stateless and stateful packet filtering

Lecture on

Advanced Internet Security

Stateful and Stateless Packet filtering

Walter Kriha

Roadmap

Part 1: Firewall Architecture

- The purpose of a firewall
- IP components important for firewalls
- Firewall Types
- Firewall limits

Part 2: Filtering Technology

- IP, TCP, ICMP filtering
- static filtering: ipchains
- dynamic (stateful) filtering: iptables
- Filtering limits
- Firewall piercing

Part 3: Services and Protocols

- frequently needed services and their problems
- dangerous services
- middleware protocols
- New threats (p2p, webservices)
- Proxies and SSL

Part 4: Securing Web Applications

- Content Management on the web
- Transactional Services
- Web Application Servers

We will deal with firewall issues rather in detail as they have a lot of impact on software architecture as well.

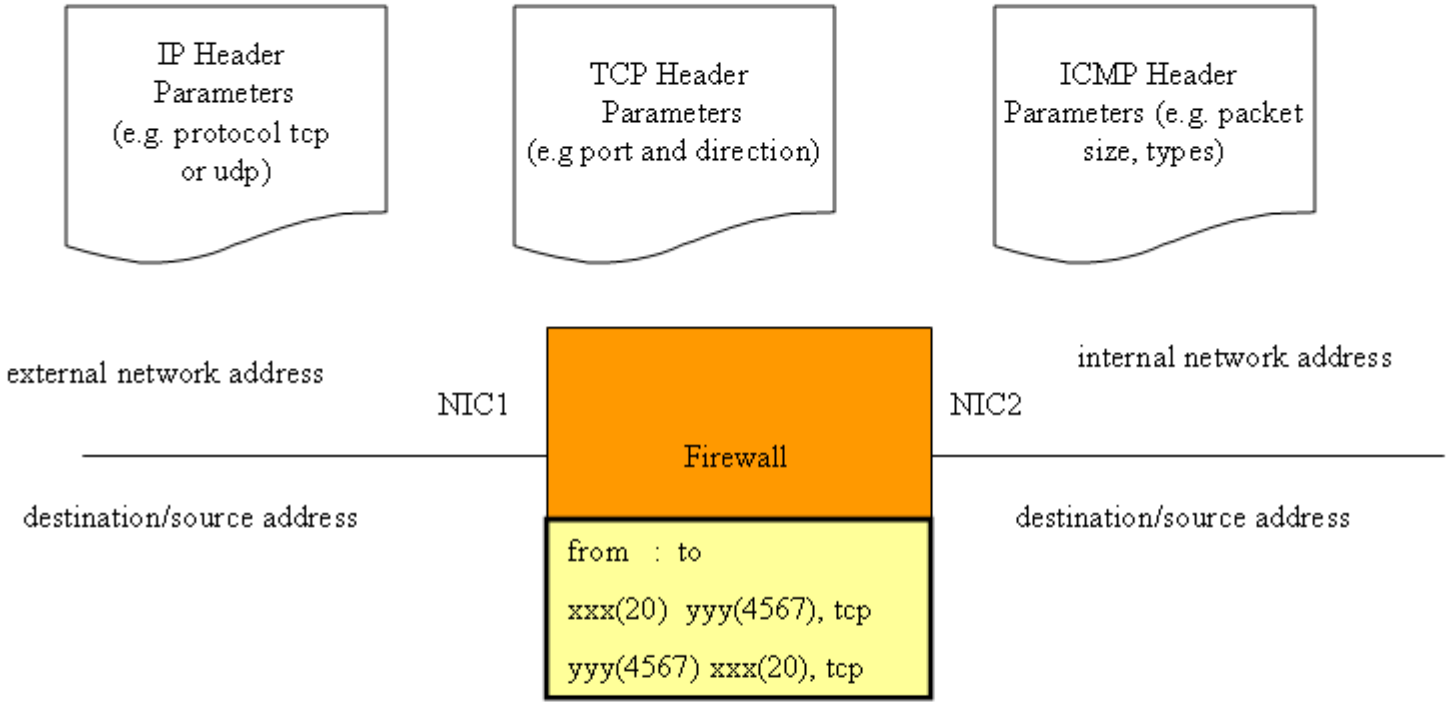
Goals for today

Learn the characteristics of packet filtering

Learn how to create a static packet filter using stateless ipchains and stateful iptables

We will cover application level filtering e.g. Web Application Firewalls later

On what can we trigger?



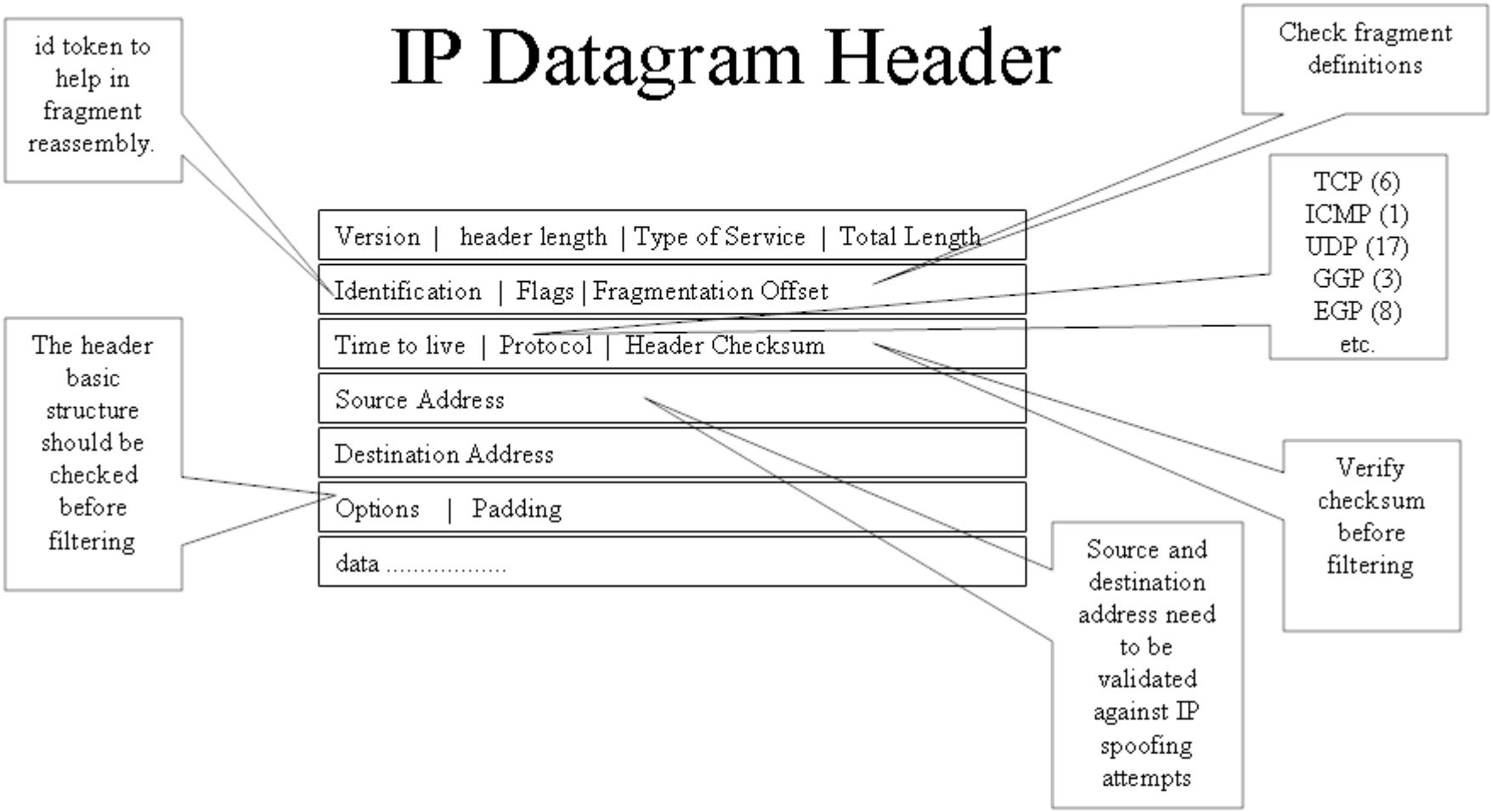
The difference between a stateless and a stateful filter lies in the memory about past requests (both incoming and outgoing) within the stateful filter. It is not necessary to allow all incoming udp requests just because the filter does not know if they are responses to previously outgoing requests. A stateful filter can selectively open a specific udp port for a certain time window because it has seen an outgoing request to this port.

Packets and Headers

- IP Header
- ICMP Packet
- TCP Packet
- UDP Packet

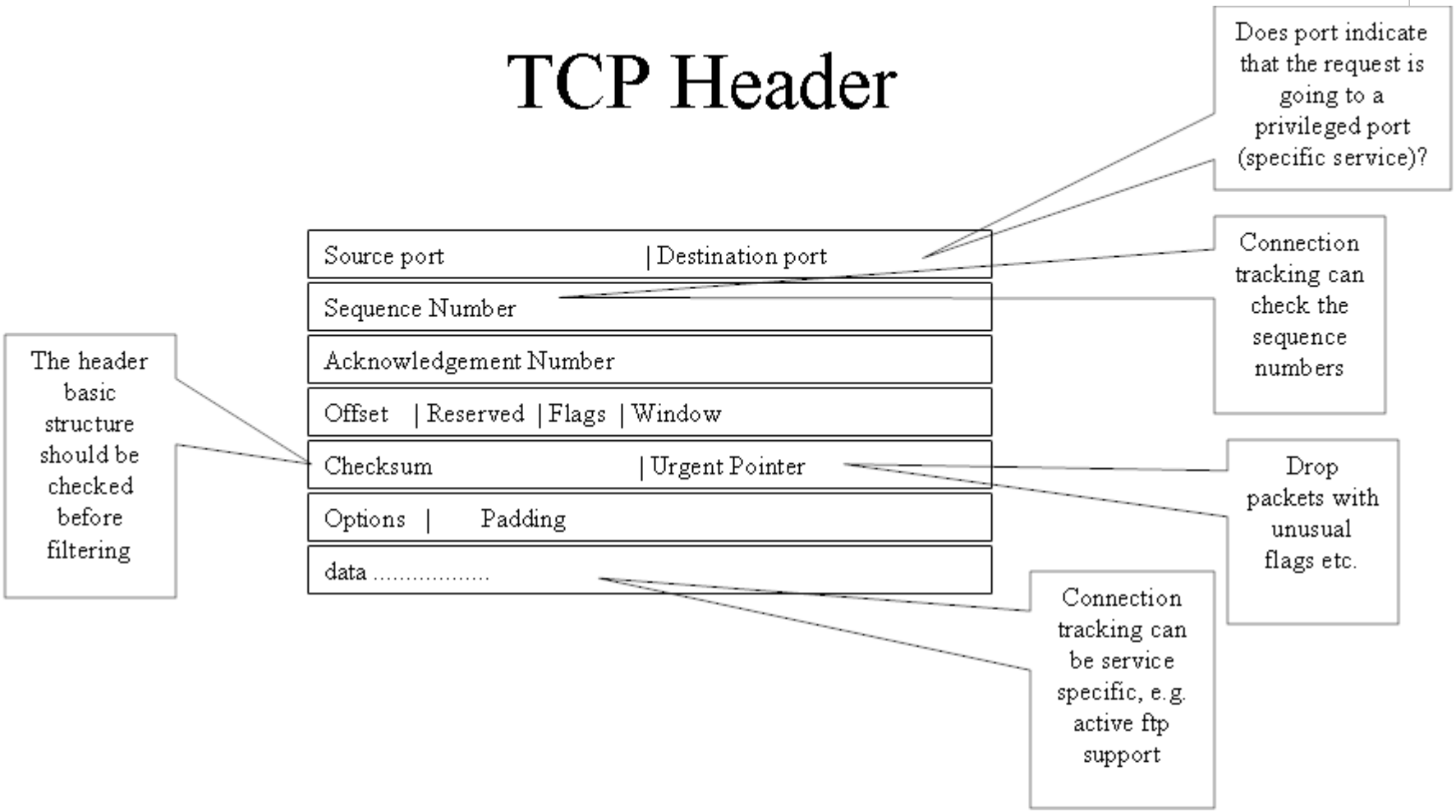
Besides context information like on which NIC a packet arrived, the information contained in headers is the most important filtering characteristic available to be matched by the patterns in filter rules. After each header we will discuss opportunities for filtering on header elements.

IP Datagram Header



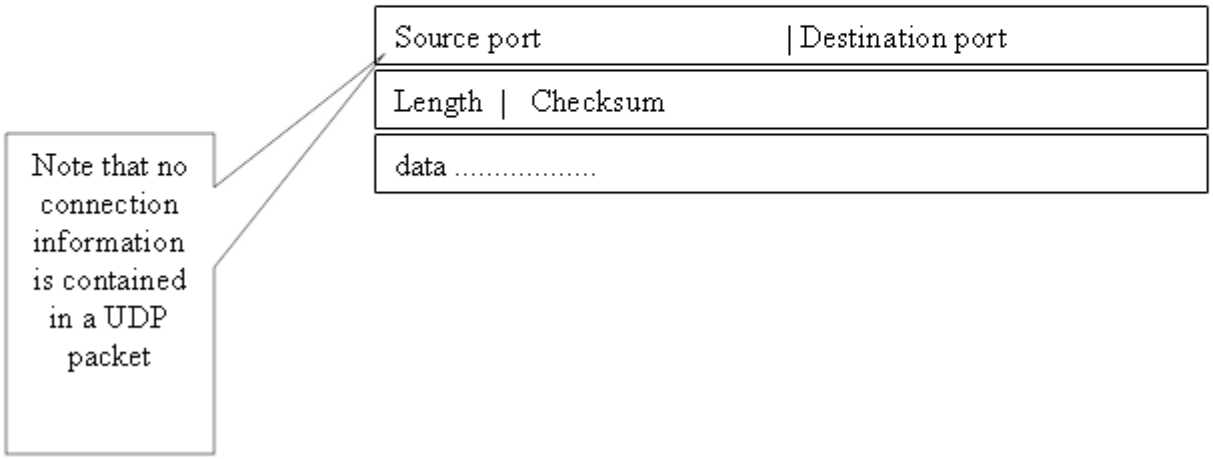
The information contained in IP headers is either used for checks against denial of service attacks, ip-spoofing etc. or used to route proper requests to the next receiver. Fragments are very critical because only the first package contains the complete information. A static packet filter should always have fragmented packets re-assembled before filtering starts. (details:http://www.freesoft.org/CIE/Course/Section3/7.htm)

TCP Header



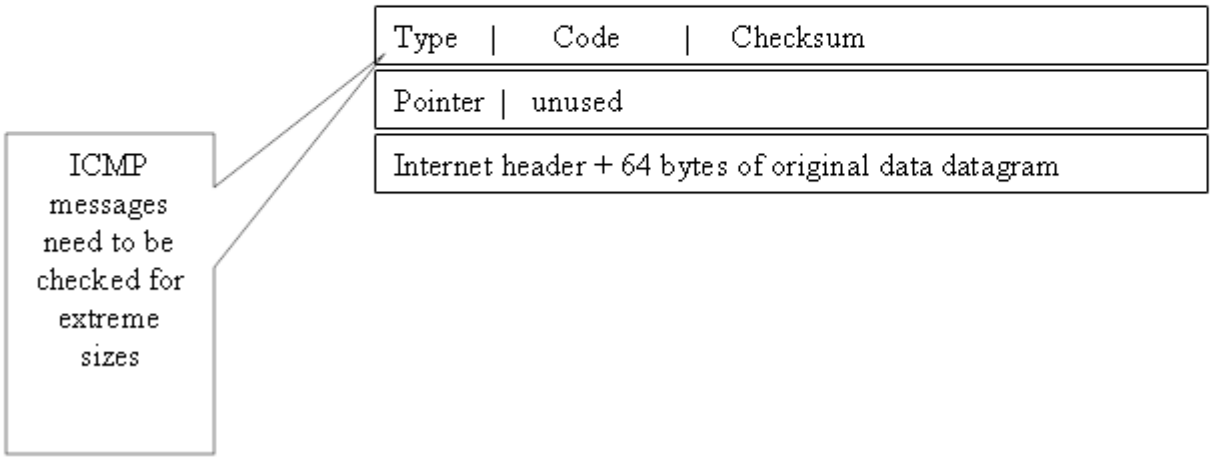
Once the basic checks for a correct format are done the most important information in the tcp header are ports, flags and possibly service information embedded in the content. Connection tracking will use sequence numbers etc. to protect the firewall from attacks on the tcp stack. Important flags for drop/accept conditions are SYN without ACK (connection init) and ACK without SYN (response to previous request)

UDP Header



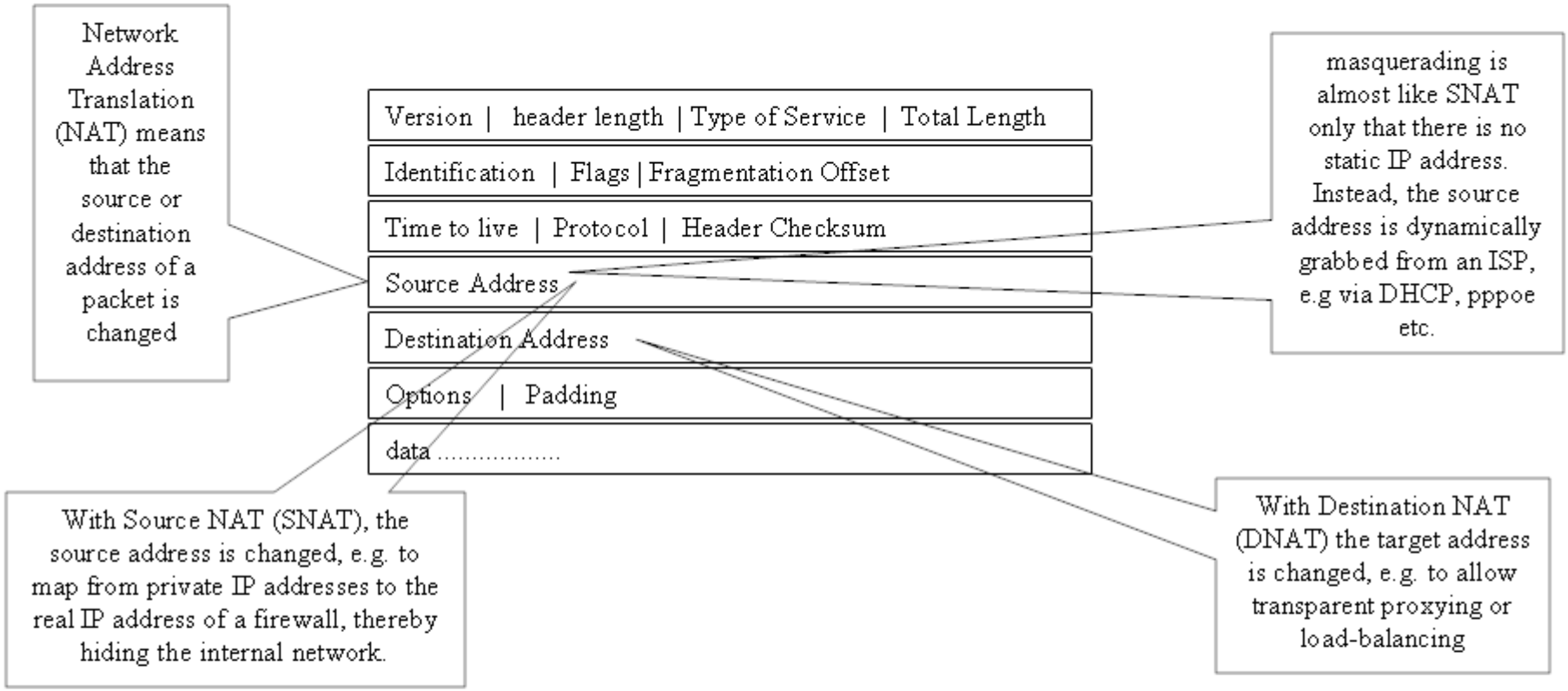
A stateless (static) filter has no way to distinguish a new UDP request coming from outside from a response to a previous outgoing request. This makes UDP especially critical.

ICMP Problem Message Header



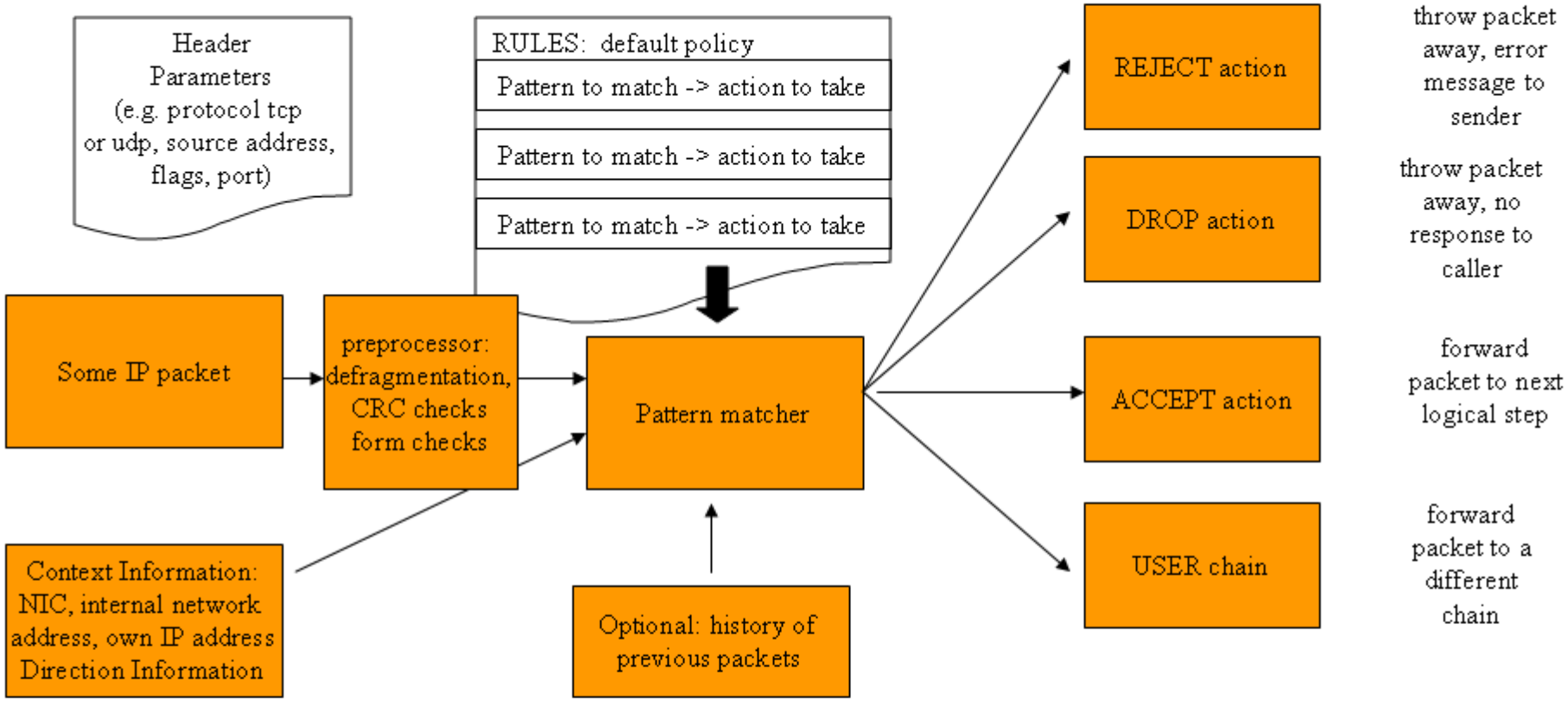
The functions of ICMP message types include redirecting routes (dangerous), fragmentation information (needed), source flow control (source quench, needed) and remote host checks (echo etc., dangerous). Please note that a static (stateless) filter needs to allow all incoming echo-replies because they could be a response to an outgoing echo-request.

NAT, SNAT, DNAT, Masquerading



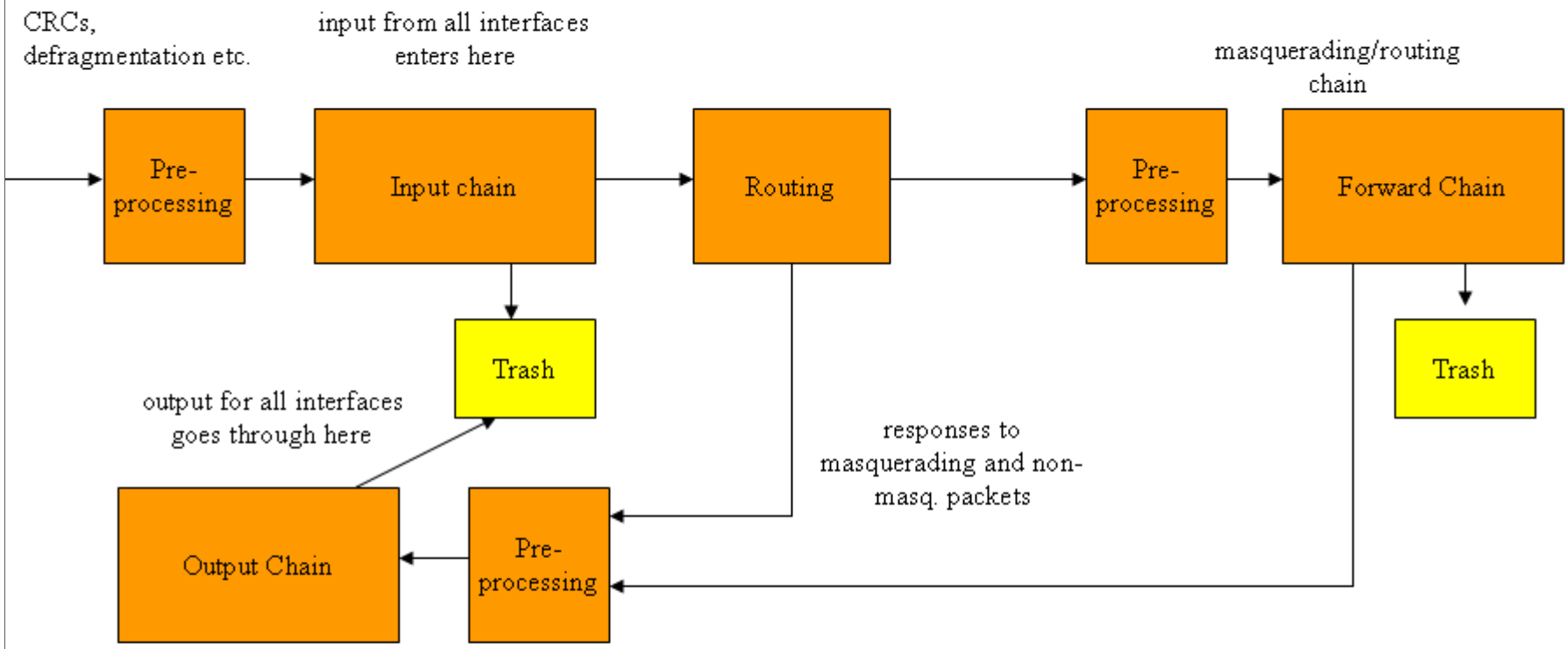
You want filtering to use the DNATed address but not the SNATed address (you care about the new target of an internal request but you don‘t care about which source address is finally used (as long as it is not an illegal one of course). That‘s why SNAT is best done after filtering.

Ipchains - A chain of rules



Rules, pattern matcher and optional history form a filter element. Some default filter elements exist (Input, Forward, Output) but users can define their own elements and build chains of filtering elements.

Default chains (ipchains example)



These are the default chains for all interfaces. Users can define additional chains. At any point a package can be thrown away. The default policy of a chain should be „reject“ or „deny“. Certain processing applies before the package contents meet the pattern matching algorithms to ensure proper package forms.

Rules which always apply

- Do net let packets with EXTERNAL SOURCE address come in through your INTERNAL NIC (those packets MUST be fakes)
- Do not let packets with INTERNAL SOURCE address come in through your EXTERNAL NIC (those packets MUST be fakes or you are using multiple interior routers to the same perimeter network (DMZ) and they believe that the shortest path is through your perimeter network. This would expose critical data from the internal network and is a big problem anyway.
- Do not let packets pass with broadcast addresses in the SOURCE address field in either direction. A reply to those packets would be a broadcast.
- Do not let packets pass with multicast addresses in the SOURCE address field in either direction. Multicast addresses are always destination addresses.
- Do not allow packets with source routing or IP flags to pass in either direction (man-in-the middle)
- Restrict ICMP packets in size
- Perform re-assembly of fragmented packets before rule processing happens.
- Set „default deny“ on all chains and log denied packets.
- During rule configuration set all chains to „default deny“ and remove those rules at the end of configuration

These rules are independent of network configuration. It is important to implement these filters IN BOTH DIRECTIONS not only to prevent malicious outsiders from entering your hosts but also malicious insiders (or unwitting users and compromised machines) from using your computing equipment for attacks on others (e.g. distributed denial of service attacks)

Special Modules for Protection

auto-defragmentation: echo 1 > /proc/sys/net/ipv4/ip_always_defrag

syn-flooding protection: echo 1 > /proc/sys/net/ipv4/tcp_syncookies

no echo broadcast replies: echo 1 > /proc/sys/net/ipv4/icmp_echo_ignore_broadcasts

no bogus error replies: echo 1 > /proc/sys/net/ipv4/icmp_ignore_bogus_error_responses

no icmp redirects: for f in /proc/sys/net/ipv4/conf/*/accept_redirects; do echo 0 > $f (for all interfaces)

no source routing: for f in /proc/sys/net/ipv4/conf/*/accept_source_route; do echo 0 > $f (for all interfaces)

no ip-spoofing: for f in /proc/sys/net/ipv4/conf/*/rp_filter; do echo 0 > $f (for all interfaces)

log suspicious packets: for f in /proc/sys/net/ipv4/conf/*/log_martians; do echo 1 > $f (for all interfaces)

allow ip-forwarding for masquerading: echo 1 > /proc/sys/net/ipv4/ip_forward

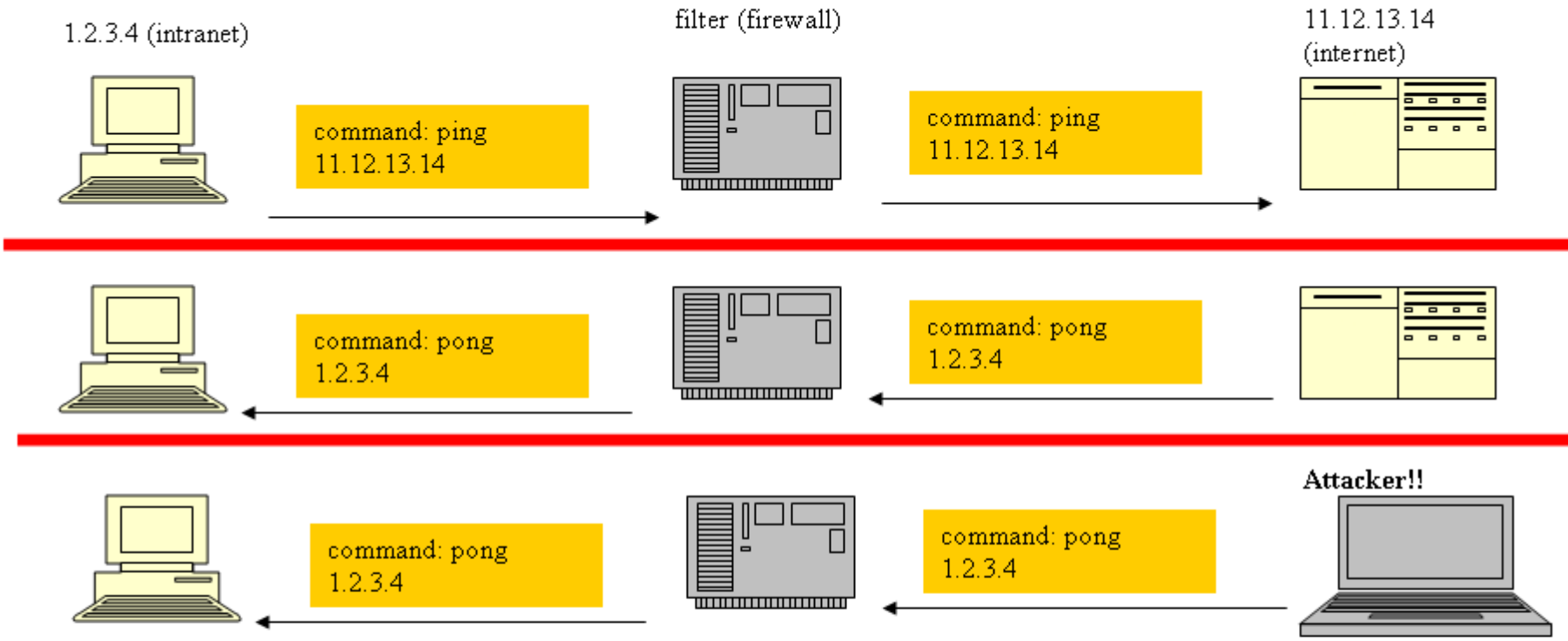
These commands need to become part of the firewall boot process (in Linux: /etc/rc.local) to make sure that they are installed properly. Modules with masquerading support for special protocols (IRC, real audio etc.) can also be installed by the filter script directly. (from Klein, Linux Sicherheit pg. 589ff)

ipchains: conditions and actions

- protocol (TCP, UDP, ICMP, IGMP)
- source and destination address
- source and destination port
- TCP connection init (ACK flag)
- ICMP types
- IP fragmentation (better solved by pre-assembling fragments before checking)
- interface

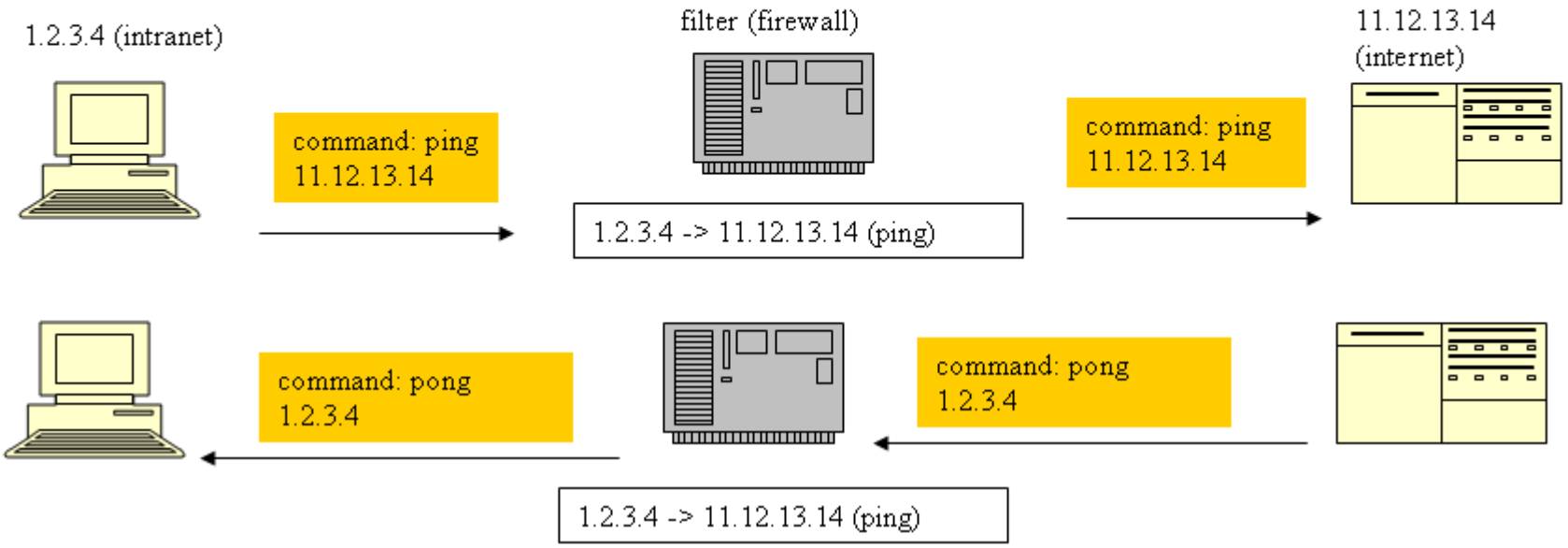
- ACCEPT (let packet pass, process it and/or forward it to next chain)
- DENY (throw packet silently away, do not generate error message)
- REJECT (generate ICMP response but throw packet away)
- MASQ (perform masquerading on the packet. Move a response directly to the output chain. Used in forward chain only)
- REDIRECT (move packet to different port on local host. Can be used for transparent proxying)
- RETURN (Use default policy in a default chain or return from user defined chain)
- NAME of user-defined chain to be called.

A stateless filter



A host in our intranet sends a ping command (icmp echo request) to an external host on the internet. As far as the stateless filter goes this is one complete transaction. It will NOT retain any knowledge about this request. When a pong (icmp echo-reply) comes in the filter can only let it pass through (no matter from where it comes!) or deny all those packets (disabling the ping service altogether). An attacker can send echo-replies which have NEVER been requested from internal hosts.

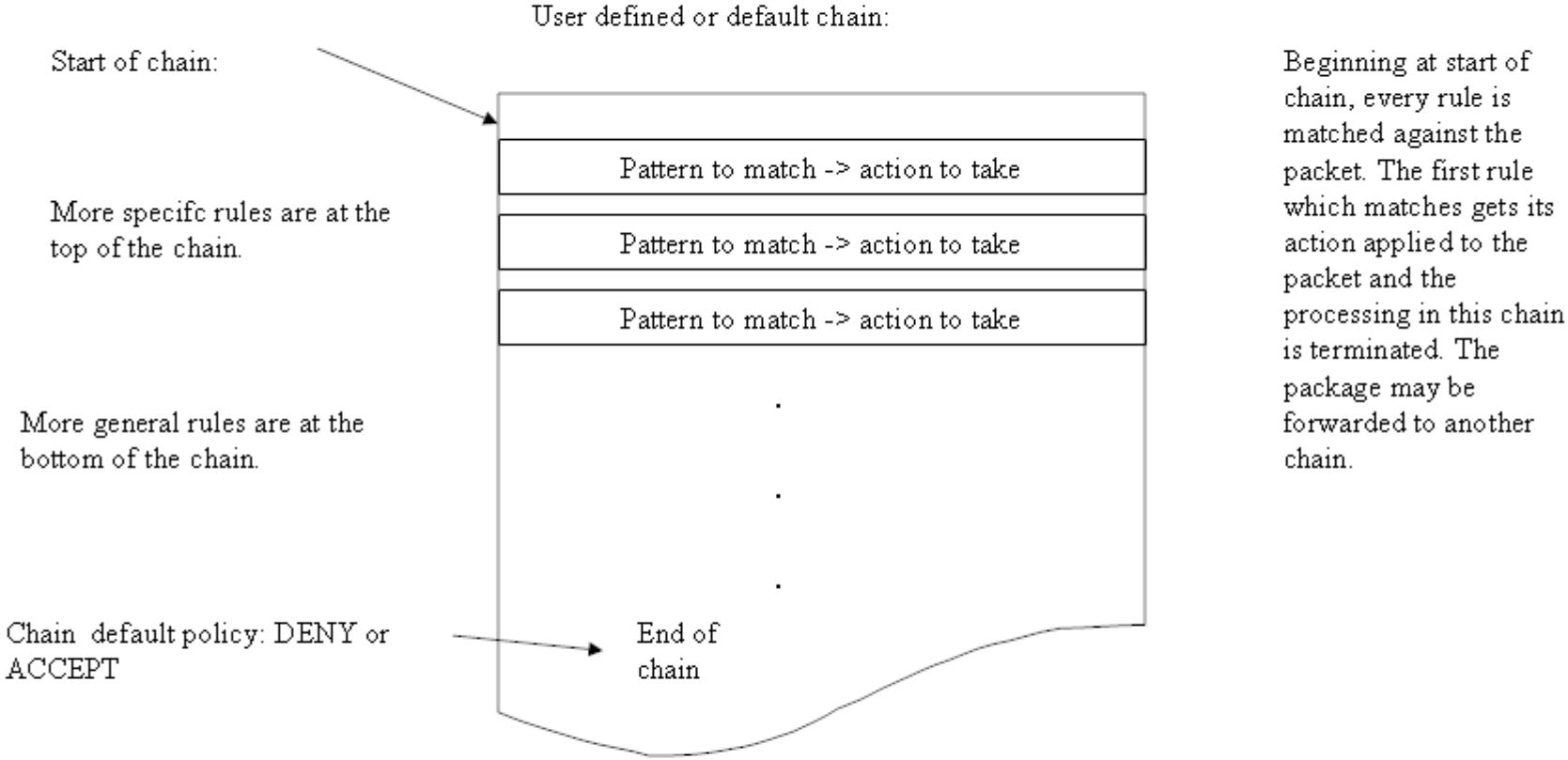
A stateful filter



There is a packet from 11.12.13.14 to 1.2.3.4 (pong), was there a ping from 1.2.3.4 to 11.12.13.14 previously? if YES, accept, if NO, drop packet

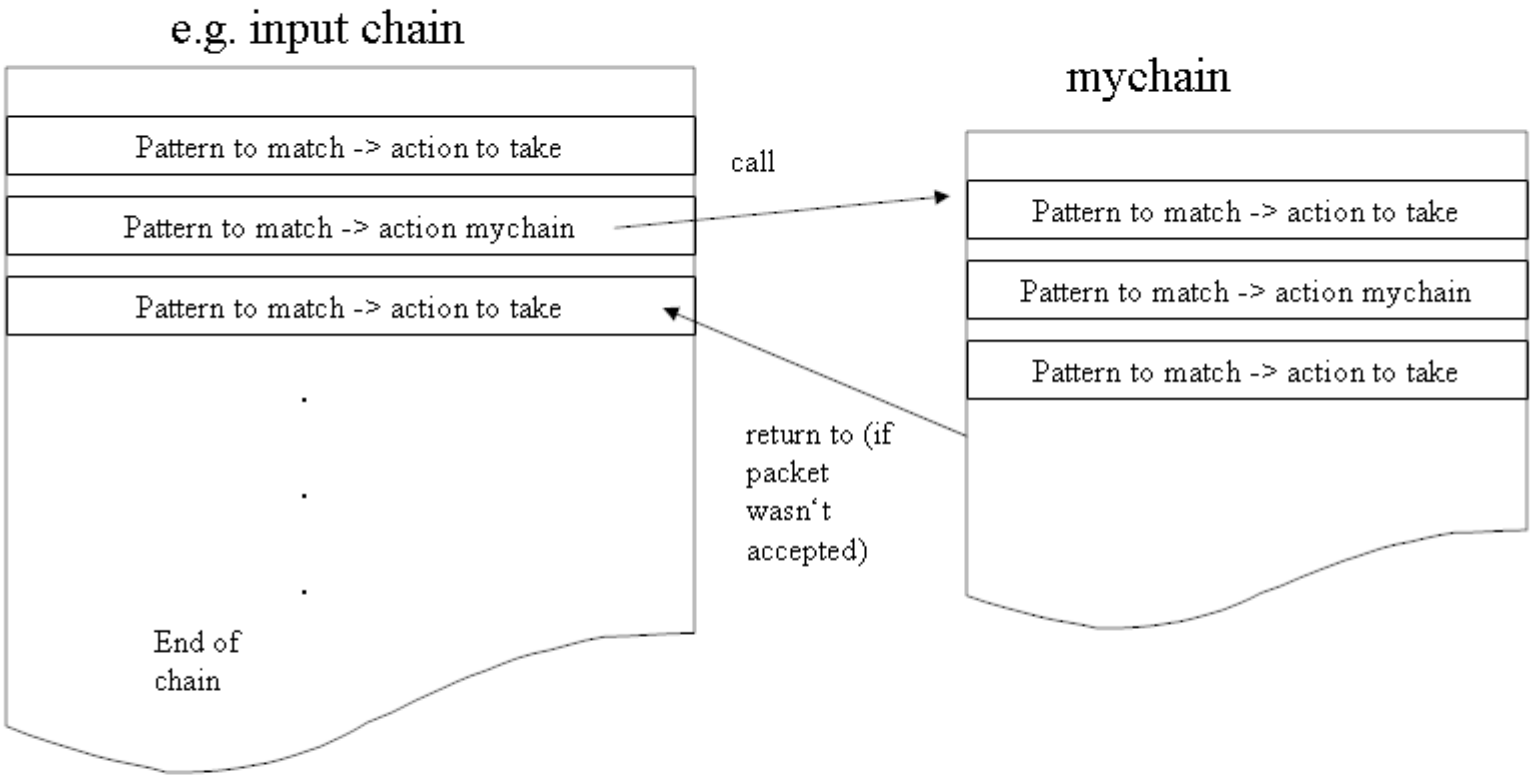
A filtering firewall keeps a table of outgoing requests. If a packet comes in from the Internet it is matched against the outgoing requests (destination, port, protocol). The whole mapping lasts only for a certain time (timeout) until the filter clears the request information. If a response is „late“, it gets dropped. An attacker would need to spoof the real sender AND hit the little window where the filter waits for responses AND the response must match port, protocol and destination. This works good for connectionless protocols (UDP, ICMP) but works even better for TCP if e.g. sequence numbers are checked as well. Even exotic protocols with several connections and directions can be tracked.

A chain of rules



Chain manipulation commands work on the whole chain. Rule manipulation commands only on a specific rule.

Chains calling chains

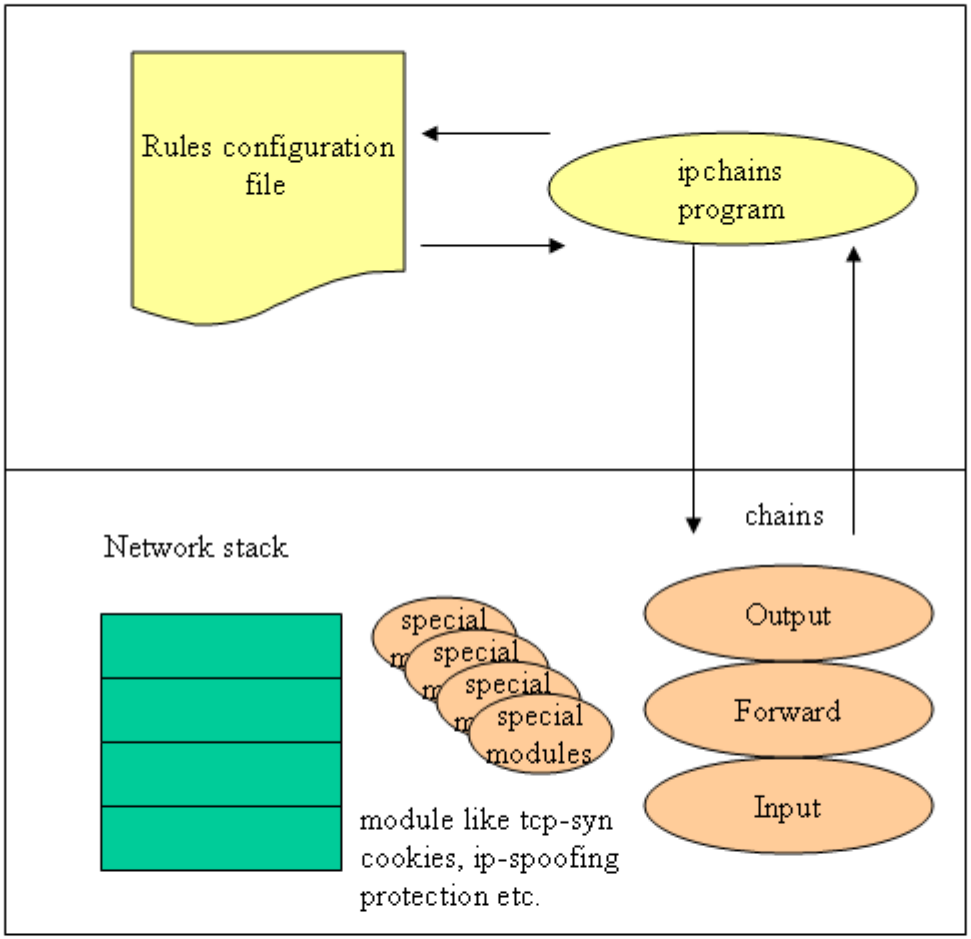


A user defined chain can be the target of an action. Processing (filtering) will continue with the first rule in the new chain and end when a rule matches the packet or at the end of the chain. At that point control returns to the calling chain.

The ipchains user level driver program

Unix user level

Unix kernel



Only the program that installs the rules is located in user space. The netfilter/iptables framework provides an interface and queue which allows filtering to happen also in user space. Kernel space filtering is extremely fast but also limited in functions and very critical. An error in a module will crash the kernel for sure.

ipchains command syntax

ipchains command [chain] parameter action

Example:

- ipchains –P input DENY (set default policy for input chain to DENY)
- ipchains –f output (delete all rules in output chain)
- ipchains –A output –i $EXT_IFACE –p tcp –s $MY_IPADDR $UNPRIVILEGED_PORTS – destination-port 80 –j ACCEPT (allow http connections from this host and ports beyond 1023 to any host port 80 using the external NIC)

„Command“ is one of the ipchains chain manipulation commands. Parameter is the pattern to match a request against. „Action“ is either a real action like „DENY“ – dropping a packet, or forwarding the packet to a user defined chain.

Chain manipulating commands

Chain manipulation:

- Create user defined chain (-N)
- Delete user defined chain (default chains cannot be deleted) (-X)
- Zero counters on all rules in chain (-Z)
- Flush default or user defined chains. (-F)
- Install default policy of a chain. (-P)
- List rules in chain (-L)

No surprise here: most commands will be the same in the new netfilter/iptables framework as well. Setting the default policy to DENY on all chains creates a „deny all“ policy and is also useful during rule manipulation.

Rule manipulating commands

Rule manipulation:

- Append new rule to chain (-A)
- Delete matching rule in chain.

with position parameter:

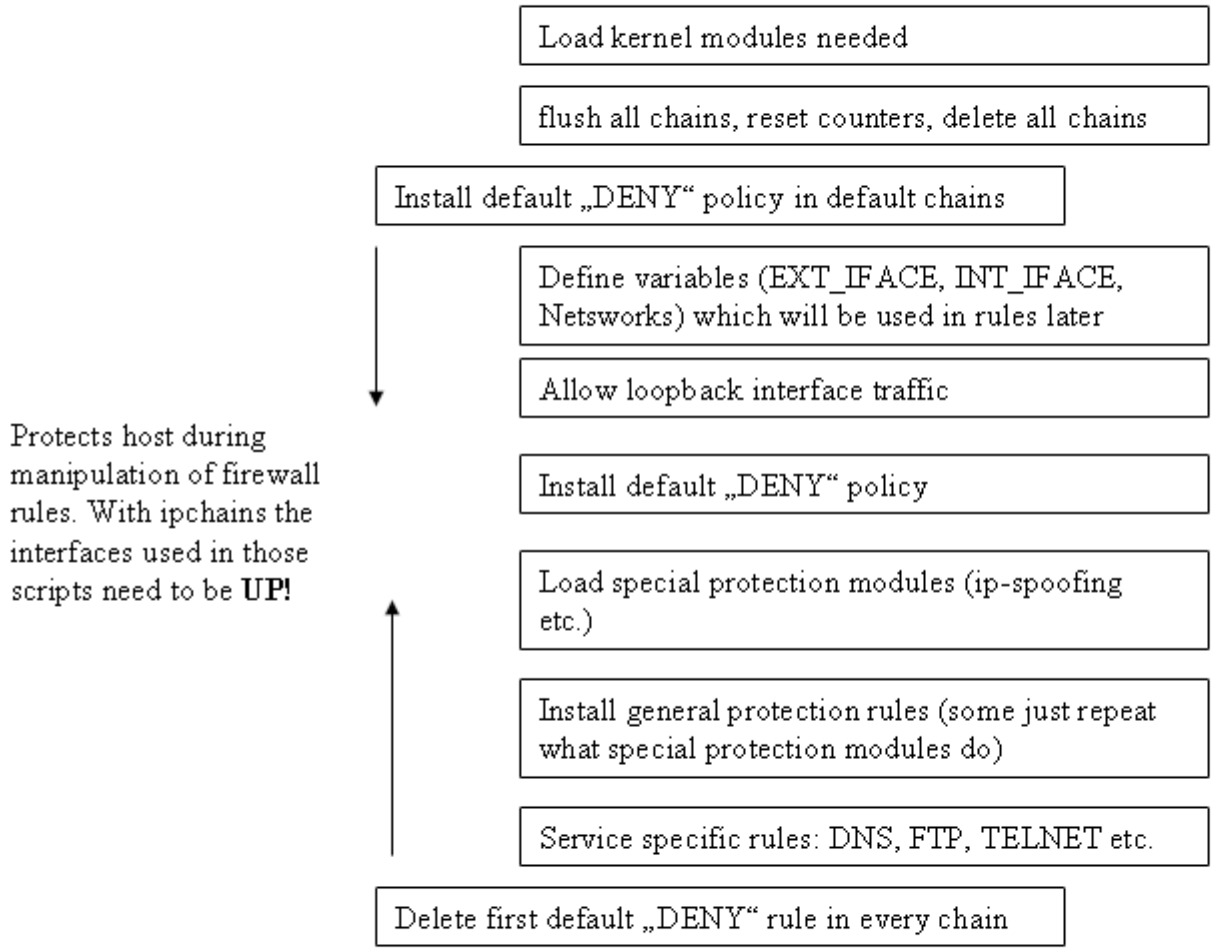
- Insert, Delete, Replace rule at POSITION in chain

masquerading rules:

- list current masquerading rules (-M –L)
- set masquerading timeout values (-M –S)

Interaktive rule manipulation is tedious and error prone. Luckily there are utilities which load and store complete configuration files containing a firewall ruleset. (ipchains-store etc.)

The structure of a firewall config file



Ipchains and also iptables follow this structure. The biggest part is of course the definition of rules for each service that needs to pass through the firewall.

A word on notations in ipchains

- Actions are in UPPER CASE (e.g. ACCEPT)
- interfaces can be given with + notation: e.g. ppp+ means all ppp... interfaces in the system
- chains are in lower case (e.g. input)
- the check for SYN bit set and ACK NOT set is –y or –syn. This is a sign for a tcp connection request.
- Significant part of a network can be described like: (CIDR notation)
    - 123.456.789.0/24 (the first three byte are significant)
    - 123.456.0.0/16 (the first two byte are significant)
    - 123.0.0.0/8 (only the first byte is significant)

ipchains examples

1. A short standalone firewall (from roaring penguin pppoe)
2. Redirection to transparent http proxy
3. User-defined chains
4. Building a DMZ

A bare bone standalone firewall

```
# Interface to Internet
EXTIF=ppp+
ANY=0.0.0.0/0
#no complete default deny policy
ipchains -P input ACCEPT
ipchains -P output ACCEPT
ipchains -P forward DENY
#flush all chains
ipchains -F forward
ipchains -F input
ipchains -F output
# Deny TCP and UDP packets to privileged ports
ipchains -A input -l -i $EXTIF -d $ANY 0:1023 -p udp -j DENY
ipchains -A input -l -i $EXTIF -d $ANY 0:1023 -p tcp -j DENY
# Deny TCP connection attempts
ipchains -A input -l -i $EXTIF -p tcp -y -j DENY
# Deny ICMP echo-requests
ipchains -A input -l -i $EXTIF -s $ANY echo-request -p icmp -j DENY
```

from: roaring penguin pppoe package. This is surely only the beginning of a firewall strategy. Masquerading can be enabled with: ipchains -A forward –s $LocalNet –d $Any -j MASQ ; echo 1 > /proc/sys/net/ipv4/ip_forward

Redirect to transparent http proxy

INT_IFACE=192.1.1.0/24

ANY=0.0.0.0/0

ipchains -A input -p tcp -s $INT_IFACE -d $ANY 80 -j REDIRECT 8080

Everything coming from the internal network with a destination of anything and port 80 (http) will be redirected to the local port 8080 where a http proxy (e.g. squid) is running.

User defined chains

```
# Create your own chain
   /sbin/ipchains -N my-chain
   # Allow email to got to the server. This is an INCOMING connection request
   /sbin/ipchains -A my-chain -s 0.0.0.0/0 smtp -d 192.1.2.10 1024:-j ACCEPT
   # Allow email connections to outside email servers. OUTGOING conn.req.
   /sbin/ipchains -A my-chain -s 192.1.2.10 -d 0.0.0.0/0 smtp -j ACCEPT
   # Allow Web connections to your Web Server. INCOMING
   /sbin/ipchains -A my-chain -s 0.0.0.0/0 1024: -d 192.1.2.11 www -j ACCEPT
   # Allow Web connections to outside Web Server. OUTGOING
   /sbin/ipchains -A my-chain -s 192.1.2.0/24 1024: -d 0.0.0.0/0 www -j ACCEPT
   # Allow DNS traffic. Opens ALL your UDP ports for UDP traffic originating
#from any server at port 53. Do you really want this?
   /sbin/ipchains -A my-chain -p UDP -s 0.0.0.0/0 dns -d 192.1.2.0/24 -j ACCEPT
```

This chain opens quite a number of ports for incoming connections. For DNS the use of an intermediate DNS server using port 53 would be advisable.

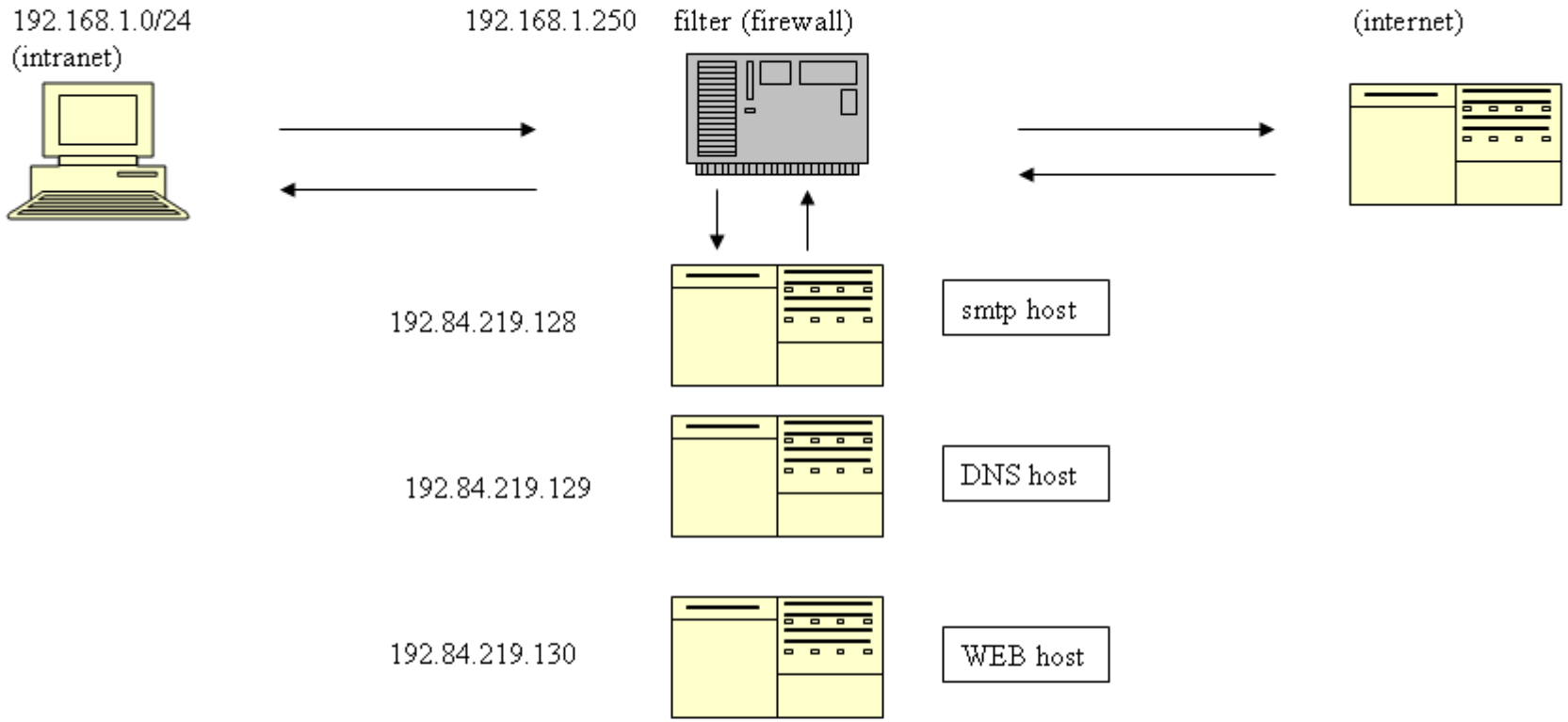
Digression: Model Checking

 # Allow Web connections to outside Web Server. OUTGOING
/sbin/ipchains -A my-chain -s 192.1.2.0/24 1024: -d 0.0.0.0/0 www -j ACCEPT

Is the ipchains statement a true implementation of the model (informally specified in the comment)? Not really. Think about the protocol used in www traffic and what the rule defines? The rule actually does allow connection establishment from internal network to any destinations port 80 using ANY kind of protocol. This is NOT what the model specified. A model checker would build a space with possible solutions of the model, reconstruct such a space from the rule and compare both. While doing so it would detect that the rule allows more solutions/options than the original model.

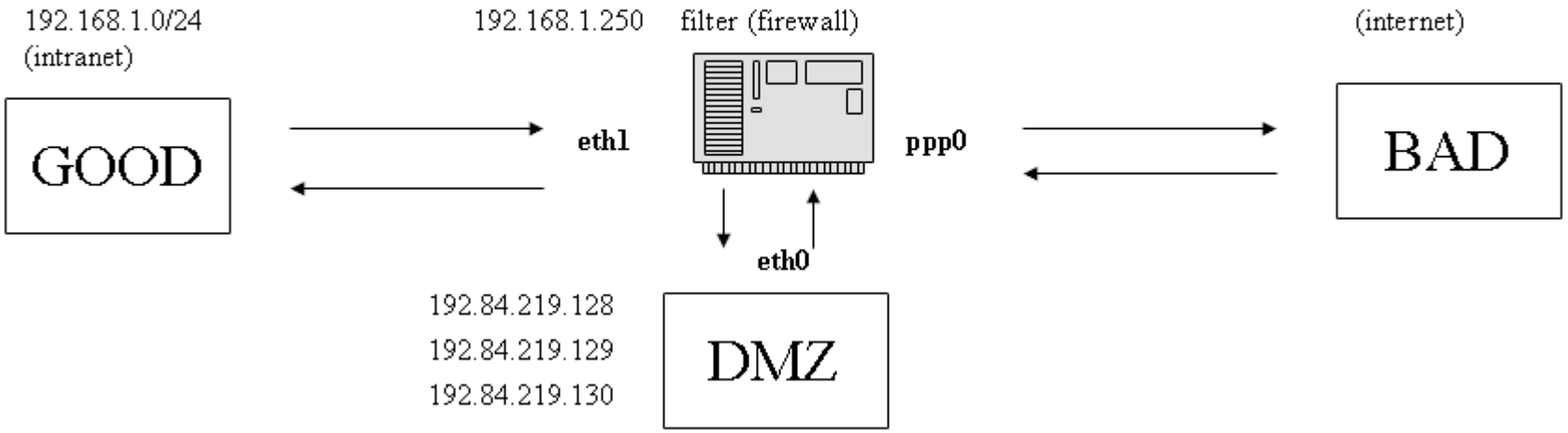
To make such model checking easier the rules/model should be monotonous, i.e. always adding rights but not taking some away. Following this advice is also good when building rule based filters manually: Do not mix deny and accept rules!

Using a DMZ



We are using routable internet IP addresses for the DMZ but internal-only addresses for our Intranet. The DMZ provides mail services (SMTP), DNS lookup via a DNS server and also access to a web server running on host 15.16.17.20. This is an example from the linux IPCHAINS-HOWTO (Rusty Russel)

Separating traffic (1)



this setup results in 6 different chains: good-bad, bad-good, good-dmz, dmz-good, bad-dmz, dmz-bad plus one chain for icmp traffic. To secure traffic directed at the firewall directly, 3 more chains are defined: good-if, bad-if, dmz-if to separate traffic coming in from the firewalls three NICs.

Separating traffic (2)

```
ipchains – N good-dmz
ipchains – N bad-dmz
ipchains –N good-bad
ipchains –N icmp-acc

ipchains – N dzm-good
ipchains – N dzm-bad
ipchains –N bad-good
```

creation of user defined chains

```
ipchains –A forward –s 192.168.1.0/24 –i eth0 –j good-dmz
ipchains –A forward –s 192.168.1.0/24 –i ppp0 –j good-bad
ipchains –A forward –s 192.84.219.0/24 –i ppp0 –j dmz-bad
ipchains –A forward –s 192.84.219.0/24 –i eth1 –j dmz-good
ipchains –A forward –s  –i eth0 –j bad-dmz
ipchains –A forward –s  –i eth1 –j bad-good
ipchains –A forward –j DENY -l
```

Only the outgoing interface is available in the forward chain. This forces us to use the source-address for filtering (protected by rp-filter)

A complete sample

we will discuss the packet filter script by Rusty Russel (Do you find the problem with the ICMP user defined chain in this script?)

Testing ipchains

ipchains -C [chain] parameter action

Original rule:

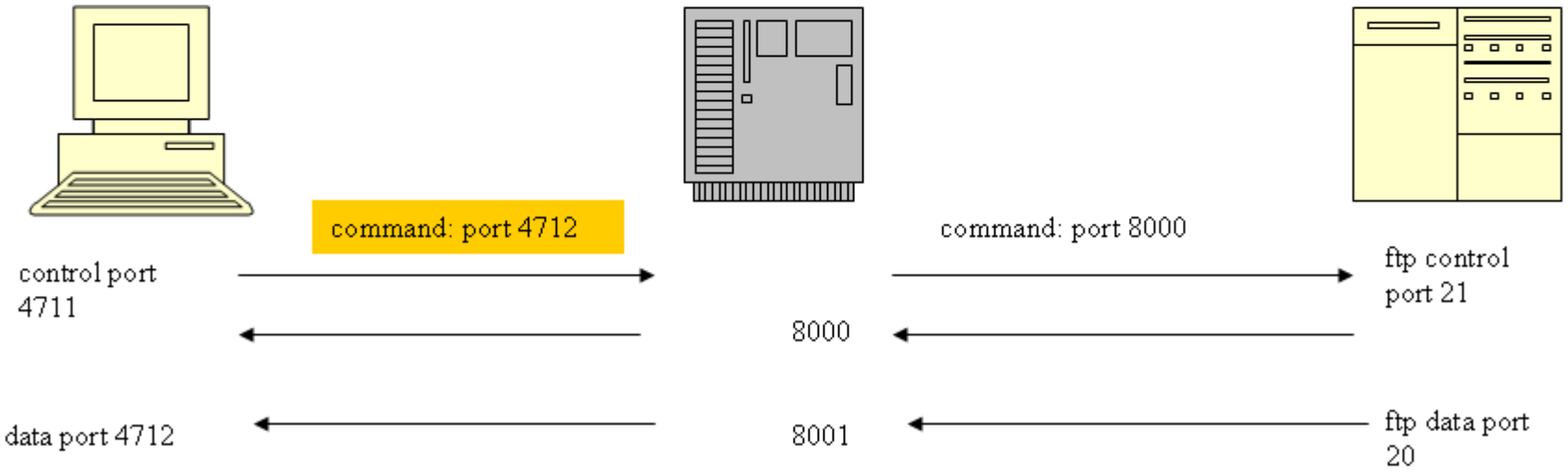
•ipchains –A output –i $EXT_IFACE –p tcp –s $MY_IPADDR $UNPRIVILEGED_PORTS – destination-port 80 –j ACCEPT

• Command to generate test packages according to above rule:

•ipchains –C output –i $EXT_IFACE –p tcp –s $MY_IPADDR 6000 – destination-port 80 – j ACCEPT

Most rules can be turned into test generators by using the –C option instead of –A or –I. Please note that port RANGES are not possible with –C. The above example should result in „accepted“.

Content filters and special protocols: ftp



The client puts the port number of its data-receiving port into a packet to the ftp-servers control port. The masquerading firewall needs to look into the data packet, detect the port number, open dummy control and data ports at the firewall and route INCOMING data connection requests to the clients data port. This is a typical example for a problematic protocol in two ways: first it hides network information in packet data – making NAT hard to perform. And secondly it expects INCOMING connection requests to be accepted. Only if the firewall tracks the connection and knows the protocol will it be able to detect that the data request from the ftp-server is really a response to a control request from the client.

Requirements for Packet Filters

- Filtering per interface
- Filtering on incoming and outgoing packets per interface
- Order of rules needs to be respected by the configuration tool – no re-ordering of rules.
- Logging of accepted and rejected packets must be possible. Logging itself needs to be configurable with respect to device and quantity (to avoid DOS by log-flooding)
- Rules need to be validated and tested.

A critique of ipchains

- No stateful filtering (no actions based on packet history)
- packets need to pass all three chains making rules complicated
- Lacking extensibility with respect to patterns and targets
- Software design in kernel needed re-design.
- Redirection of packets to user space lacking.

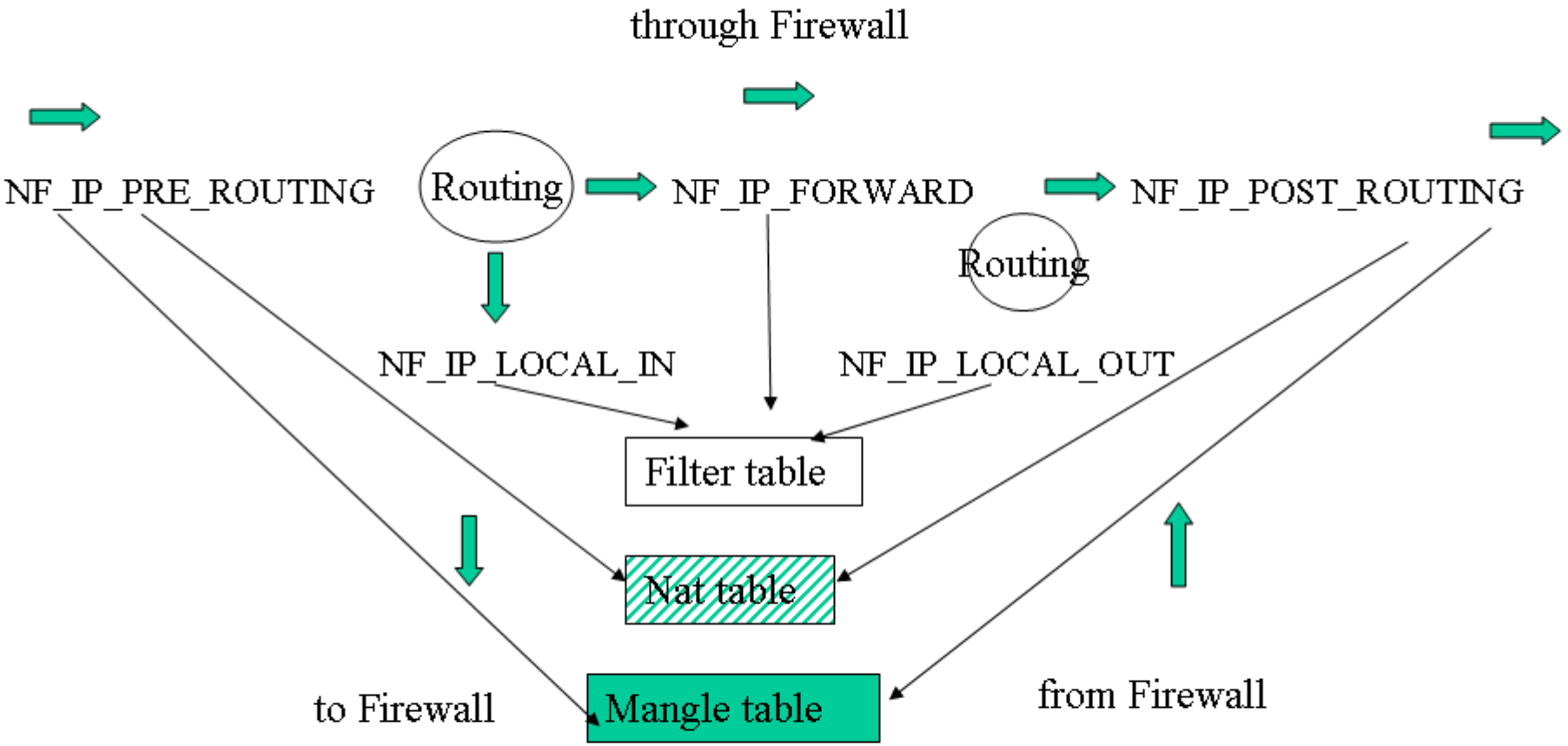
Still, ipchains is a stable and reliable mechanism to build a firewall on a general purpose Linux system. The new netfilter/iptables are of course a different game.

Differences between ipchains and iptables

- packets pass all three chains (input, forward, output)
- Only outgoing interface available in FORWARD chain
- Filtering and mangling mixed (no separate „tables“)
- Filter rules affected by masquerading (SNAT)

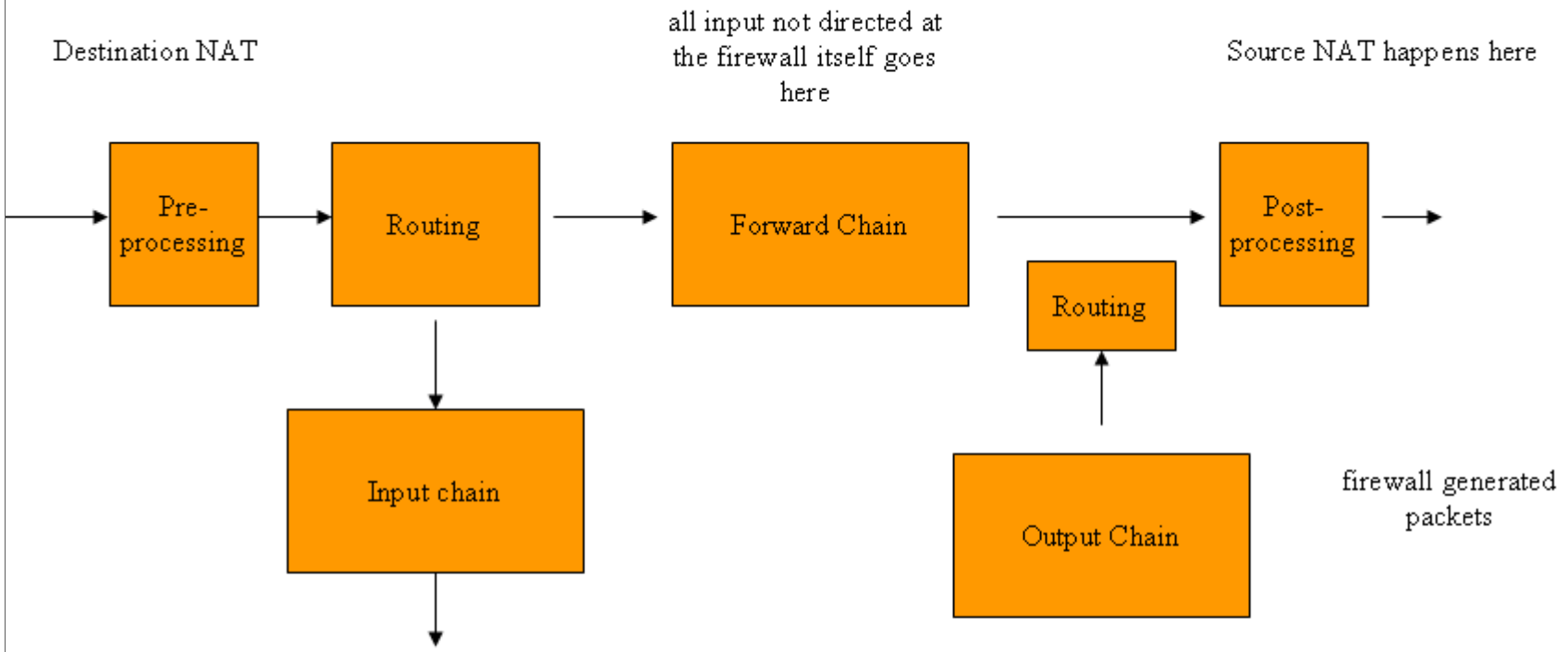
- packets pass only ONE chain
- Incoming and outgoing interfaces available in all chains
- clear separation of filtering (INPUT, FORWARD, OUTPUT) and mangling/NAT (PRE/POST-ROUTING)
- Filtering rules independent of masquerading. Rules operate always on real addresses.

Netfilter Architecture in ipv4



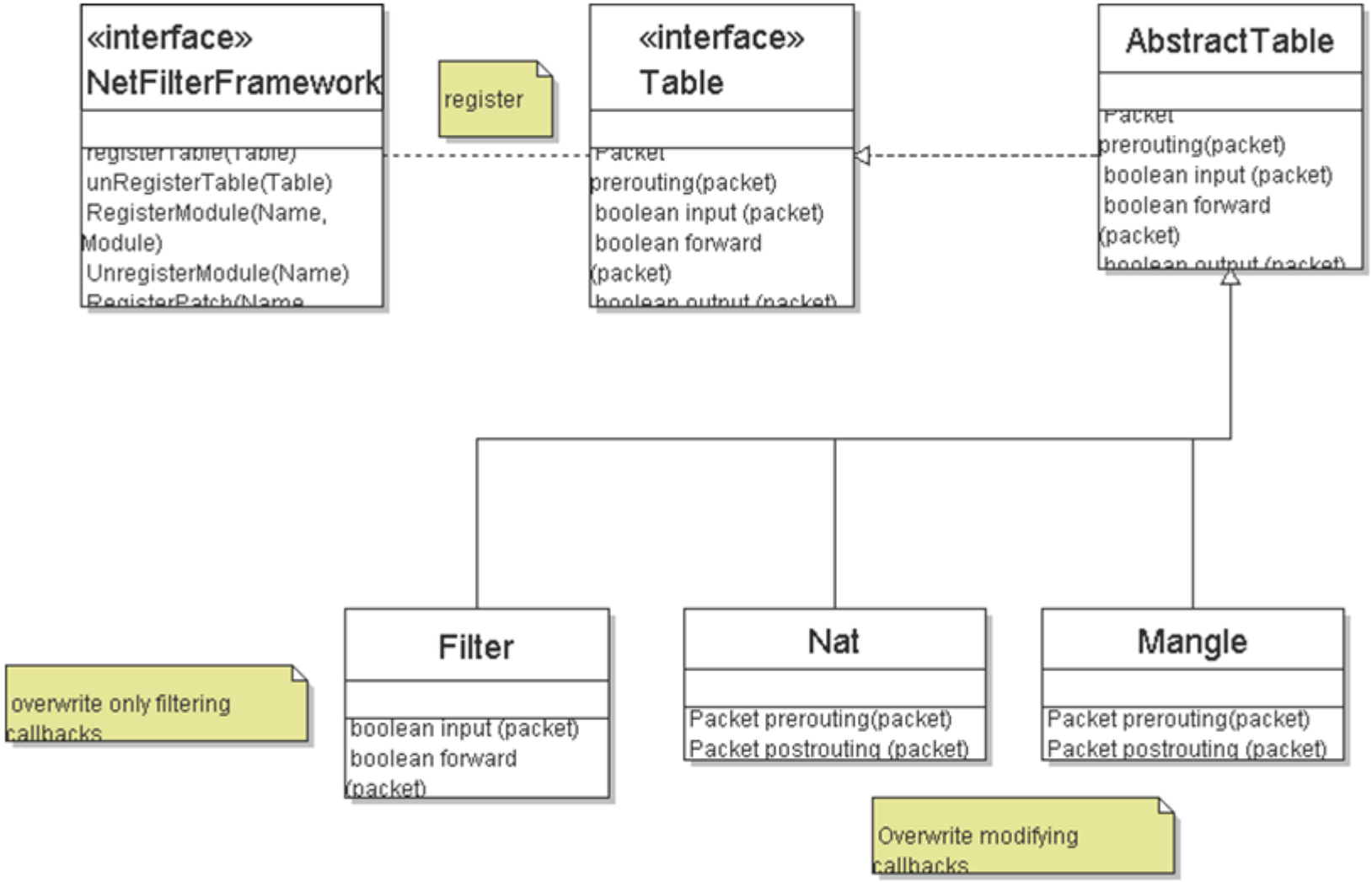
All NF_IP_xxx points are hooks where a table can register for a callback if a packet passes. The most important change compared to ipchains is that every packet in the filter tables passes ONE and ONLY ONE chain. The chains are the processors of the respective hook callback,e.g. NF_IP_LOCAL_IN events are processed in the INPUT chain of the filter table.

Default chains (iptables example)



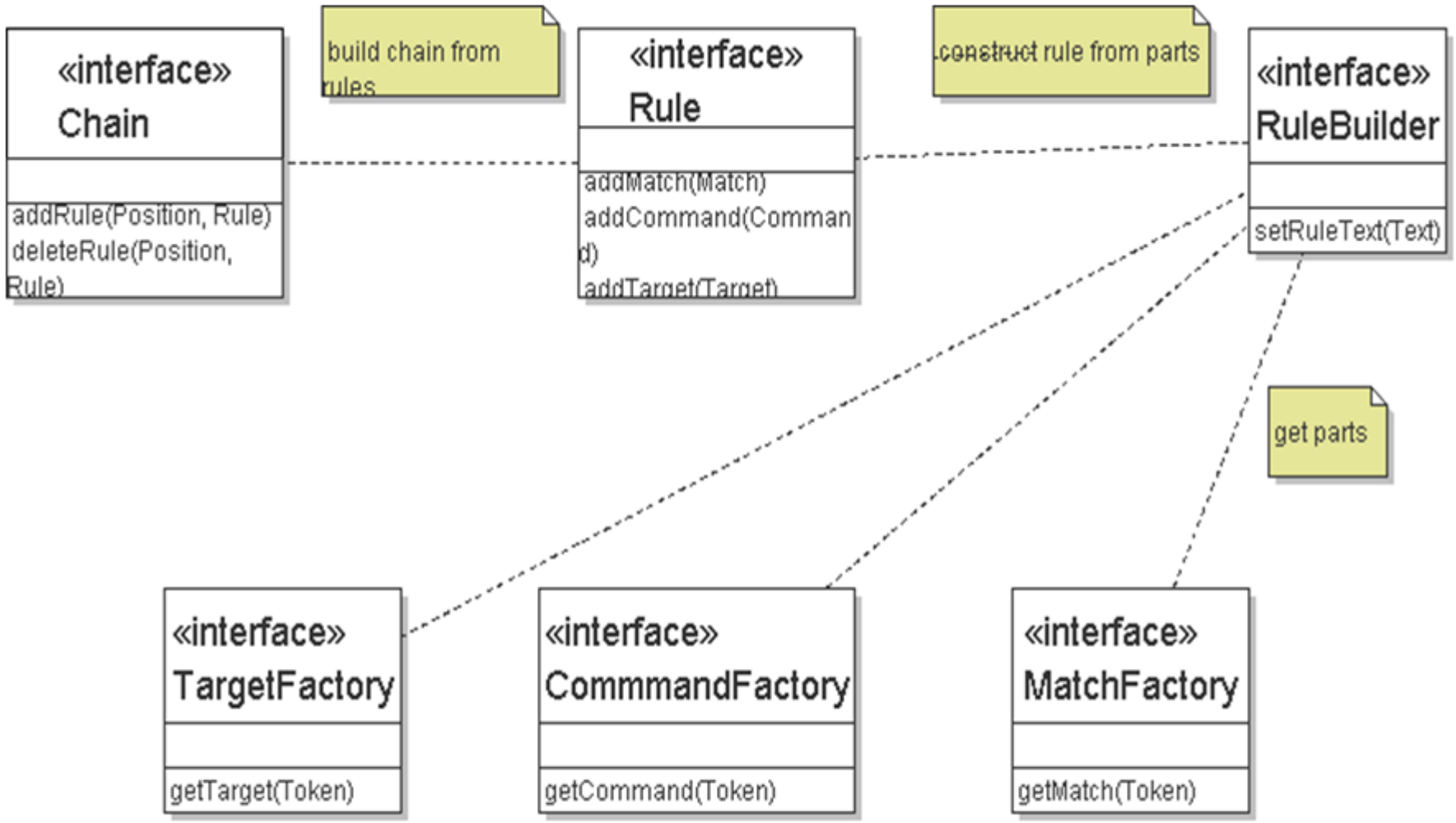
No packet modification is done in INPUT, FORWARD or OUTPUT chains. No filtering does happen in the pre- and postrouting phases: If NAT is used, only the first packet of a request would hit this chain and the rest could not be filtered here anyway.

An OO-view on the netfilter framework



Template/hook pattern as well as strategy and factory patterns could be used to design the netfilter framework. A table implements a number of hooks which will be called from the framework at the proper time.

An OO-view on the netfilter framework



Special Modules for Protection

dynamic IP addr: echo 1 > /proc/sys/net/ipv4/ip_dynaddr

(and those mentioned in the ipchains lecture!)

additional modules:

ip_tables, ip_conntrack, iptable_filter, iptable_mangle, iptable_nat, ipt_LOG, ipt_limit, ipt_state, ipt_owner, ipt_MASQUERADE, ip_conntrack_ftp, ipt_conntrack_irc

Modules are installed with /sbin/modprobe [modulname]. Especially important are the connection tracking modules.

iptables: conditions and actions

- protocol (TCP, UDP, ICMP, IGMP)
- source and destination address
- source and destination port
- TCP connection init (ACK flag)
- ICMP types
- IP fragmentation (better solved by pre-assembling fragments before checking)
- interface

- ACCEPT (let packet pass, process it and/or forward it to next chain)
- DENY (throw packet silently away, do not generate error message)
- REJECT (generate ICMP response but throw packet away)
- MASQ (perform masquerading on the packet. Move a response directly to the output chain. Used in forward chain only)
- REDIRECT (move packet to different port on local host. Can be used for transparent proxying)
- RETURN (Use default policy in a default chain or return from user defined chain)
- NAME of user-defined chain to be called.

iptables command syntax

iptables -t table -command [chain] [match] –j [target/jump]

Example:

• iptables –A INPUT –i $IFACE –p tcp –sport 80 –m state –state ESTABLISHED –j ACCEPT (allow incoming web traffic if it belongs to a previous outgoing request)

• iptables –A INPUT –i $IFACE –p tcp –sport 20 –m state –state ESTABLISHED, RELATED –j ACCEPT (allow incoming ACTIVE ftp traffic if it belongs to a previous outgoing request, even though the incoming request is for a new – but related - port)

• iptables –A INPUT –i $IFACE – p udp –j LOG –log-prefix „UDP Incoming:“

•iptables –A INPUT –i $IFACE – p udp –j DROP (log and drop all udp traffic)

The „filter“ table is the default. Other tables (nat, mangle) need to be specified explicitely. Note that most of the syntax is still the same as with ipchains.

Match Modules

-m <match module> -- <module parameter>

- Owner (gid-owner, pid-owner, sid-owner). Matches packets from specific owners. Some ICMP packets do not have an owner.
- TCP/UDP/ICMP/TOS/MARK
- mac address (interesting in connection with DHCP)
- multi-port (range of non-sequential ports)
- limit (see next pages)
- state (see next pages)

Limit

--limit hits/time

iptables – A INPUT –p icmp –icmp-type ping –m limit 10/minute –l LOG

iptables –A INPUT –p icmp –icmp-type ping –j DROP

limits how many times a rule applies per time period. E.g. do not accept and log more than 50 echo request per 5 minutes to prevent log based DOS attacks

The basic sequence of a bandwidth limiting rule pair is always: first comes a limited accept rule, followed by an unconditional DROP rule on the same pattern.

State

--state INVALID, NEW, ESTABLISHED, RELATED

iptables – A OUTPUT –m state –state NEW, ESTABLISHED, RELATED –j ACCEPT

iptables – A INPUT –m state –state ESTABLISHED, RELATED –j ACCEPT

Allows only outgoing new connections. Allows established and related connections in both directions. Requires a default deny policy installed.

The sate module does advanced connection tracking and knows e.g. that a certain UDP packet is a response to a previous outgoing UDP packet. A connection is „ESTABLISHED“ if several packets have been going back and forth – this way even connectionless UDP packets can belong to an „ESTABLISHED“ session. „RELATED“ connections are physically independent connections which nevertheless belong to an existing connection (like active ftp)

Load Balancing

-A PREROUTING -i eth0 -p tcp --dport 80 -m state --state NEW -m nth --counter 0 --every 4 --packet 0 \ -j DNAT --to-destination 192.168.0.5:80

-A PREROUTING -i eth0 -p tcp --dport 80 -m state --state NEW -m nth --counter 0 --every 4 --packet 1 \ -j DNAT --to-destination 192.168.0.6:80

Or with „random“:

-A PREROUTING -i eth0 -p tcp --dport 80 -m state --state NEW -m random --average 25 \ -j DNAT --to-destination 192.168.0.5:80

-A PREROUTING -i eth0 -p tcp --dport 80 -m state --state NEW -m random --average 25 \ -j DNAT --to-destination 192.168.0.6:80

From Barry O‘Donovan, Advanced features… (see ressources)

Patch-O-Matic

New matches:

•iplimit: restrict connections from a certain host or network

• length: filter packets based on their lenght

• nth: filter on each „nth“ packet

• string: match against a certain string in a packet

new targets:

• FTOS: sets TTL to arbitrary value

• ULOG: user space logging

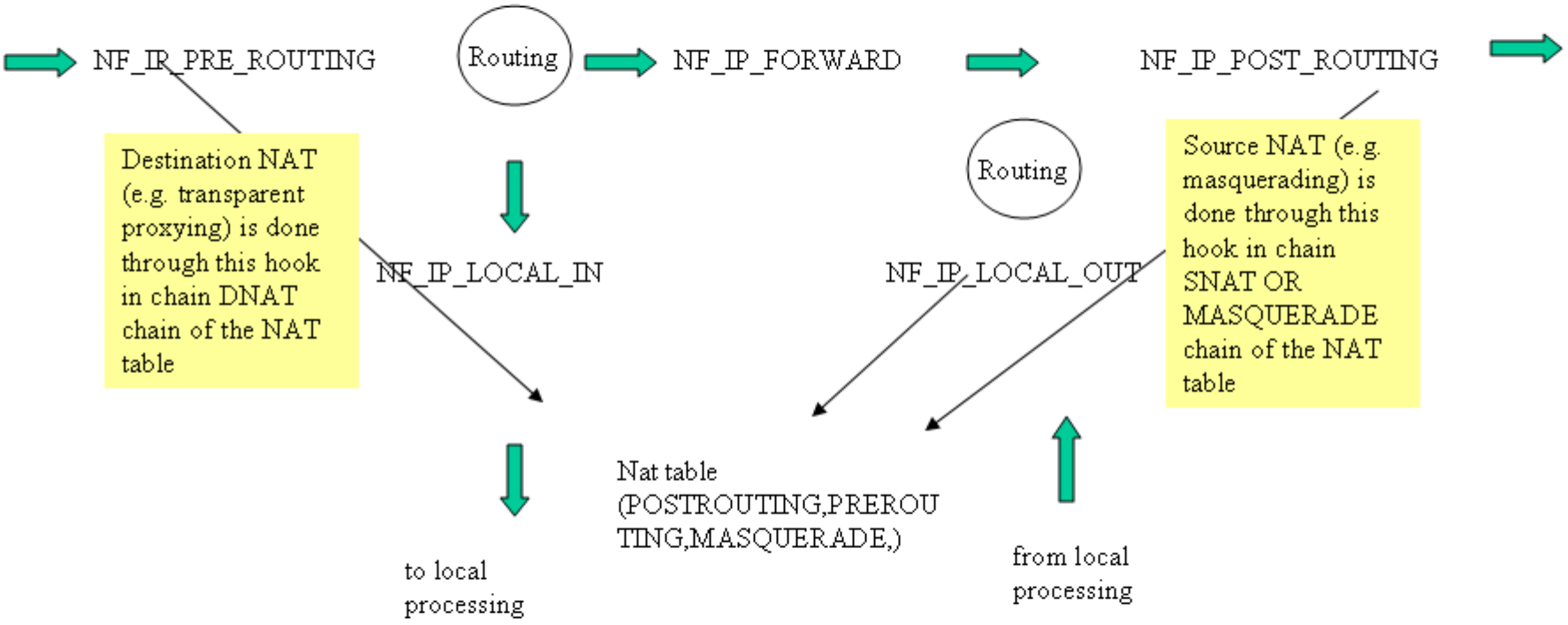
• NETMAP (SNAT like behavior)

See the netfilter extensions howto for more matches and targets. The iptables user space driver program is so flexible that it will deal automatically with new matches or targets that were installed.

Targets (actions or jumps)

- ACCEPT (ends rule processing in this chain)
- DROP (throws packet away without error message to sender)
- REJECT (same but sends error message, works only in input, forward and output chain)
- LOG (logs packet AND continues: the only target that does not stop rule processing)
- RETURN (return back from user defined chain to default chain)
- QUEUE (sends packet from kernel space (tcp stack) to user space for further processing. Comes with complete API for proxies etc.)
- REDIRECT (routes packets to a port on the local host, used e.g. for transparent proxying)
- userdefined target (subroutine like flow of control to user defined chain)
- MARK (used in mangle table to set internal routing information for this packet in the kernal)
- TOS (sets routing information within a packet, can be communicated to other firewalls or routers)
- MIRROR (experimental, exchanges source and destination address of a packet and sends it back. Watch out for spoofing attacks creating loops)
- SNAT (Source NAT, used to map internal addresses to a real IP address)
- DNAT (Destination NAT, used e.g. to forward incoming internet packets to servers on the internal (hidden) network or DMZ)
- MASQUERADE (like SNAT only with dynamic IP address)
- TTL (change time-to-live value in mangle table, e.g. to hide several machines from an ISP or to hide a firewall from a firewalking probe)
- ULOG (user defined logging facility. Multicasts packets to user space loggers)

Nat with netfilter



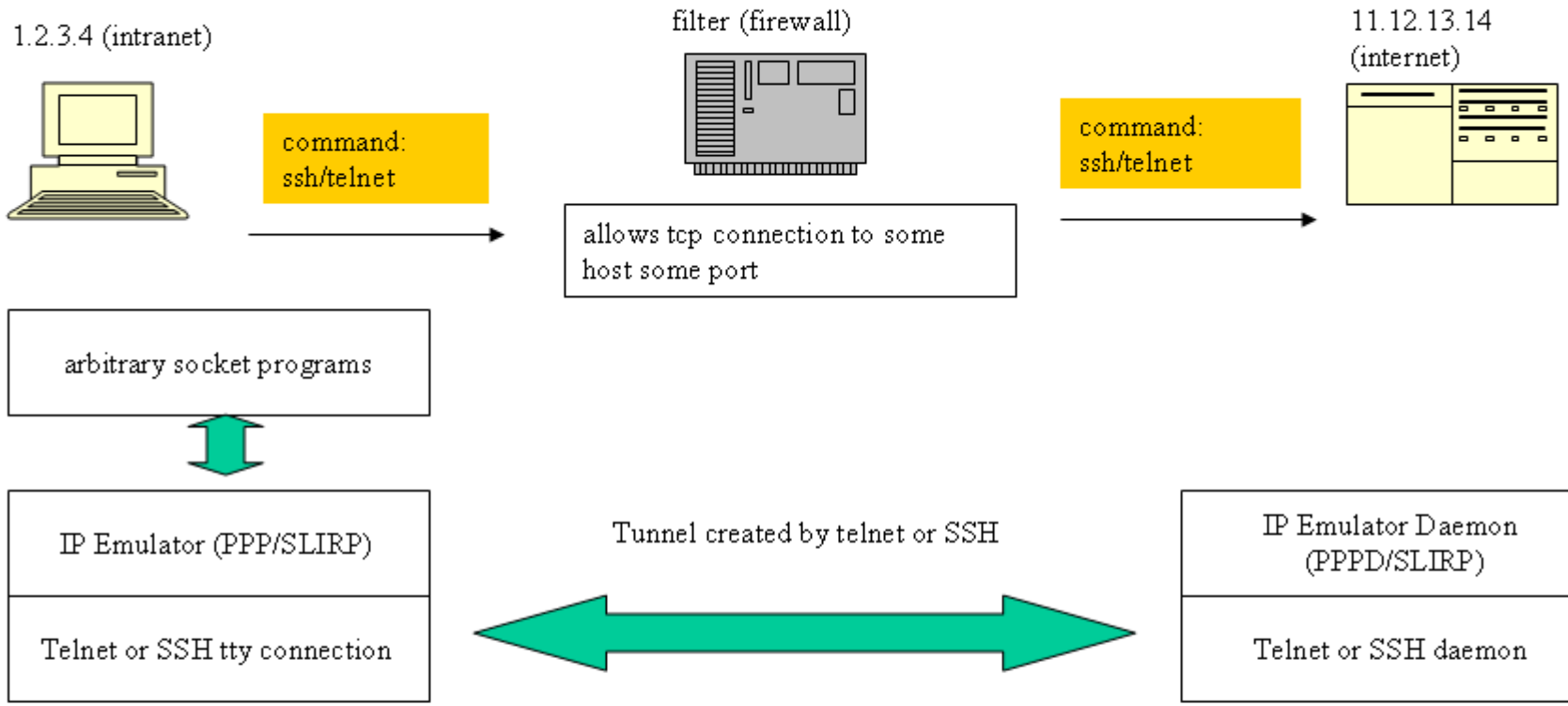
The big difference to ipchains is the fact that now all packets pass with their real source/destination address through the netfilter process and routing is based on those real addresses as well. This make logging and decision making much easier.

iptables examples

1. rc.firewall script
2. rc.DMZ.firewall script

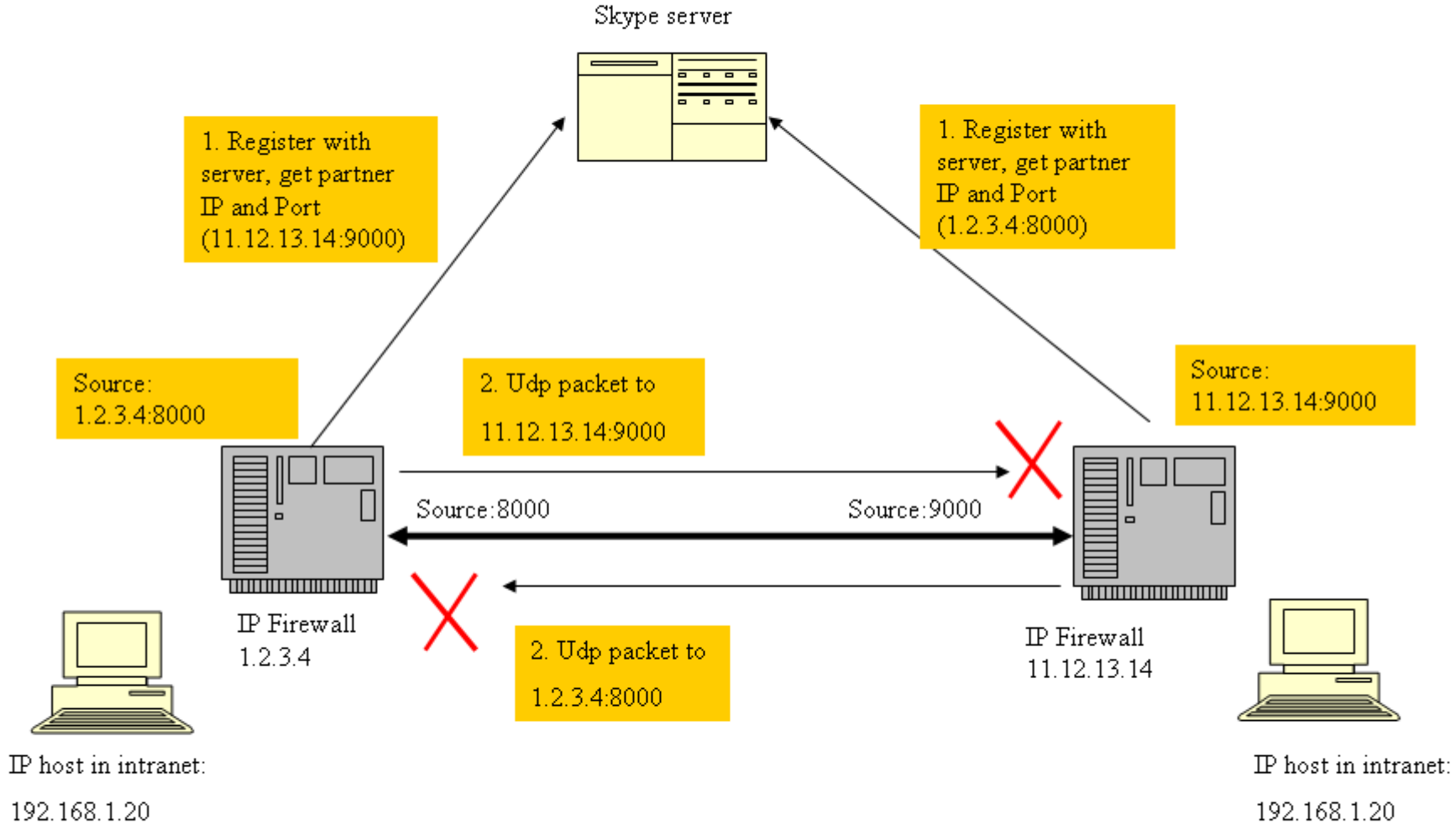
These examples are from the Iptables tutorial by Oskar Andreasson

Firewall Piercing



The only requirement for firewall piercing is: you must be able to connect to some port on a machine on the Internet. This machine runs a telnet or secure shell daemon (possibly started after receiving a mail from the client behind the firewall using procmail). On top of this tty like connection a generic IP emulator daemon emulates a complete IP connection over this tty. The client can then run arbitrary socket based programs directly through this channel into the Internet. Routing needs to be carefully configured to separate tunnel and tunnelled addresses. Works like a VPN with ssh (http://tldp.org/HOWTO/mini/Firewall-Piercing/x296.html

Firewall Piercing: How P2P Software does it



The trick is in the 2. step: by sending a upd packet to destination address:target port (which gets thrown away) the OWN firewall learns to expect packages from this address because it believes them to be a RESPONSE (see Jürgen Schmidt, the hole trick in ressources)

Resources (1)

- www.linuxguruz.org/iptables The portal for all iptables related information. Especially useful are:
- Linux iptables HOWTO http://www.linuxguruz.org/iptables/howto/iptables-HOWTO.html
- Netfilter Extensions HOWTO - Patch-O-Matic http://www.linuxguruz.org/iptables/howto/netfilter-extensions-HOWTO.html (really interesting matches and targets, e.g. load-balancers)
- Linux 2.4 Packet Filtering HOWTO http://www.linuxguruz.org/iptables/howto/packet-filtering-HOWTO.html
- Manpage of IPTABLES http://www.linuxguruz.org/iptables/howto/maniptables.html
- IPTables Tutorial http://www.boingworld.com/workshops/linux/iptables-tutorial/ (the complete tutorial on iptables)
- Iptables Basics http://www.linuxnewbie.org/nhf/intel/security/iptables_basics.html

Resources (2)

- Netfilter framework in Linux 2.4 http://www.gnumonks.org/papers/netfilter-lk2000/presentation.html (a description of the whole framework. You will understand what tables really are after reading it. Lots of template/hook design patterns.
- Iptables - What is it http://www.cs.princeton.edu/~jns/security/iptables/index.html
- **Jürgen Schmidt, The hole trick, How Skype & Co. get round firewalls http://www.heise-security.co.uk/articles/82481 Explains udp piercing used by p2p software**
- **Barry O'Donovan, Advanced Features of Netfilter/Iptables. Shows how to use NAT table and random/nth for load balancing and other matches. http://linuxgazette.net/108/odonovan.html . Look for loadbalancing and iptables for other solutions to loadbalancing.**

Resources (3)

- Craig Hunt, TCP/IP Network Administration. If you start building firewalls you will need this book if questions about routing, sockets etc. show up. Contains headers, protocols and services.
- Douglas Comer, Internetworking with TCP/IP. Code explained. A classic if you want to build your own protocol stack or want to understand how TCP REALLY works.
- Elizabeth D. Zwicky, Simon Cooper, Brent Chapman: Building Internet Firewalls (also available in German this book covers most types of firewalls and also discusses services, protocols and middleware. Use the first part as a general introduction and the rest as a dictionary if users request certain services.)
- Tobias Klein, Linux Sicherheit. Contains example script for ipchains.
- Scott Mann, Ellen L.Mitchell, Linux Security System. Use open source tools to achieve host and network security with Linux. Contains a larger part on ipchains.
- Feiner Filtern, c't 2001 Heft 26. A good explanation of the new netfilter/iptables firewall concept in Linux.

Resources (4)

- Rusty Russel, IPCHAINS-HOWTO (contains DMZ example and use of user defined chains)
- Linux Magazin 04/05 page 6: „Zecke“ from Tobias Klein. A syscall proxy which uses an existing application protocol to transport syscalls from the attacker to the victim and routes results back. Stateful inspection and application level tracking do not help as the same connection and established protocol are used