Network Security 7: Firewalls

i.g.batten@bham.ac.uk

What is a firewall?

- A device which looks at packet **headers** and blocks or passes them based on some policy.
  - Not based on packet contents
  - Attacker controls the entire packet
    - You cannot assume that the header is only constructed by the good guys

Where do firewalls live?

- On routers
- On dedicated “firewalls” which are routers with a different/bigger software load
- On computers acting as routers
- On computers looking after themselves
- On computers acting as dedicated firewalls
  - Distinction between router and computer is generality of operating system and application mix

What can firewalls do?

- Keep unwanted packets away from operating systems and applications
- Limit (imperfectly) which machines can send packets
- Keep unusual packets away from computers

What can firewalls **not** do?

- Detect Malware / Trojans / Viruses / Etc
- Detect mis-use of legitimate protocols
- Stop attackers from crafting packets themselves
- Detect the malice or otherwise of packets
- Firewalls are not magic: “I have got a firewall” is not a protection against evil.

What can a firewall do which an application or operating system cannot?

- **Nothing**
- This is very important: a firewall can only look at packets and take decisions, a job which every operating system does as those packets arrive

So why do we have firewalls?

- Set policy at borders
  - Take some decisions out of computers' hands
- Use hardened operating systems with smaller attack surfaces
- Use simpler code than the full IP stack to "ride shotgun"
- Rate limiting to protect under-powered computers
- Pay homage to the concerns, and standards, of the 1990s (PCI-DSS, in particular)

Operation of a firewall

- Receive a packet
- Look at the headers (particularly the source IP, the destination IP and the destination port)
- Optionally update some state (“stateful”)
- Pass or reject the packet

TCP State: Revision

- A connection is proposed by the client sending a SYN with a proposed sequence number for the client's data.
- The server sends a packet with a SYN, a proposed sequence number for the server's data, and an acknowledgement of the client's SYN.
  - Alternatively, an RST to refuse the connection
- The client sends an ACK.
- SYN – SYN/ACK – SYN: the "three way handshake"

TCP State: Revision

- Once we are communicating data, each side sends data with a sequence number, and we expect acknowledgements which are up to and including the last byte.
  - We send 1500 bytes starting at sequence number 20000, we expect acknowledgments up to 21500, but not later)
  - We expect acks to be monotonically increasing

TCP State: Revision

- When a connection is finished with, FIN is sent.
- The other side ACK's the FIN, and passes an indication to the application.
- After possibly sending more data, a FIN is sent.
- And finally ACK'd.
- FIN – ACK – FIN – ACK, four packets to close a connection

Connection State

- If we have seen a SYN go out, we expect a packet with a SYN and an ACK, or a packet with an RST, to come back, and nothing else.
- Once data is being transferred, we expect data and ACKs to be in rough step with each other
- We can similarly check the progress of FIN

Typical implementations

- Solaris / OSX <10.8 / FreeBSD / etc “ipf”
- OpenBSD / OSX >=10.8 / Solaris >= 11 “pf”
- Extra code in either the ethernet interface or just behind it, which passes TCP packets to a set of state machines

Sample Fragment

```
block return-rst in log first level local1.info quick \
    on mailprod0 proto tcp from any to 147.188.192.248/30 \
    port = 25 flags S/SA head 101
pass in quick from 147.188.128.54/32 to any keep state group 101
pass in quick from 147.188.128.127/32 to any keep state group 101
pass in quick from 147.188.128.129/32 to any keep state group 101
pass in quick from 147.188.128.219/32 to any keep state group 101
pass in quick from 147.188.128.221/32 to any keep state group 101
pass in quick from 147.188.192.250/32 to any keep state group 101
pass in quick from 147.188.192.249/32 to any keep state group 101
```

Sample Fragment

```
block return-rst in log first level local1.info quick \
    on mailprod0 proto tcp from any to 147.188.192.248/30 \
    flags S/SA head 102
# 5877 is obsolete clone of 587,
# 993 is obsolete imaps prior to use of STARTTLS verb
# block return-rst in quick from any to any port = 993 keep state group 102
block return-rst in quick from any to any port = 5877 keep state group 102
pass in quick from any to any port = 587 keep state group 102
pass in quick from any to any port = 993 keep state group 102
pass in quick from any to any port = 53 keep state group 102
pass in quick from any to any port = 143 keep state group 102
pass in quick from any to any port = 80 keep state group 102
pass in quick from any to any port = 443 keep state group 102
pass in quick from any to any port = 22 keep state group 102
pass in quick from 128.204.195.144 to any port = 2010 keep state group 102
pass in quick from 128.204.195.144 to any port = 2007 keep state group 102
pass in quick from 128.204.195.144 to any port = 2009 keep state group 102
pass in quick from 128.204.195.144 to any port = 2003 keep state group 102
```

State is complex

src/dst

seq #

```
client-8-41.eduroam.sxuni.org.uk -> mail.barsten.eu.org pass 0x40008502 pr 6 state 4/6
        tag 0 ttl 11997
        49341 -> 143 2b8fa4fa:adc9c1 65152<<5:63336<<1
cmsk 0000 smsk 0000 s0 2b8f6540/00ad83d2
FWD:ISN inc 0 sumd 0
REV:ISN inc 0 sumd 0
forward: pkts in 147 bytes in 23970 pkts out 0 bytes out 0
backward: pkts in 0 bytes in 0 pkts out 108 bytes out 24279
pass in quick keep state        IPv4
pkt_flags & 0(10000) = 1000,            pkt_options & ffffffff = 0, ffffffff = 0
pkt_security & ffff = 0, pkt_auth & ffff = 0
is_flx 0x1 0 0 0x1
interfaces: in @[mailprod0],@[] out @[],@[mailprod0]
```

Value of this?

- In principle, implementations should discard out-of-state packets anyway
- Concern is Denial of Service and untested code-paths

Denial of Service

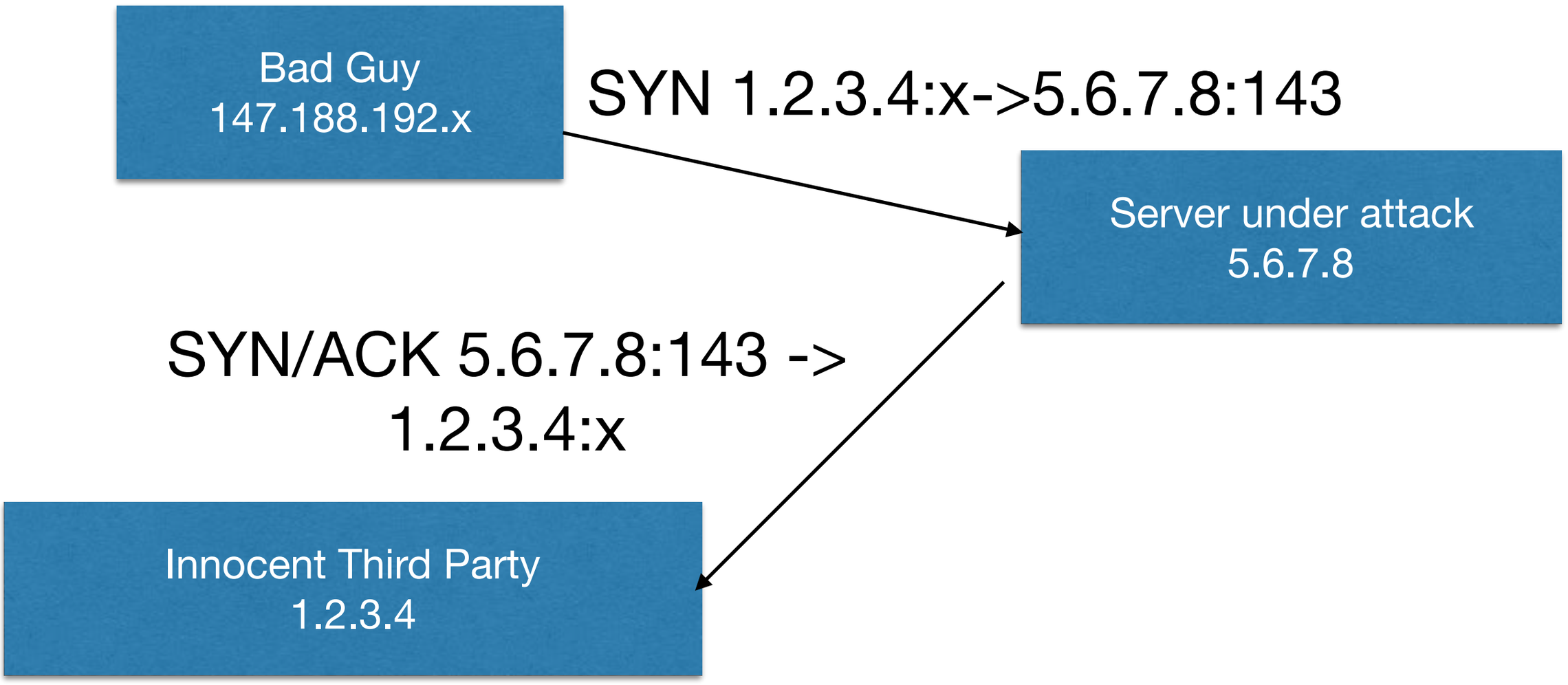
- Send a host a packet proposing a new connection to the IMAP server
  - 1.2.3.4:12345 -> 5.6.7.8:**143**, SYN 43279
- Assuming there is an IMAP server running, the server software is not informed of this until the three-way handshake is finished
- OS however acknowledges the SYN (SYN/ACK), generates its own sequence number, sets up a protocol control block and awaits the expected ACK from the client.
- “Client” is an attacker, forging packets.

Impact #1

- The server that is receiving the requests runs out of kernel memory, because it keeps on setting up control blocks which are never used.
- Might take minutes for them to be reclaimed (spec allows 2 minutes for connection to set up)
- Attacker can set up a lot of control blocks in two minutes (potentially hundreds per second).

Impact #2

- An innocent third party might get a high rate of SYN-ACKs from the server under attack and be unable to determine real source



Firewalls can stop this

- Connection tracking allows it to count number of outstanding connection requests that have not been acknowledged
- Can set absolute thresholds, rates, etc
- Could be retrofitted to OS kernels, but would be a risk: sometimes this is (briefly) reasonable behaviour on a LAN

Dealing with SYN floods

- Still a common DoS by unsophisticated attackers

The problem

- Attacker constructs a packet with a SYN for a common port that will be accepted by firewalls, and a fake source address
- Server allocates a PCB and buffers on arrival of each SYN, and runs out of memory.
- SYN/ACK floods innocent third party: two for the price of one
- Fake source address can be a trusted source (so attack port 25 with fake MessageLabs address).
- No need to control source address as attacker does not need (indeed, does not want) the responses

Firewall Solutions

- State tracking to limit the maximum number of incomplete connections per host
- Rate limiting to limit the number of new connections (probably a good idea anyway)

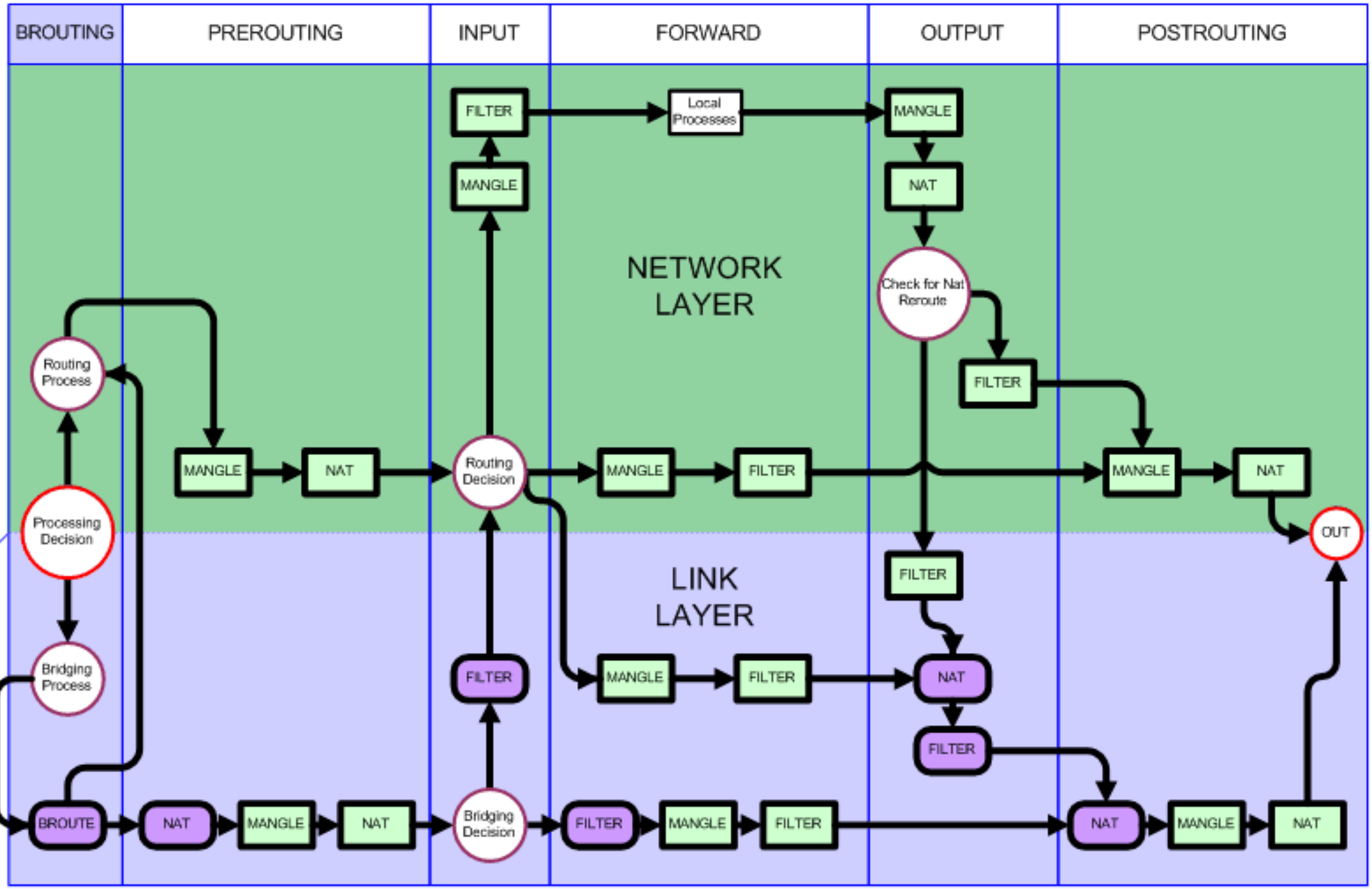
Host solutions

- Timeout the PCBs more quickly
  - Breaks standards
- Defer allocation of full PCB until receipt of ACK of the SYN/ACK
  - Complex, invasive changes to kernel
  - Will introduce delay into every connection

Firewalls on Linux

- Linux Firewalls are much more complex, as they sit in the IP stack, not in the ethernet drivers.
- The firewall code is called at a variety of points during the progress of a packet through the kernel

I say complex…



Slightly Simpler

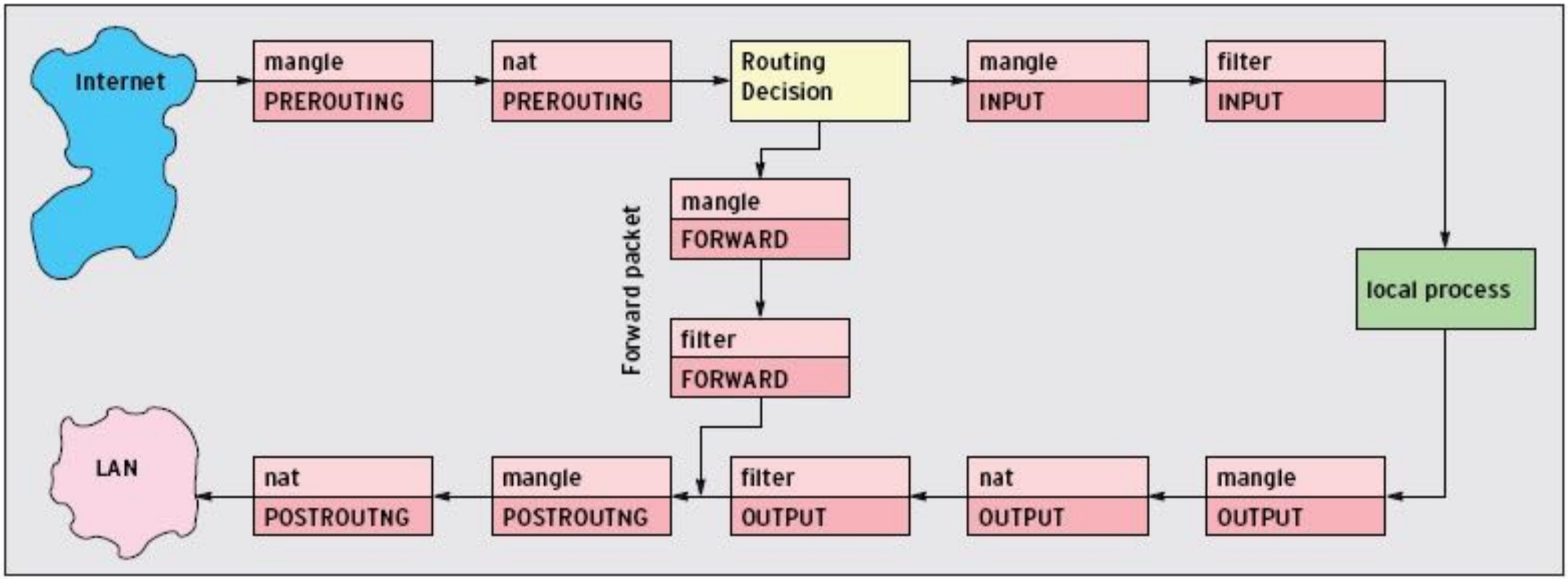
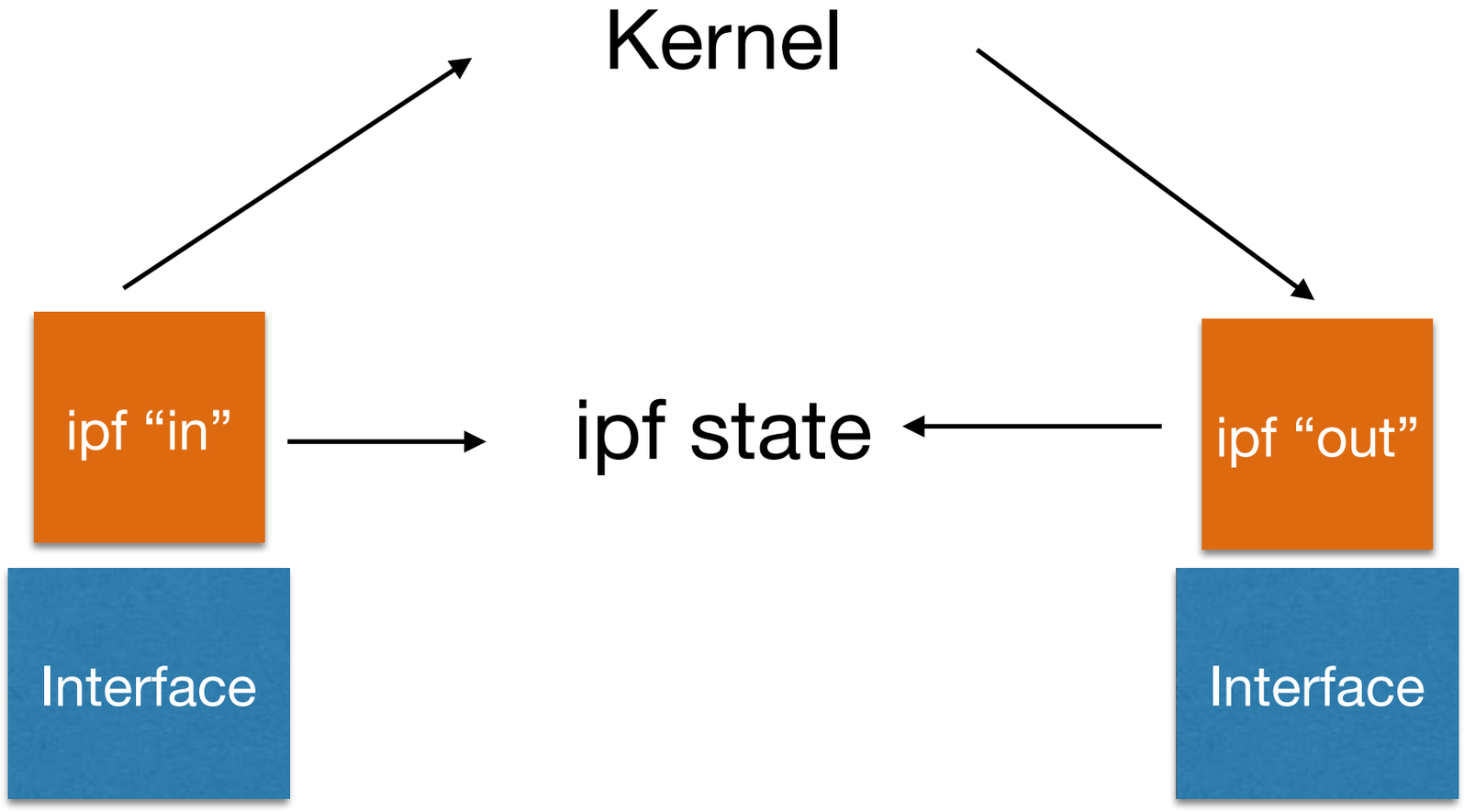


Figure 3: With Iptables, incoming packages first pass through the mangle table in the PREROUTING chain; the NAT table then follows. If the packet is destined for another host, the packet is simply passed to the mangle table in the POSTROUTING chain.

Contrast with…



Linux iptables

- Appears by name in Linux machines
- Architecture is borrowed, usually with implementation, for cheap routers that run Linux
  - For example, Mikrotik
- Sometimes with an added dash of ipchains, the earlier Linux implementation

Linux iptables

- Mixture of:
  - “NAT” rulesets, which map addresses between internal and external (or whatever)
  - “mangle” rulesets, which can modify packets in a variety of (unexpected) ways and add marks for later use
  - “filter” rulesets, which can drop or accept packets

Linux iptables

- Called
  - When packets arrive (“pre routing”)
  - When packets leave (“post routing”)
    - When packets are sent up to applications (“input”)
    - When packets come down from applications (“output”)
    - When packets are pass through by the machine acting as a router (“forward”)
- A given packet will go through exactly one of input, output or forward

So…

- We can mangle, and then filter, packets in many different places
- And the packets will be in varying stages of processing at each point.
- Effective architectures use **marks**

Input v Forward

- The split between “input” and “forward” can be confusing.
- By the time either is called, the decision has been taken whether to pass to application or send out of another interface.
- For a machine doing routing, the “input” policy on the “outside” network should probably be “block everything”. Remote management of routers is unwise.

Practical Examples

- Simple enterprise firewall
- Policy is basically “drop everything we don’t explicitly need”
- Accepts various VPN protocols, ssh, and logging from places it expects logging from.
- Everything else goes on the floor

Pre-Routing Mangle

```
[igb@rb2011-1] /ip firewall mangle> /ip firewall mangle print
where chain=prerouting
Flags: X - disabled, I - invalid, D - dynamic
 0    chain=prerouting action=jump jump-target=typecheck
protocol=!tcp in-interface=pppoe-aa-uplink log=no log-
prefix=""

 1    ;;; Select on WAN interface and don't bother running
these tests on other packets
      chain=prerouting action=jump jump-target=pppoa-groom in-
interface=pppoe-aa-uplink log=no log-prefix=""
[igb@rb2011-1] /ip firewall mangle>
```

Checking IP Types

```
[igb@rb2011-1] /ip firewall mangle> /ip firewall mangle print where
chain=typecheck
Flags: X - disabled, I - invalid, D - dynamic
 0 X  chain=typecheck action=return protocol=tcp log=no log-prefix=""

 1    chain=typecheck action=return protocol=udp log=no log-prefix=""

 2    chain=typecheck action=return protocol=icmp log=no log-prefix=""

 3    chain=typecheck action=return protocol=ipsec-ah log=no log-
prefix=""

 4    chain=typecheck action=return protocol=ipsec-esp log=no log-
prefix=""

 5    chain=typecheck action=log log=no log-prefix="bad protocol"

 6    chain=typecheck action=mark-packet new-packet-mark=bogus
passthrough=yes log=no log-prefix=""
[igb@rb2011-1] /ip firewall mangle>
```

Checking and Limiting

```
[igb@rb2011-1] /ip firewall mangle> /ip firewall mangle print where chain=pppoa-groom
Flags: X - disabled, I - invalid, D - dynamic
 0    ;;; Mark bogus (RFC1918 etc) source input packets as bogus
      chain=pppoa-groom action=mark-packet new-packet-mark=bogus passthrough=yes src-address-list=bogons
log=no log-prefix=""
 1    ;;; Bless AH
      chain=pppoa-groom action=mark-packet new-packet-mark=blessed passthrough=no protocol=ipsec-ah log=no
log-prefix=""
 2    ;;; Bless ESP
      chain=pppoa-groom action=mark-packet new-packet-mark=blessed passthrough=no protocol=ipsec-esp log=n
log-prefix=""
 3    ;;; TCP PSD
      chain=pppoa-groom action=mark-packet new-packet-mark=bogus passthrough=no protocol=tcp psd=21,3s,3,1
log=no log-prefix=""
 4    ;;; UDP PSD
      chain=pppoa-groom action=mark-packet new-packet-mark=bogus passthrough=no protocol=udp psd=21,3s,3,1
log=no log-prefix=""
 5    ;;; Mark all packets in invalid states as bogus
      chain=pppoa-groom action=mark-packet new-packet-mark=bogus passthrough=no connection-state=invalid
log=no log-prefix=""
 6    ;;; Drop loose source routing
      chain=pppoa-groom action=mark-packet new-packet-mark=bogus passthrough=no ipv4-options=loose-source-
routing log=no log-prefix=""
 7    ;;; Drop strict source routing
      chain=pppoa-groom action=mark-packet new-packet-mark=bogus passthrough=no ipv4-options=strict-source
routing log=no log-prefix=""
[igb@rb2011-1] /ip firewall mangle>
```

Input Ruleset

```
0    ;;; Accept everything for input chain that has not come from outside
     chain=input action=accept in-interface=!pppoe-aa-uplink log=no log-prefix=""

1    ;;; Drop packets with the bogus tag from pre-routing chain
     chain=input action=drop packet-mark=bogus log=yes log-prefix="bogus on input:"

2    ;;; Accept blessed packets
     chain=input action=accept packet-mark=blessed log=no log-prefix=""

3    ;;; Rate limit ICMP
     chain=input action=accept protocol=icmp in-interface=pppoe-aa-uplink dst-limit=1,5,dst-address/
1m40s log=no log-prefix=""

 4    ;;; Accept established connections
      chain=input action=accept connection-state=established in-interface=pppoe-aa-uplink log=no log-
prefix=""

 5    ;;; Accept protocols we use, with appropriate limits (rate of creation, total connections)
      chain=input action=accept connection-state=new protocol=tcp in-interface=pppoe-aa-uplink dst-
port=443,1194 limit=1,5 log=no
      log-prefix=""

 6    ;;; Accept protocols that we use, with rate limit on packets from unknown sources
      chain=input action=accept connection-state=new protocol=udp in-interface=pppoe-aa-uplink dst-
port=1194,500,5678,1701,4500 log=no
      log-prefix=""

 7    ;;; And drop everything else
      chain=input action=drop in-interface=pppoe-aa-uplink log=yes log-prefix="default input:"
```

Forward Ruleset (1)

```
[igb@rb2011-1] /ip firewall mangle> /ip firewall filter print where chain=forward
Flags: X - disabled, I - invalid, D - dynamic
 0    ;;; Accept everything that is not either in or out of the pppoa link
      chain=forward action=accept in-interface=!pppoe-aa-uplink out-interface=!pppoe-aa-uplink log=no
log-prefix=""

 1    ;;; Forward everything outbound
      chain=forward action=accept out-interface=pppoe-aa-uplink log=no log-prefix=""

 2    ;;; Drop everything in the ossec-list
      chain=forward action=drop src-address-list=ossec-list in-interface=pppoe-aa-uplink log=no log-
prefix=""

 3    ;;; Forward all inbound established traffic
      chain=forward action=accept connection-state=established in-interface=pppoe-aa-uplink log=no log-
prefix=""

 4    ;;; Drop packets with bogus mark from pre-routing
      chain=forward action=drop packet-mark=bogus log=yes log-prefix="bogus on forward:"

 5    ;;; [ Weird and not interesting, deleted ]

 6    ;;; Pass blessed packets from pre-routing
      chain=forward action=accept packet-mark=blessed log=no log-prefix=""
```

Forward Ruleset (2)

```
 7    ;;; Forward new https connections (mostly VPN) with rate limiting
      chain=forward action=accept connection-state=new protocol=tcp dst-address-list=https-
servers in-interface=pppoe-aa-uplink dst-port=443
      dst-limit=1,5,src-and-dst-addresses/1h log=no log-prefix=""

 8    ;;; Permit ssh connections to appropriate places
      chain=forward action=accept connection-state=new protocol=tcp src-address-list=trusted-
sources dst-address-list=ssh-servers
      dst-port=22 dst-limit=3/1m,1,src-and-dst-addresses/1m40s log=no log-prefix=""

 9    ;;; OSSEC to pi-two and ossec-sol
      chain=forward action=accept protocol=udp dst-address-list=ossec-sink in-interface=pppoe-
aa-uplink dst-port=1514 log=no log-prefix=""

10    ;;; tcp 514c for syslog to pi-two
      chain=forward action=accept connection-state=new protocol=tcp dst-address=81.187.150.213
src-address-list=offsite-locations
      in-interface=pppoe-aa-uplink dst-port=514 log=no log-prefix=""

11    chain=forward action=accept protocol=udp dst-address=81.187.150.213 src-address-
list=offsite-locations in-interface=pppoe-aa-uplink
      dst-port=514 log=no log-prefix=""

12    ;;; Drop inbound traffic that is out of state or otherwise unwanted
      chain=forward action=drop in-interface=pppoe-aa-uplink log=yes log-prefix="out of state on
forward"

[igb@rb2011-1] /ip firewall mangle>
```