol - IETF RFC 792 (partially superseded and amended by RFC 950, RFC 4884, RFC 6633 and RFC 6918)

iptables: stateful *firewall* (I)

The stateful packet firewall makes use of behavioural characteristics to keep track of the connection states.

There are mainly three types of states that are attempted to track:

- `NEW`: when a packet arrives that is not associated with an existing connection, but is not invalid as a first packet, a new connection will be added to the system with this label.
- `ESTABLISHED`: a connection is changed from NEW to `ESTABLISHED` when it receives a valid response in the opposite direction.
- `RELATED`: packets that are not part of an existing connection but are associated with a connection already in the system are labeled `RELATED`. This could mean a helper connection, as is the case with `FTP` data transmission connections, or it could be `ICMP` responses to connection attempts by other protocols.

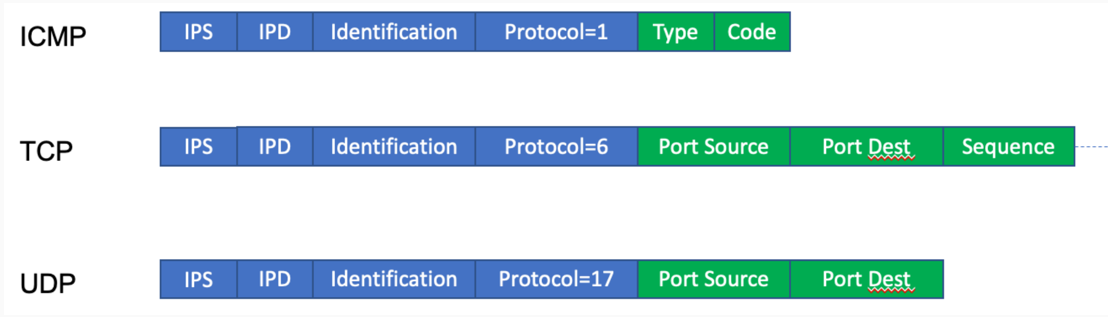


- NanoPI starts sending an icmp echo request, it creates thus a new connection: connection 1.
- Connection table contains the state of connection 1: NEW
- *netfilter* OUTPUT chain checks the state of connection 1: NEW
- NanoPI sends the icmp echo request

- *IP1* sends back an icmp echo reply (connection 1)
- Linux checks if it is a new or an existing connection → here is an existing connection → connection table changes the state of connection 1 to Established
- *netfilter* INPUT chain checks the state of this connection (Established) and ACCEPT this icmp echo reply

iptables: stateful *firewall* (II)

Packet inspection capabilities depend heavily on the availibility of the information. In the case of *TCP*, both the *3-way Handshake* and *4-way Disconnect* are used for tracking the states. However, this is easily done with this protocol but less so with others. In the picture on the right, one can see that other pieces of logic are used for tracking *"a connection"*.



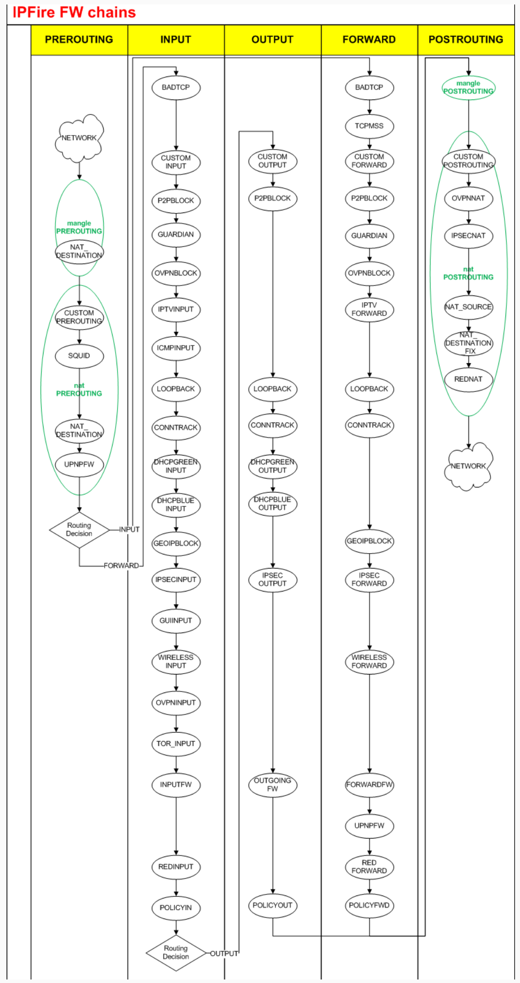
A real *firewall*

A real stateful packet firewall makes use of many more *tables* and *chains* than those we see in the present course.

On the right, the *chains* of https://www.ipfire.org/, an *open source firewall* can be seen.

Info

In fact, the real implementation of IPFire_ is even more complex if one issues the `iptables -L` command.



Resources

- `iptables` documentation: https://www.netfilter.org/documentation/
- *Digital Ocean*'s "A Deep Dive into Iptables and Netfilter Architecture" https://www.digitalocean.com/community/tutorials/a-deep-dive-into-iptables-and-netfilter-architecture
- *So Byte*'s "In-depth understanding of netfilter and iptables" https://www.sobyte.net/post/2022-04/understanding-netfilter-and-iptables/
- *Wikipedia* "Stateful firewall" : https://en.wikipedia.org/wiki/Stateful_firewall
- Various IETF RFC : RFC 792, RFC 950, RFC 2647, RFC 4884, RFC 6633, RFC 6918, RFC7754 and RFC 9099