

Packet Filtering and NAT

Alessandro Barenghi

Dipartimento di Elettronica e Informazione
Politecnico di Milano

barenghi - at - elet.polimi.it

May 14, 2014

Lesson contents

Overview

- Netfilter/Iptables Structure
- Policy construction
- Rules setting
- Network Address Translation
- IP-over-IP tunnels

Packet filtering

What's in a Firewall...

- A firewall (or packet filter) is a toolkit deciding whether packets passing from an host are to be kept or discarded
- Structurally :
 - Integrated with the network stack as much as possible
 - Usually the packet filtering is in kernelspace, mainly due to performance reasons
 - Firewall management tools usually reside in userspace, due to ease of use
- We will examine the NetFilter (kernelspace) / IpTables (userspace) packet filtering suite

Packet filtering

Where?

The (main) firewall should be the **single** point of contact between the secure and insecure zone

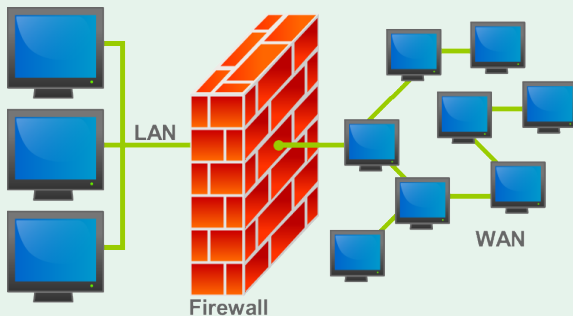


Figure: Firewall Placement

Packet filtering

Why firewalling?

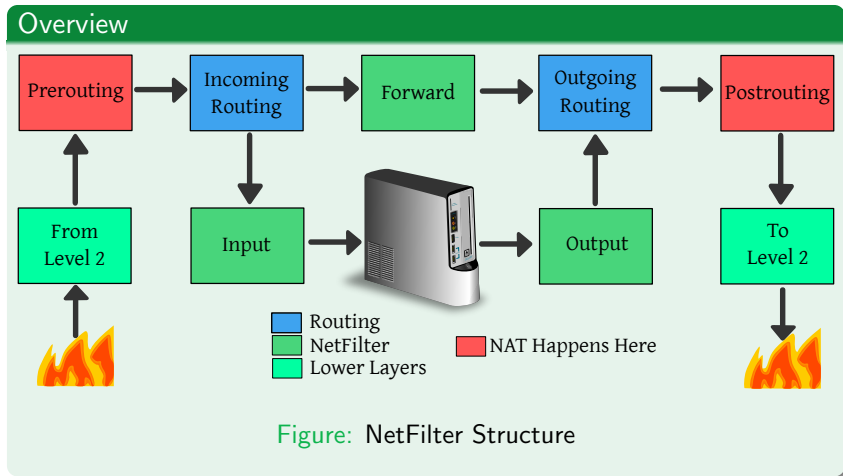
- Avoiding unauthorized connections regardless of the availability of a server
- Packet sanitization (integrity check via checksum) can be performed during filtering
- Stateful packet filtering also enforces observance of Level 3+ protocols
- Network and Port Address translation strategies can be employed by a packet-mangling firewall

Netfilter Structure

Overview

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

Structure



Netfilter chains

Overview

- 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
- Up to four **tables** containing chains are present (filter,nat, mangle and raw) for each Netfilter hook
- It is possible to create custom **chains** of rules in order to avoid the crowding of the default chain
- There is no possibility to add hook structures by default (obviously, you can write an extra module :))

Hook policies

Setting the defaults

- 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
- A hook policy can be set up with `iptables -P <chain> <policy>`
- The default policy, the one the kernel bootstraps NetFilter with, is ACCEPT for all the base chains

Hook policies

Reasonable policies

- Reasonable policies usually are:
 - PRE/POSTROUTING: set to ACCEPT, these chains are not meant for dropping
 - INPUT: set to DROP, whitelist is better than blacklist
 - FORWARD: set to DROP, “Thou shall not pass” is a reasonable default for the same reasons
 - OUTPUT: set to ACCEPT, although particularly restrictive policies may need a DROP

Rules - management

Rule structure

- 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
- For basically all match specifications , prepending a **!** mark inverts the match

Rules - management

Targets

- Possible targets (with extensions) for a rule are:
 - ACCEPT/DROP : behave exactly as the policies
 - REJECT: The packet is dropped but, if allowed by the protocol, the sender is notified of the rejection
 - LOG: A line in the kernel log is written, and the check on the chain of rules goes on
 - MIRROR: Swaps source and destination address and immediately sends the packets without passing via the other chains
 - RATEEST: adds this packet to the statistic of a rate estimator, then the chain checks are resumed

Rules - management

Rule management

- The generic iptables command is structured as : `iptables [-t table] <action> <rule>`
- Possible actions are :
 - `-A <chain>` : appends a rule at the end of the chain
 - `-D <chain>` : deletes the specific rule (the number of the rule may be indicated instead)
 - `-I <chain> <num>`: inserts the rule as the n-th
 - `-R <chain> <num>`: replaces the n-th rule
 - `-L`: lists all the rules of a chain
 - `-F`: flushes a chain (but does **not** reset the policy to ACCEPT)

Rules - 1

Matching interfaces

- The first and most simple 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
- The `-i/-o` options are limited to some chains, namely:
 - `-i` can only be used in INPUT, FORWARD and PREROUTING
 - `-o` can only be used in OUTPUT, FORWARD and POSTROUTING
- The most common use of this match is to differentiate the reasonably trusted zone of the network (LAN side) from the really untrusted side (WAN side)

Rules - 1

Matching interfaces - 2

- A special case for interface matching is the loopback interface `lo`
- This interface should **never** be filtered, lest a couple of applications *will* misbehave
- Accepting all packets with destination address equal to `127.0.0.1` is not equivalent to accepting `lo` (See RFC3330)
- Accepting all packets with destination address equal to `127.0.0.0/8` is not equivalent to accepting `lo` either (packets directed to an address you own are routed to `lo` when you self connect)

Rules - 2

Matching Addresses and ports

- The most common match is the one checking either the source `-s` or the destination `-d` address
- It is possible to specify the mask as the number of contiguous bits set to one `/n` or explicitly `/a.b.c.d`
- If the rule does not specify any mask, the default is `/32`, i.e. an exact match of the specified address
- Also non contiguous masks are usable: e.g. `255.255.255.249` (`0xFFFFFFFF9`) matches all the odd hosts up to `.7`
- Employing non contiguous masks may help in reducing the number of rules

Rules - 2

Matching Addresses

- The most common match is the one checking either the source `-s` or the destination `-d` address
- It is possible to specify the mask as the number of contiguous ones `/n` or explicitly `/a.b.c.d`
- If the rule does not specify any mask, the default is `/32`, i.e. host only
- Also non contiguous masks are usable: e.g. `255.255.255.249` (`0xFFFFFFFF9`) matches all the odd hosts up to `.7`
- Employing non contiguous masks may help in reducing the number of rules

Rules - 3

Matching protocols

- After matching the address, the next most simple match is the one on the L4 protocol
- The `-p [tcp|udp|udplite|icmp|esp|ah|sctp|all]` option specifies the protocol to be matched
- Take care in not filtering fundamental ICMP messages, f.i. Type 3 (Destination Unreachable)
- Filtering non fundamental-but-useful messages (traceroute, echo/echo reply) is widely considered a brain damage unless specific reasons to do so are present

Rules - 4

Matching Ports

- In addition to source/destination address matching, also port matching is allowed via the `--sports/--dports`
- Both options allow to match a set of comma-separated ports (e.g. `--dport 22,80`)
- If the ports to be matched are contiguous, the range : operator can be used (e.g. `--dport 6881:6890`)
- The `--sports/--dports` need the `-p` option to be explicitly specified and to be matching either UDP or TCP

Rules - 5

Matching connection status

- The difference from a regular and a **stateful** packet filter resides in the ability to filter according to the connection status
- The `-m state --state <conn_state>` match allows to specify the status of the connection (for connection oriented protocols)
- Possible statuses are :
 - **NEW** : The packet beginning a connection (f.i.TCP/SYN)
 - **ESTABLISHED** : The packet is part of a connection flow
 - **RELATED** : The packet belongs to a related connection (f.i. active FTP mode)
 - **INVALID** : The packet does not be part of a valid connection (TCP SYN/FIN packets)
 - **UNTRACKED** : The packet is not being tracked

Rules - 6

Matching rate

- Sometimes it is desirable to limit the bandwidth for a specific class of connections
- The `-m limit <times/s> match` allows to send to the rule target only a specific amount of connections
- The `-m recent --set` option tags a connection as one of a set of recently happened ones
- The `-m recent --<time> <n> --hitcount` option allows to send all the connections exceeding the hitcount/time to a specific target
- Notice that rate limiting does not in any way limit the bandwidth of the single connection

Configuration Management

Saving and Restoring

- The `iptables` utility updates a rule at a time via Netlink
- In case multiple rule changes should be performed atomically it is not a good idea to call it a volley of times
- The `iptables-apply` is able to insert atomically the changes in the Netfilter tables
- The `iptables-save` and `iptables-restore` command provide a way of dumping and restoring a full ruleset at once
- There is also an `iptables-xml` utility which converts a ruleset in XML for whatever purposes it may have

Up to now

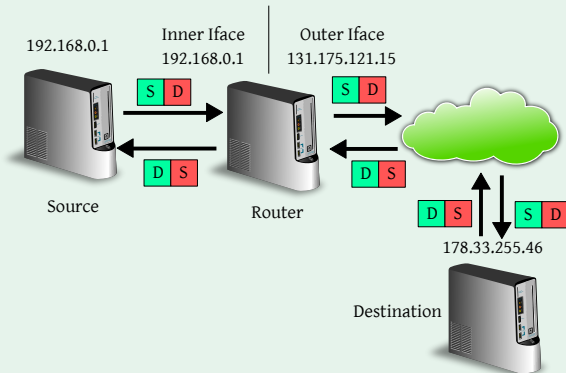
The end-to-end transparency

- Up to now, we assumed that the “any-to-any” principle of IP was respected
- All the IP addresses were assumed to be reachable via a number of routing hops
- A single packet, once built, was always delivered as-is , without any changes
- The end-to-end transparency of a communication was never questioned

Up To Now

Why?

Regular network communication



Address Translation

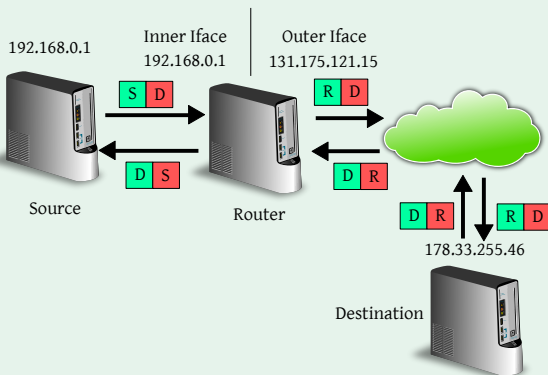
Source NAT

- Although it raises some issues, it may be needed to mask a number of hosts under a single one
- Common when a LAN needs to access a public network but only one public IP address has been bought from IANA or the ISP
- Useful to “concentrate” accesses behind a single IP
- The best candidate to perform the packet mangling is the router

Address Translation

SNAT

The general structure of a source network address translation based architecture



Address Translation

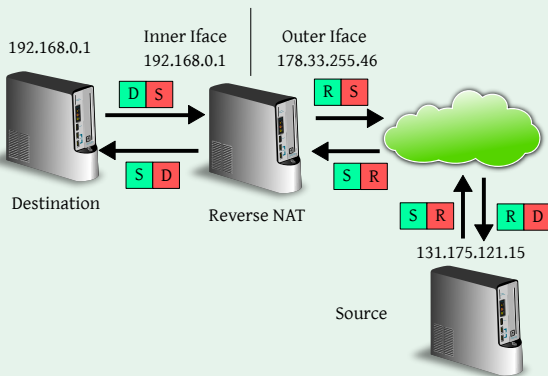
Destination NAT

- Symmetrically, it may be helpful to split the network load managed by a server
- This can be done through dynamically modifying the destination of the communication
- The operation must be performed by the alleged target of the communications
- Bonus: it allows to replace machines behind the DNAT without interrupting a service

Address Translation

DNAT

The general structure of a destination network address translation based architecture



NAT Features/Issues

Opacity

- 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^a
- The host performing NAT must actively alter the packets, no end-to-end transparency

^aalmost perfectly actually

NAT Strategy

Address Translation Table

- In order to perform a correct NAT, a table containing all the connections must be kept
- Every time a new connection is requested, 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 Issues

Potholes

- Stateless protocols do not have a proper session
- The number of connections which can be opened between two hosts is lower than the one between n and one host
- Some upper level protocols contain redundant information on the network layer^a which must be mangled too

^ayes, this **is** a terrible practice

NAPT

Potholes

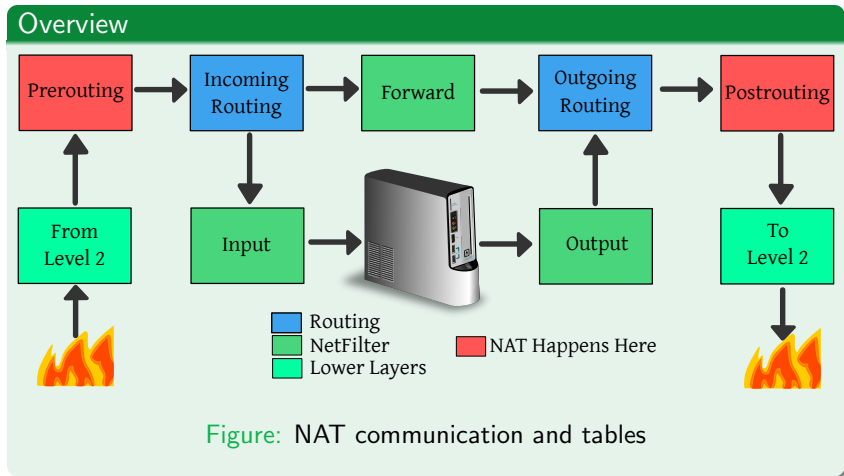
- The most straightforward extension of the NAT mechanism involves also the mangling of the 4th level datagram
- The Network Address and Port Translation is employed when a protocol needs to communicate exactly on a port
- Destination NAPT is the most common form, in order to split the incoming communications on different servers according to the service needed
- Source NAPT is rather uncommon, as usually the source port is an ephemeral port with no particular meaning

NAT: How?

Overview

- The Netfilter infrastructure allows natively to perform [S|D]NA(P)T on all the connections
- The rules specifying the NAT policies are inserted in the `nat` tables present on the `PREROUTING` and `POSTROUTING` hooks adding `-t nat` to the rule
- There is also a `nat` table in the `OUTPUT` hook, if you want to perform pre-output routing
- The connection state tables are automatically kept by the system

Structure



Source NAT

Overview

- 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 `-t nat -A POSTROUTING -j SNAT --to <address>` rule sets all the packets matching it to be masked
- The output interface specifying option `-o` and comes in handy to specify which connection to mask
- The special target `-j MASQUERADE` instructs Netfilter to choose automatically the outgoing address according to the egress interface

Destination NAT

Overview

- Destination NAT is performed (symmetrically) in the PREROUTING hook, before anything is done to the packets
- The bidirectional communication of an established connection is also automatically managed
- The `-t nat -A PREROUTING -j DNAT --to-destination <address>` rule indicates the address where the packet should be redirected
- The input interface specifying option `-i` allows rough balancing in multi-interface routers
- Obviously, no automatic destination selection can be performed here

Destination [S|D]NAPT

Overview

- 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
- For privacy reasons, it is also possible to specify a `--random` option forcing the mapping to a random port for each connection

Caveats

Overview

- Do not set the policies of the PREROUTING and POSTROUTING hooks to anything but ACCEPT, filter later
- Remember that the packets will still pass through the FORWARD chain most of the times
- Take care that the DNAT-ed packets will flow through with a different destination IP and thus should be filtered accordingly
-

Bidirectional traffic

Overview

- NATs have the great advantage of being transparent to either the source or the destinations of the connection
- However, the main issue with NATs is that only one of the sides can start a connection.
- If two separate networks are required to communicate, regardless of the fact there are not enough public IP for all hosts tunnels can be used
- The general idea of a tunnel is to employ a connection as a virtual Level 2 link

IP-over-IP tunnels

Overview

- The simplest strategy is to encapsulate an IP datagram into another IP datagram
- The outer IP communication acts as the L2 link, while the actual IP communication is the one going on inside
- This methodology is described in RFC2003 and was the first proposal for tunnels
- It has a couple of issues, among which the bad handling of multicast connections

GRE tunnels

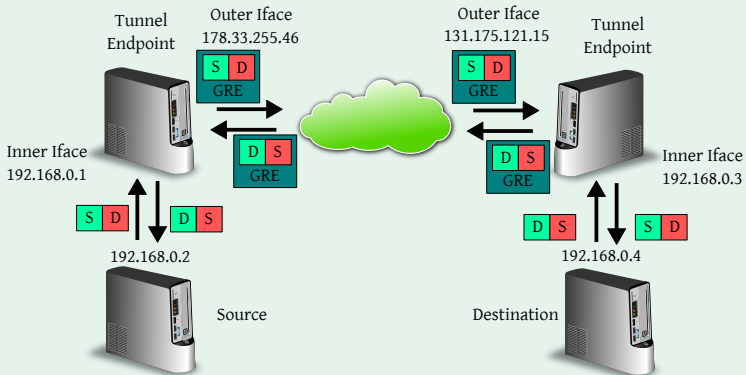
Overview

- The Generic Routing Encapsulation protocol (GRE) was developed by Cisco as a replacement for IP-over-IP
- The protocol supports multicast messages, IPv4 and IPv6 payloads natively
- It has become the de-facto standard in tunneling
- It handles tunnel loops (the destination address of a tunneled datagram is through the tunnel itself)
- Coupled with IPsec, it forms the basic structure for the large majority of the VPNs around

GRE tunnel

DNAT

The general structure of typical GRE based tunnel



iproute2 suite GRE tunnels

How-To

- The iproute2 suite provides also full support for tunnels
- Tunnels are treated as virtual, point-to-point, interfaces on which the host communicates
- Setting up a tunnel is as simple as : `ip tunnel add <interface> mode gre remote <endpoint> local <endpoint>`
- The same command, with swapped addresses should be employed on the other endpoint
- After the tunnel has been pulled up via `ip link <interface> up`, it is possible to route through it as a regular interface