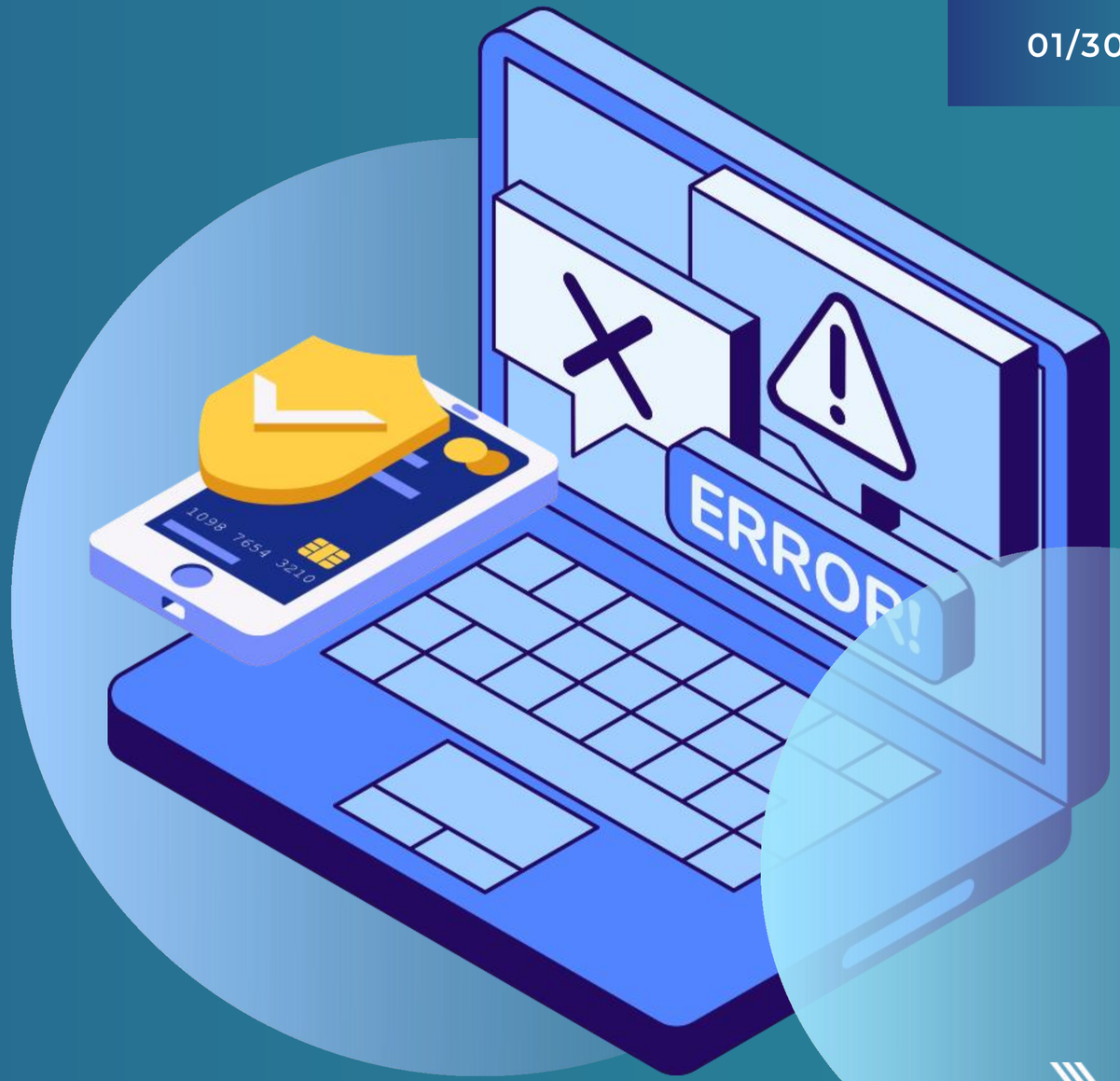
CYBER SECURITY

INTRUSION DETECTION AND NETWORK SECURITY



What is firewall ??

FIREWALL

INTRODUCTION TO FIREWALL

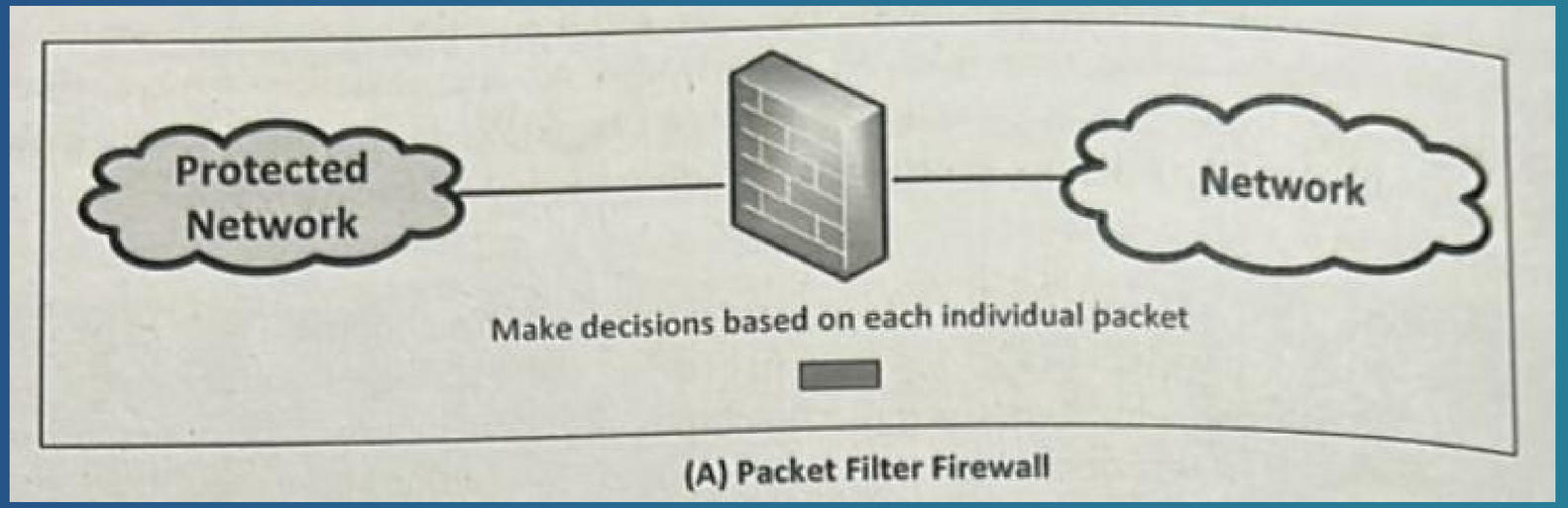
A firewall is a security system designed to stop unauthorized traffic from entering or leaving a computer network. It acts as a barrier between trusted and untrusted networks and helps protect data and systems from attacks. Firewalls can be implemented in both hardware and software forms, and their main functions include filtering data, redirecting traffic, and preventing network attacks. A good firewall ensures that all communication between two trusted zones passes through it, only allows authorized traffic as per the security policy, and remains secure from penetration itself.

A firewall's effectiveness depends on the rules or policies it enforces. These policies define how traffic is controlled within the network. The controls include **user control** (restricting access based on user roles), **service control** (based on type of service or address), and **direction control** (controlling data flow direction— inbound or outbound). When network packets pass through a firewall, they can be accepted, denied, or rejected depending on the rules. Firewalls also perform **ingress filtering** (inspecting incoming traffic) and **egress filtering** (inspecting outgoing traffic) to protect the network from external and internal threats.

TYPES OF FIREWALL

PACKET FILTER

A **packet filter** is a basic type of firewall that controls network traffic by checking information in each packet's header. It looks at details such as the source address, destination address, port number, and protocol to decide whether to allow or block the packet. However, it does not check the actual content or data inside the packet. Packet filters make decisions quickly because they do not keep track of ongoing connections or the state of the traffic. This is why they are also called stateless firewalls. The main advantage of a packet filter is its speed and efficiency, as it can inspect and control large amounts of traffic with minimal delay. Although simple, it provides a good first level of security by filtering unwanted or unauthorized network traffic before it reaches the internal system.

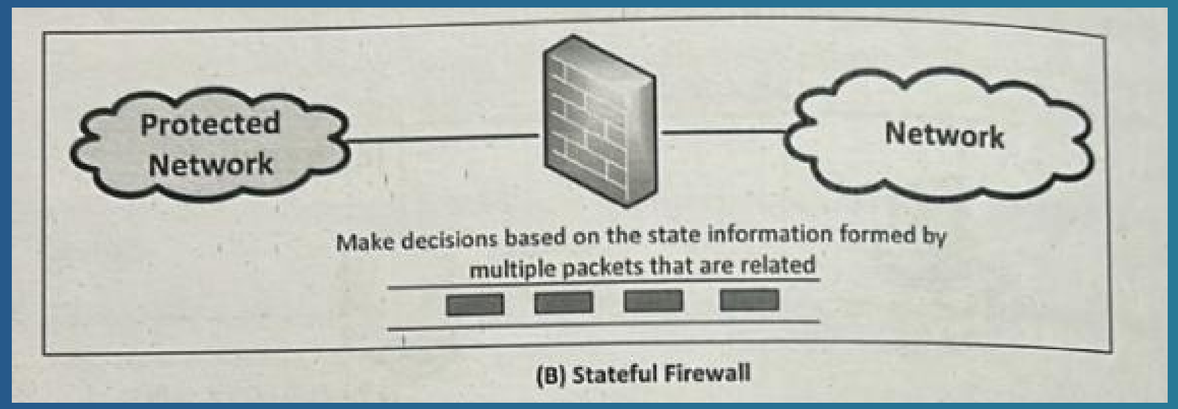


(A) Packet Filter Firewall

STATEFUL FIREWALL

A **stateful firewall** monitors the state of network connections and keeps track of all communication between devices until the connection is closed. It stores information about active connections in a state table to understand the context of packets, not just their headers. This helps the firewall make better decisions about which packets are safe to allow or block. Some stateful firewalls also check application-level data to identify and track related connections for well-known protocols.

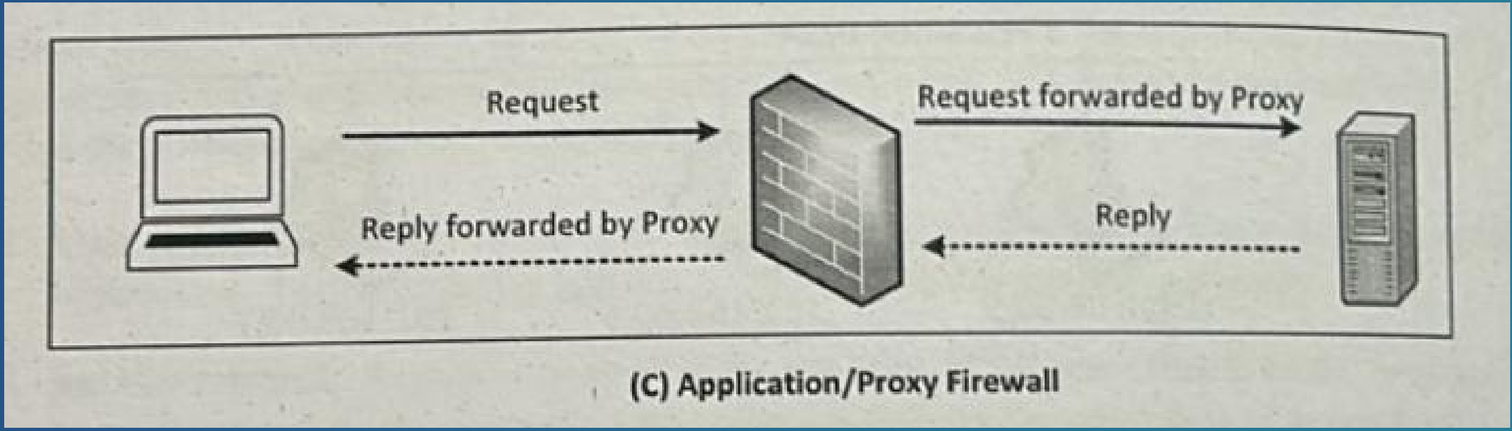
Compared to packet filters, stateful firewalls offer stronger security. For example, they allow only packets that belong to an already established connection, reducing the risk of spoofing and unauthorized access. They do not need to open multiple ports for communication like packet filters, which makes them more efficient and secure. Overall, a stateful firewall provides deeper inspection and more reliable protection for network systems.



(B) Stateful Firewall

APPLICATION/PROXY FIREWALL

An **application or proxy firewall** controls the input, output, and access of data to and from an application or service. Unlike normal firewalls that check only up to the transport layer, this firewall examines network traffic up to the application layer. It works as an intermediary between the client and the destination host, meaning the client connects to the proxy, which then connects to the destination. The firewall analyzes the data and decides whether it should be allowed or blocked, protecting the internal network from direct contact with outside sources. This provides better security and helps prevent the leakage of sensitive information. However, it can be slower than other firewalls because it inspects the entire data packet, making it less suitable for real-time or high-bandwidth use. One major advantage is its ability to authenticate users directly, which helps prevent IP spoofing and improves overall network safety.



APPLICATION/PROXY FIREWALL AND WEB PROXY

- An application or proxy firewall controls the input, output, and access of data to and from an application or service.
- Unlike packet filters that only check up to the transport layer, it inspects data up to the application layer.
- A common type is the web proxy, which controls what websites users can access and is mainly used for filtering outgoing traffic.
- A popular web proxy program is Squid, which helps with content filtering, caching, and monitoring user activities.
- Web proxies can also hide the real IP address of a client, providing privacy and additional security.
- Another related type is the SOCKS proxy, which works at a lower layer and can handle different protocols like HTTP, FTP, and SMTP, providing flexible and secure data transmission.

IMPLEMENTING SIMPLE FIREWALL USING A NETFILER

IMPLEMENTING SIMPLE FIREWALL USING A NETFILTER

- The best way to understand a technology is by implementing it practically.
- In this section, a simple packet filter firewall is created in Linux.
- Packet filtering happens inside the kernel, so the code must run within the kernel space.
- Linux provides two key technologies for this: Netfilter and Loadable Kernel Modules (LKM).
- Netfilter provides hooks at important points in the packet's path, allowing extra logic for filtering packets.
- These program logics are installed in the kernel, and loadable kernel modules make it easy to add them without recompiling the entire kernel.

WHAT ARE LOADABLE KERNEL MODULES

The Linux kernel is designed to be modular, which means only the most important parts are loaded into memory when the system starts. Extra features or device drivers can be added later as loadable kernel modules (LKMs) without restarting the system. These modules can be loaded and unloaded dynamically whenever needed. Kernel modules allow developers to extend kernel functionality easily.

Modules do not run as separate programs but work within the kernel on behalf of user processes. To insert or remove a module, root privileges are required. Each kernel module has two main parts: an initialization function that runs when the module is loaded and a cleanup function that runs when it is removed. These are defined using module_init() and module_exit(). This modular approach makes Linux more flexible, efficient, and easy to update or modify without changing the entire kernel.

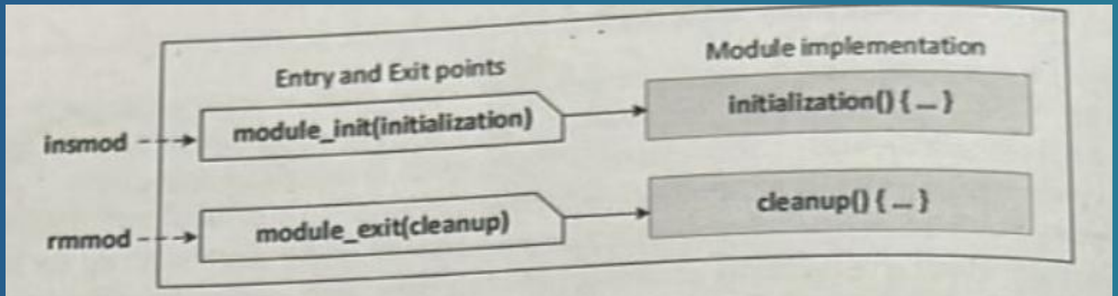


Figure 7.3: Loadable Kernel Modules

WRITING AND COMPILING A KERNEL MODULE


```c
#include <linux/module.h>
#include <linux/kernel.h>

int initialization(void) {
    printk(KERN_INFO "Hello World!\n");
    return 0;
}

void cleanup(void) {
    printk(KERN_INFO "Bye-bye World!\n");
}

module_init(initialization);
module_exit(cleanup);
MODULE_LICENSE("GPL");
```

- A simple kernel module example is hello.c. It prints a message when loaded and another when removed
- printk() is used to display messages in the kernel log.
- The MODULE_LICENSE("GPL") line defines the license to avoid warnings during compilation.
- To build the module, we use a Makefile that compiles the .c file into a .ko (kernel object) file.
- Commands like make and make clean help build and remove the compiled files.
- This process prepares the module so it can be safely inserted or removed from the Linux kernel at any time.

MAKEFILE FOR BUILDING MODULE

```
obj-m += hello.o

all:
    make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules

clean:
    make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
```

INSTALLING AND MANAGE KERNEL MODULES

```
$ make
$ sudo insmod hello.ko      # Insert module
$ lsmod | grep hello        # List loaded modules
$ sudo rmmod hello          # Remove module
```

```
$ dmesg
[... ] Hello World!
[... ] Bye-bye World!
```

Building and Loading the Module

- Use make to compile the kernel module.
- Run sudo insmod hello.ko to insert the module into the kernel.
- Use lsmod | grep hello to check if the module is loaded.
- Run sudo rmmod hello to remove the module from the kernel.

Verifying Execution

- Use the dmesg command to check the kernel log.

You should see:

- “Hello World!” when the module is loaded.
- “Bye-bye World!” when it is removed.
- This confirms the module is working correctly.

NETFILTER AND HOOOKS

>Netfilter is a Linux kernel framework for packet processing and filtering.
>It defines hooks (specific points) in the network stack where modules can analyze or modify packets.
>Developers register callback functions to these hooks; when a packet reaches a hook, all functions are executed.

Five main hooks (IPv4):

NF_INET_PRE_ROUTING – Before routing decision (incoming packets)
NF_INET_LOCAL_IN – Packets destined for local host
NF_INET_FORWARD – Packets being routed to another host
NF_INET_LOCAL_OUT – Locally generated packets
NF_INET_POST_ROUTING – After routing (outgoing packets, SNAT)

Hook Return Values:

NF_ACCEPT: Allow packet
NF_DROP: Discard packet
NF_QUEUE: Send to user space
NF_STOLEN: Take packet ownership
NF_REPEAT: Reprocess packet

Hook Priority:

Multiple hooks can exist at each point; lower value = higher priority
Example: NF_IP_PRI_FIRST (highest), NF_IP_PRI_LAST (lowest)

HOOKING FUNCTION TO NETFILTER

Netfilter allows developers to attach custom functions (called hooks) inside the Linux kernel to inspect or modify packets as they pass through the network stack.

To hook a function, three main steps are followed:

1.Define Hook Functions: Functions are defined in C to handle packets and return a verdict such as NF_ACCEPT, NF_DROP, etc.

Example: unsigned int hello1(...) { printk("Hello, Netfilter!"); return NF_ACCEPT; }

2.Register Hook: Each function is linked to Netfilter using a structure called nf_hook_ops.

Important fields include:

hook – the function name
hooknum – the hook point (e.g., NF_INET_LOCAL_OUT)
priority – order of execution (lower value = higher priority)

The hook is registered using: **nf_register_net_hook(&init_net, &hook);**

3.Unregister Hook: Hooks must be unregistered when the module is removed using: nf_unregister_net_hook(&init_net, &hook); This prevents kernel crashes caused by invalid hook references.

EXPERIMENTING WITH THE HOOK FUNCTIONS

After compiling the kernel module, we get helloFilter.ko.

This module is added to the kernel using the command:

sudo insmod helloFilter.ko

When we ping an external host (e.g., ping 1.1.1.1), the ICMP packet passes through the LOCAL_OUT hook and triggers the registered functions hello1 and hello2.

Since hello2 has higher priority (-180) than hello1 (-100), hello2 executes first.

If we change the return value of hello2 from NF_ACCEPT to NF_DROP, the packet is dropped and the ping fails with the message “Operation not permitted.”

This experiment shows that:

- Higher priority hooks execute before lower ones.
- Return values like NF_ACCEPT or NF_DROP control whether packets continue or are blocked.
- Hook behavior can be verified using the dmesg command to view kernel messages.

STEPS TO IMPLEMENTATION

The goal is to implement a simple packet filter that blocks DNS queries to the IP address 8.8.8.8 using Netfilter.

To achieve this, we write a hook function that drops any UDP packet whose destination IP is 8.8.8.8 and destination port is 53.

Code Function (blockUDP):
- Converts the IP address (e.g., “8.8.8.8”) into binary format for comparison.
- Checks the packet’s destination IP and UDP port.
- If both match, it drops the packet using **NF_DROP**; otherwise, it allows it using **NF_ACCEPT**.

Hook Registration:
- The hook is registered to the **POST_ROUTING** hook point.
- Given the highest priority using **NF_IP_PRI_FIRST**.
- Registered using: **nf_register_net_hook(&init_net, &hook2);**

Testing:
- Compile the code and insert the module using: **sudo insmod seedFilter.ko**
- Before inserting the module, DNS query **(e.g., dig @8.8.8.8 www.example.com)** succeeds.
- After inserting the module, the query fails — packet to 8.8.8.8 is dropped.

Result:
The packet filter successfully blocks outgoing DNS requests to 8.8.8.8, showing how a simple firewall can be implemented using Netfilter hooks.

OTHER APPLICATIONS OF NETFILTER

Overview:

- Netfilter is not only used for firewalls — it can also modify network packets.
- **Example:** Changing the TTL (Time To Live) value in IP packets.
- This demonstrates how Netfilter hooks can be used for packet manipulation.

Working Principle:

- A hook function is registered to **NF_INET_LOCAL_IN**.
- When a packet is received:
- The function checks if the packet is valid.
- The TTL field in the IP header is changed to 99.
- The modified packet is then accepted and forwarded.

Conclusion:

- Netfilter can inspect and modify packets dynamically.
- Useful for custom packet handling, not just filtering.
- Demonstrates Netfilter’s flexibility in kernel-level networking.

OTHER APPLICATIONS OF NETFILTER

Overview:

- Netfilter is not only used for firewalls — it can also modify network packets.
- **Example:** Changing the TTL (Time To Live) value in IP packets.
- This demonstrates how Netfilter hooks can be used for packet manipulation.

Working Principle:

- A hook function is registered to **NF_INET_LOCAL_IN**.
- When a packet is received:
- The function checks if the packet is valid.
- The TTL field in the IP header is changed to 99.
- The modified packet is then accepted and forwarded.

Conclusion:

- Netfilter can inspect and modify packets dynamically.
- Useful for custom packet handling, not just filtering.
- Demonstrates Netfilter’s flexibility in kernel-level networking.

CONFIGURING LINUX FIREWALL USING IPTABLES

THE STRUCTURE OF IPTABLES FIREWALL

The iptables firewall is used not only for packet filtering but also for modifying and managing network packets. It organizes rules into tables, chains, and rules.

- The filter table handles packet filtering through the INPUT, FORWARD, and OUTPUT chains.
- The nat table modifies source or destination network addresses.
- The mangle table is used for packet content modification.

Each chain corresponds to a Netfilter hook, where rules are enforced. Rules can be added using:

iptables -t filter -A INPUT <rule>

Here, –t specifies the table, and –A appends a rule to the chain.

Common options include:

- –A: Add rule
- –D: Delete rule
- –R: Replace rule
- –L: List rules

Rules can also be deleted by their number, e.g.:
iptables –D FORWARD 2
This deletes the second rule in the FORWARD chain.

Table 7.1: `iptables` tables and chains

Table	Chain	Functionality

`filter`	`INPUT`
	`FORWARD`
	`OUTPUT`
		Packet filtering

`nat`	`PREROUTING`
	`INPUT`
	`OUTPUT`
	`POSTROUTING`
		Modifying source or destination network addresses

`mangle`	`PREROUTING`
	`INPUT`
	`FORWARD`
	`OUTPUT`
	`POSTROUTING`
		Packet content modification

TRAVERSING CHAINS AND RULE MATCHING

When a network packet reaches an interface, it passes through iptables chains in a specific order. The path depends on whether the packet is for the local system or being forwarded to another network.

- Incoming packets first go through PREROUTING (in mangle and nat tables).
- If the packet is for the local machine, it passes through INPUT chain and then to the local process.
- If it's to be forwarded, it goes through the FORWARD chain.
- Outgoing packets pass through OUTPUT, then POSTROUTING.

Each chain has rules checked one by one. When a rule matches, an action (target) is applied. Common targets:

- ACCEPT – allow packet
- DROP – block packet
- JUMP (-j) – move to another chain

Example:

iptables -A INPUT -p udp --dport 53 -j ACCEPT
iptables -P INPUT DROP

This allows DNS (UDP port 53) and drops all other packets.

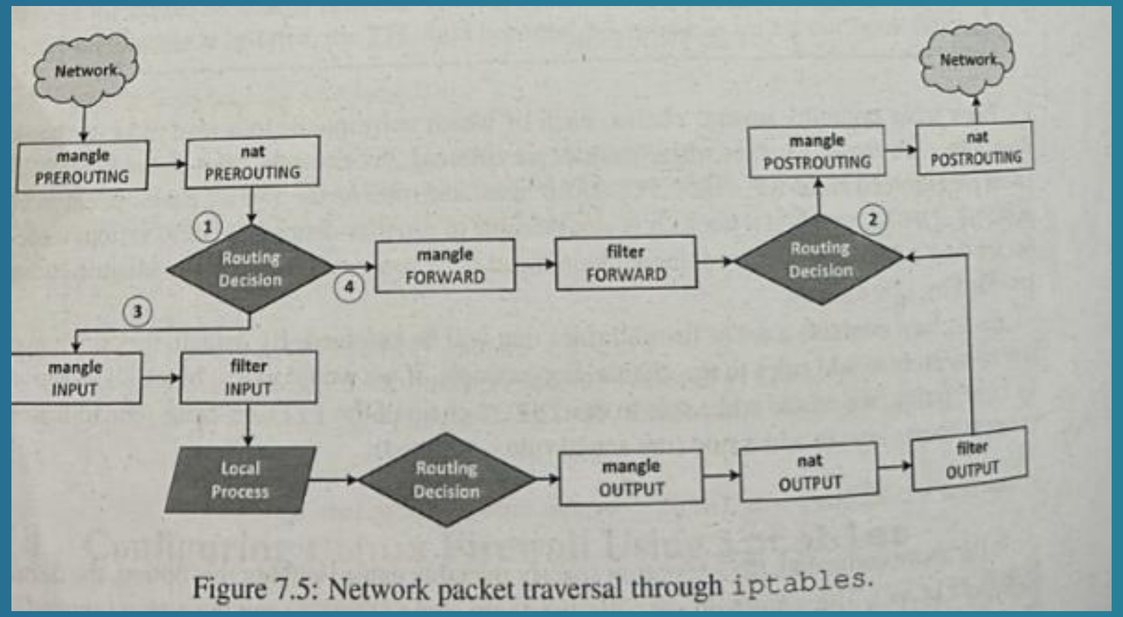


Figure 7.5: Network packet traversal through iptables.

SETTING FIREWALL RULES USING IPTABLES

The iptables command is used to set up firewall rules that control how packets are handled. The basic format is:

iptables [-t table] -A INPUT <rule> -j <target>

Here, -t specifies the table, -A adds a rule to a chain, and -j defines the target action such as ACCEPT or DROP.

Rules use criteria to match packets, such as:

- **-s → Source address**
- **-d → Destination address**
- **-i → Incoming interface**
- **-o → Outgoing interface**
- **-p → Protocol (TCP, UDP, ICMP)**

Example:

iptables -A INPUT -s 192.168.60.6 -j ACCEPT
iptables -A INPUT -s 192.168.60.7 -j DROP

These rules accept packets from one IP and drop from another.

You can also filter by protocol type:

iptables -A FORWARD -i eth0 -p tcp --dport 22 -j ACCEPT
iptables -A INPUT -p icmp --icmp-type echo-request -j DROP

IPTABLES MATCH EXTENSIONS

iptables can be extended using match extensions and target extensions. These extensions add more control and flexibility when defining rules.

1. Limit Module:

Controls the rate of traffic. For example, to allow only 5 ICMP packets per minute:

```
iptables -A INPUT -p icmp -m limit --limit 10/min --limit-burst 5 -j ACCEPT
iptables -A INPUT -p icmp -j DROP
```

This accepts only the first few packets and drops the rest temporarily.

2. Statistic Module:

Used for random or periodic packet dropping.

```
iptables -A INPUT -m statistic --mode random --probability 0.5 -j DROP
iptables -A INPUT -m statistic --mode nth --every 3 --packet 0 -j DROP
```

These drop packets randomly or every third packet.

3. Owner Module:

Matches packets based on the user ID or process ID that created them (mainly in OUTPUT chain).

```
sudo iptables -A OUTPUT -m owner --uid-owner 1000 -j DROP
```

This drops packets from a specific user process.

IPTABLES MATCH EXTENSIONS

In addition to common targets like ACCEPT, DROP, and RETURN, iptables provides several extended target modules for advanced control.

1. TTL (Time To Live):
Used in the mangle table to change the packet's TTL value.

Example:

iptables -t mangle -A PREROUTING -j TTL --ttl-inc 5

This increases the TTL by 5 for all packets.

2. MASQUERADE:
Used in the nat table's POSTROUTING chain for Network Address Translation (NAT).

iptables -t nat -A POSTROUTING -j MASQUERADE -o eth0

This replaces the source IP with the system's external IP.

3. DNAT (Destination NAT):
Used to change the destination IP or port.

iptables -t nat -A PREROUTING -p tcp --dport 8000 -j DNAT --to-destination 192.168.60.5:23

It redirects traffic to another system.
DNAT can also be combined with the statistic module for load balancing across multiple servers based on probability.

CONNECTION TRACKING AND STATEFUL FIREWALL

CONNECTION TRACKING

A stateful firewall tracks active connections to monitor both incoming and outgoing packets. This tracking system records details like source and destination IPs, ports, and the connection's state.

Linux uses the conntrack framework, built on netfilter, to manage connection tracking. It helps identify whether a packet belongs to a new or existing connection.

Types of connections tracked:

TCP connections: Since TCP is connection-oriented, the firewall tracks each session using its three-way handshake process.

UDP connections: UDP doesn't form connections, but if packets are exchanged for some time, it's treated as a temporary connection.

ICMP connections: ICMP uses request-reply messages (like ping). Each pair is considered a short-lived connection.

This command lists all tracked connections with their status and expiry times.

TCP connections usually last up to 12 hours, while ICMP and UDP last around 30 seconds.

USING CONNECTION TRACKING IN FIREWALL

Connection tracking itself only monitors connections, but it helps create stateful firewalls that make intelligent decisions based on connection states.

In the given setup, the goal is to allow only SSH (port 22), HTTP (port 80), and HTTPS (port 443) traffic from outside, while blocking all others.

Basic rules:

iptables -A FORWARD -i eth0 -p tcp --dport 22 -j ACCEPT
iptables -A FORWARD -i eth0 -p tcp --dport 80 -j ACCEPT
iptables -A FORWARD -i eth0 -p tcp --dport 443 -j ACCEPT
iptables -P FORWARD DROP

To improve accuracy, connection tracking is added using the conntrack module:

iptables -A FORWARD -p tcp -m conntrack --ctstate NEW,ESTABLISHED,RELATED -j ACCEPT

This allows only packets belonging to active or related connections.

Testing shows that valid HTTPS connections succeed, while other ports (like 441 or 9090) are blocked, proving that the firewall allows only legitimate ongoing sessions.

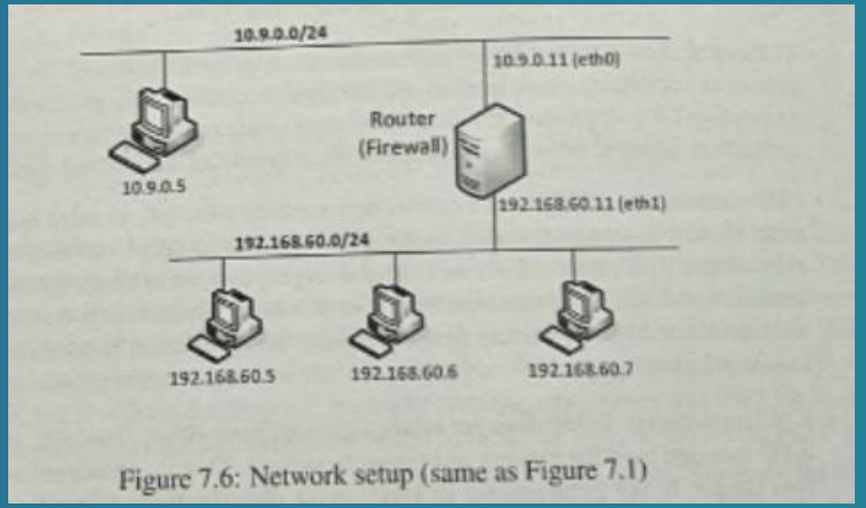


Figure 7.6: Network setup (same as Figure 7.1)

SUMMARY

- A firewall acts as a barrier between a trusted and an untrusted network, controlling the flow of data based on rules and content.
- It inspects incoming and outgoing traffic to decide which packets are allowed or blocked.
- Some firewalls only check packet headers, while others inspect application data or analyze the context of connections.
- In Linux, firewalls are built using the Netfilter framework and iptables tools provided by the Linux kernel.
- These technologies allow users to create powerful and flexible firewall rules.
- Firewalls help secure networks but are not a perfect security solution, as there are methods to bypass them.
- Common techniques used to bypass firewalls include tunnels, VPNs, and port forwarding.

THANK YOU!