Piattaforme Software per la Rete

Firewall and NAT

A.A. 2016/17

Federico Reghenzani, Alessandro Barenghi
Outline

1) Packet Filtering
2) Firewall management
3) NAT review
4) NAT management
5) The netcat tool
Packet Filtering – Introduction

- A **firewall** (or **packet filter**) is a toolkit deciding whether packets passing from an host are to be kept or discarded.
- The (main) firewall should be the **single** point of contact between the secure and insecure zone.
Packet Filtering – Why?

- Why Firewall?
  - Avoiding unauthorized connections in both directions
  - Packet sanitization: packets integrity checks are performed during filtering
  - NAT and NAPT strategies can be employed by a packet-mangling firewall
  - It can provide several other network services (e.g. VPN, DHCP, ...)
Stateful packet filtering

- A **stateful firewall** is a firewall able to track the connections status and to decide accordingly the packet filtering actions.
- It inspects also protocols of OSI level > 3.
- It discards any packet not involved from an active connection:
  - e.g. TCP packets without a previous SYN, SYN-ACK, ACK.
- Useful with some high-level protocols, such as FTP, that require ports to be opened on-the-fly.
Packet Filtering in Linux

• In Linux:
  – It’s in kernel-space due to performance reasons and integration with network stack
  – Tools are in user-space

• We will see Netfilter (kernel-space) / IPTables (user-space)
Netfilter – Structure

- NetFilter is a set of kernel modules implementing filtering functions
- The communication with the userspace management tools happens via Netlink sockets
- The NetFilter structure is based on five hooks, placed on the path of incoming/outgoing packets
- Each of the five hooks executes a set of rules each time a packet passes through it
Netfilter – Hooks

Pre-Routing → Ingoing Routing → Forward → Outgoing Routing → Post-Routing

From Level 2 → Input → Output → To Level 2

Network

NetFilter Hooks
Lower layers
Routing
A Netfilter chain is characterised by an ordered list of rules which are triggered on a certain condition on the packet.

If no rule matches the packet, the default action, i.e. the **chain policy** is adopted.

Chains are organized in tables:

- **Filter**: accept/reject packet rules
- **NAT**: network address translation rules
- **Mangle**: alter IP headers (e.g. change the TTL)
- **RAW**: general purpose (e.g. stateful actions)
Netfilter chains (2/2)

• Every builtin chain has a default policy, i.e. a default action to be performed on the packet:
  − ACCEPT: the packet flows through the hook, towards its destination
  − QUEUE: the packet is sent to the userspace via Netlink for examination
  − DROP: the packet is discarded and treated as it never existed
  − ...

• The default policy is ACCEPT for all the chains
Reasonable default policies

- **Pre-Routing**
- **Ingoing Routing**
- **Forward**
- **Outgoing Routing**
- **Post-Routing**

- **ACCEPT**
- **DROP**
- **ACCEPT**

- **From Level 2**
- **Input**
- **Output**
- **To Level 2**

**NetFilter Hooks**

**Lower layers**

**Routing**
Firewall management
IPTables

- The Netfilter behaviour is modified via the `iptables` command

- A rule is composed of two parts: the `match` and the `target`
  - The match specifies the conditions regarding the packet which will trigger the rule
  - The target specifies the fate of the packet
Possible targets for a rule are:

- ACCEPT/DROP (behave exactly as the policies)
- REJECT: drop the packet but the sender is notified of the rejection (e.g. ICMP destination unreachable)
- LOG: write a line in the kernel log
- MIRROR: swap source and destination addresses and immediately sends the packets without passing via other chains
- RATEEST: add the packets to the statistic of a rate estimator
The generic `iptables` command is structured as:

```
iptables [-t table] <action> <rule>
```

Possible actions are:

- `-A <chain>`: append a rule at the end of the chain
- `-D <chain>`: delete the specific rule
- `-I <chain> <num>`: insert the rule as the n-th
- `-R <chain> <num>`: replace the n-th rule
- `-L`: list all the rules (`-v` for additional information)
- `-F <chain>`: flush a chain (but do not reset the policy)
• Set target:
  - `-j <target>`

• Change the default policy:
  ```bash
  iptables -P <chain> <policy>
  ```
  - Policy can be ACCEPT, QUEUE, DROP
IPTables – Matching (1/6)

- The first and simplest match for a packet is to decide an action depending on the interface it was received on.
- The inbound/outbound interface matches are specified via the `-i <iface>` / `-o <iface>` option.
  - `-i <iface>` can only be used in INPUT, FORWARD and PREROUTING.
  - `-o <iface>` can only be used in OUTPUT, FORWARD and POSTROUTING.
- Usually used to differentiate the trust zone (e.g. LAN) from the untrust one (e.g. WAN).
Another common match is the source or destination IP address.

The source/destination interface matches are specified via the `-s <source> / -d <destination>` option:
- It is possible to specify the CIDR notation with `/n` bits or `/a.b.c.d` netmask.
- It is also possible to use non-contiguous netmask, e.g. `255.255.255.249` (that matches all the odd hosts > .7)
• Matching the L4 protocol used

• Option: `-p [tcp|udp|udplite|icmp|esp|ah|sctp|all]`

• Pay attention when filtering ICMP messages
• Port matching (only for TCP and UDP matching)
• Options: 
  - `--sport[s]`  `--dport[s]`
• If you want to match multiple ports you have to add:
  - `-m multiport`
• Both options allow to match a set of comma-separated ports
  - e.g. `--dports 22,80`
• If the ports to be matched are contiguous a range can be specified:
  - e.g. `--dports 6881:6890`
- Connection state matching
- Options: `-m state --state <conn_state>`
- Possible statuses:
  - NEW: the beginning of a connection
  - ESTABLISHED: the packet is part of a connection flow
  - RELATED: the packet belongs to a related connection (e.g. FTP)
  - INVALID: the packet cannot be part of valid connection
  - UNTRACKED: the packet is not being tracked
IPTables – Matching (6/6)

• Matching the connections rate

• Limiting the action of the rule: \texttt{-m limit --limit <times/s>}
  - Often used for LOG rules
  - Can be exploited for simple DOS protection

• Recent connections matching:
  - \texttt{-m recent --set}
  - \texttt{-m recent --update --seconds <n> --hitcount <k>}
  - Useful to avoid brute-force attacks
IPTables – Saving configuration

- IPTables does not keep configuration across reboots
- You can use the `iptables-save` and `iptables-restore` commands to respectively dump and restore the configuration
- Usually, each Linux distribution has a method to change the boot iptables configuration
  - e.g. in Ubuntu you can use the `netfilter-persistent` command
NAT Review
(S|D)NA[P]T

- **NAT**: Network Address Translation
  - Translating IP addresses of IP packets
- **NAPT**: Network Address Port Translation
  - It involves the L4 (typically TCP, UDP)
  - Translate IP address based on L4 used port
- **SNAT**: Source NAT
- **DNAT**: Destination NAT
- **SNAPT, DNAPT**
Source NAT

- It concentrates a number of hosts behind a single IP
- Common when a LAN needs to access a public network but only one public IP address is available
Destination NAT

- Symmetrically, it translates a public IP to a private one.
NAT features/issues

- Once a NAT strategy is actuated, the IP domains on the two sides of the NAT are effectively split
- It is possible to mitigate the IPv4 address exhaustion
- The hosts behind a NAT are (perfectly) opaque
- The host performing NAT must actively alter the packets, no end-to-end transparency
NAT table

- In order to perform a correct NAT, a table containing all the connections must be kept.
- Every time a new connection is required, a line is added to the table.
- The address translation mechanism will consistently map back the returning packets to the correct host.
- Once a connection is torn down, the line in the mapping is removed.
NAT vs NAPT

- **Network Address Translation**
  - It works at L3
  - It works with any IP traffic
  - It allows 1-to-1 mapping

- **Network Address Port Translation**
  - It works at L3 and L4
  - It works with TCP, UDP, ICMP traffic (other extensions exist)
  - It allows many-to-1 mapping
Source NAPT

- It concentrates a number of hosts behind a single IP
It enables the mapping of one single IP public address to multiple private servers.
NAT Management
Netfilter – NAT Hooks

- Pre-Routing
- Ingoing Routing
- Forward
- Outgoing Routing
- Post-Routing

From Level 2 → Input → Forward → Output → To Level 2

NetFilter Hooks
- Lower layers
- Routing

Network

NAT (rare)
Source NAT – IPTables

- Source NAT is performed in the POSTROUTING hook, when the packet is about to leave.
- The corresponding translation for the returning packet is automatically managed.
- A simple rule to match all traffic:
  - `t nat -A POSTROUTING -j SNAT --to <address>`
- If the output address is not known (e.g. dynamic public IP):
  - `t nat -A POSTROUTING -j MASQUERADE -o <dev>`
Destination NAT - IPTables

- Destination NAT is performed in the PREROUTING hook, before anything is done to the packets.
- The bi-directional communication of an established connection is also automatically managed.
- A simple rule to match all traffic:
  - `t nat -A PREROUTING -j DNAT --to-destination <address>`
- Obviously, no automatic destination selection can be performed here.
Both the Source and Destination NAT in NetFilter can be performed taking also into account ports.

The destination port of a NAT retargeted packet can simply be specified adding :port to the translated address.

A port range for both destination and source can be specified as :port-port.

- By default the ports are mapped 1:1 on the range.
The netcat tool
Netcat

- The tool **netcat** (or **nc**) allows the user to easily send UDP or TCP packets
- It can work as server or client
- **TCP Server:** `netcat -l -p <port>`
- **TCP Client:** `netcat <ip> <port>`
- **UDP Server:** `netcat -u -l -p <port>`
- **UDP Client:** `netcat -u <ip> <port>`