Firewalls

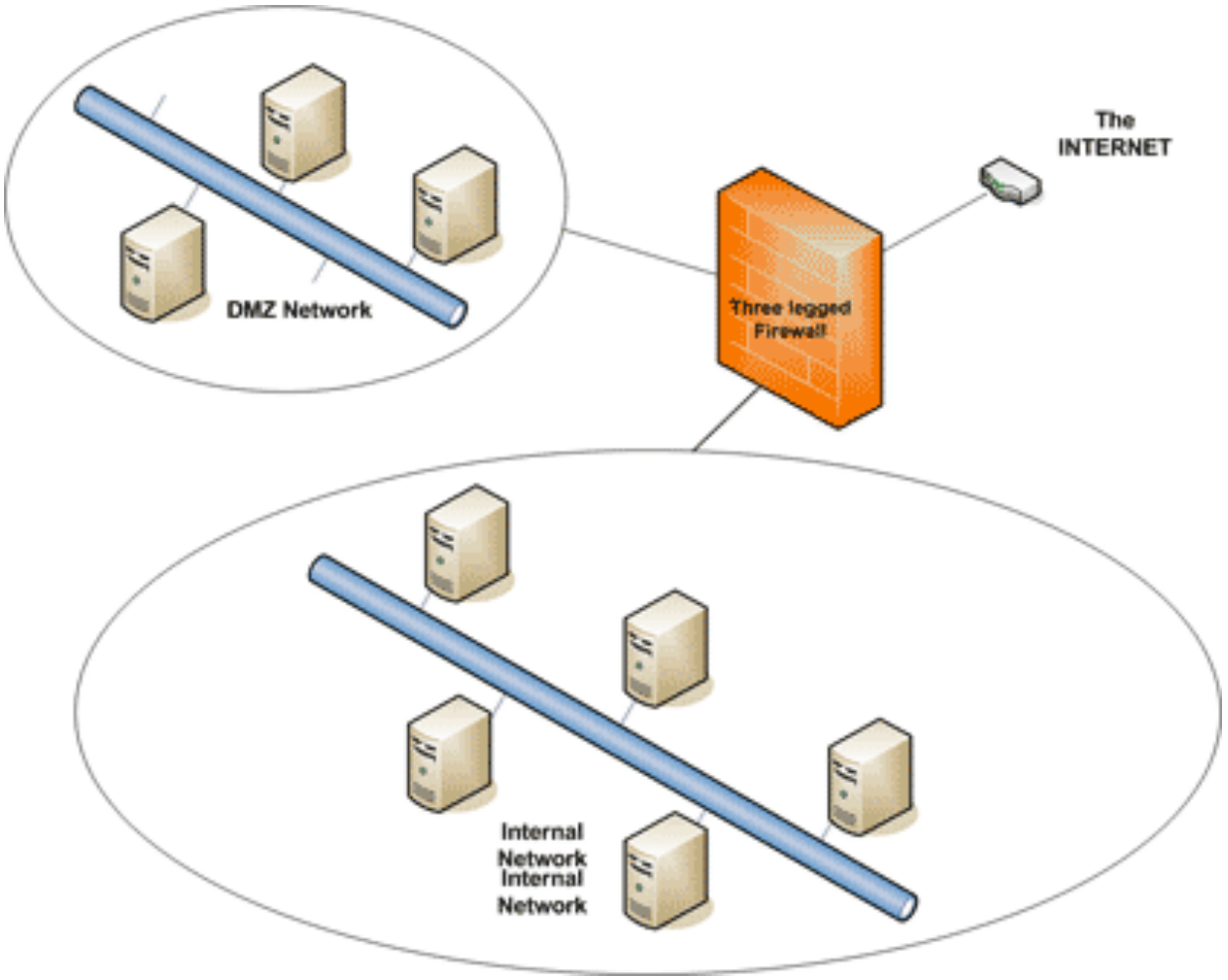
jnlin

Firewalls

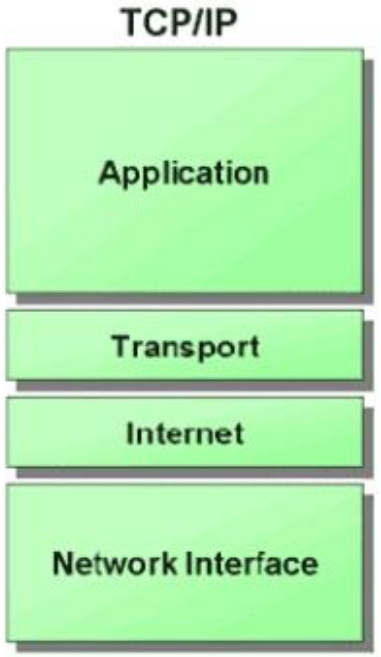
- Firewall
  - hardware/software
  - choke point between secured and unsecured network
  - filter incoming and outgoing traffic
  - prevent communications which are forbidden by the security policy

- What it can be used to do
  - Incoming: protect and insulate the applications, services and machines
    - Such as telnet, NetBIOS
  - Outgoing: limit or disable access from the internal network
    - Such as MSN, ssh, ftp, facebook, SC2, D3
  - NAT (Network Address Translation)

Typical Network Design



Firewalls – Capabilities



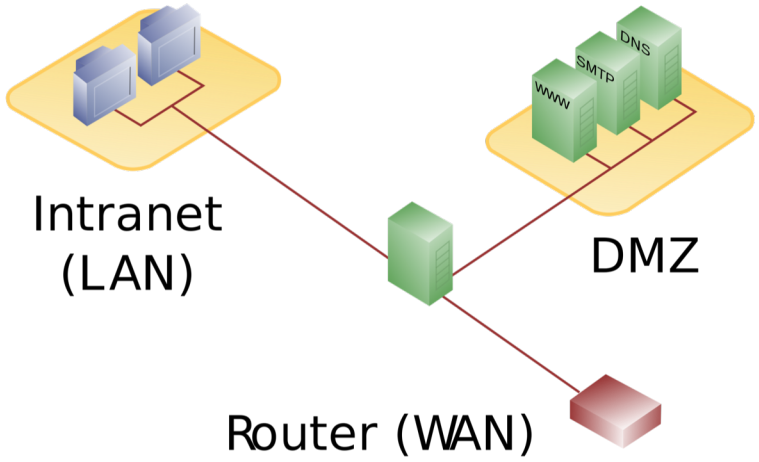
- Network Layer Firewalls
  - Operate at a low level of TCP/IP stack as IP-packet filters.
  - Filter attributes
    - Source/destination IP
    - Source/destination port
    - TTL
    - Protocols
    - ...
- Application Layer Firewalls
  - Work on the application level of the TCP/IP stack.
  - Inspect all packets for improper content, a complex work!
- Application Firewalls
  - The access control implemented by applications.
  - TCP Wrapper (libwrap)

Firewalls – Rules

- Exclusive
  - Only block the traffic matching the rulesets
- Inclusive
  - Only allow the traffic matching the rulesets
  - Offer much better control of the incoming/outgoing traffic
  - Safer than exclusive one
    - (Y) reduce the risk of allowing unwanted traffic to pass
    - (N) increase the risk to block yourself with wrong configuration
- State
  - Stateful
    - Keep track of which connections are opened through the firewall
    - Be vulnerable to Denial of Service (DoS) attacks
  - Stateless

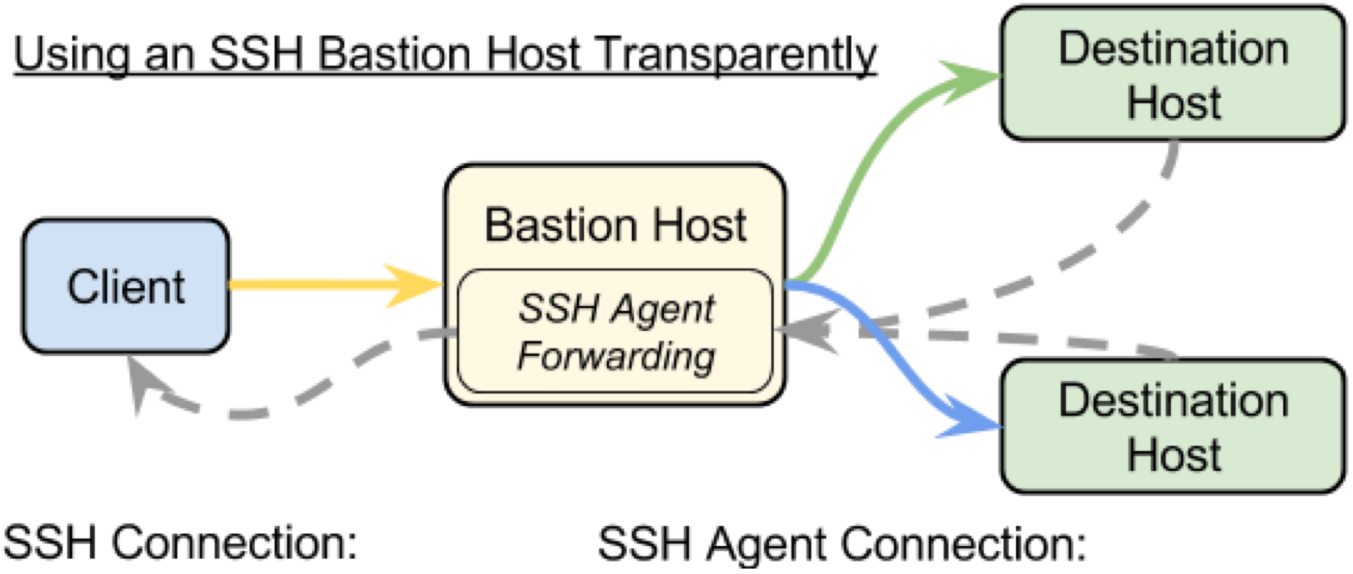
Firewalls – DMZ

- Demilitarized zone (Perimeter Network)
  - Between untrusted and trusted networks
  - Limited access to internal networks
  - Open service to WAN (Internet)
    - SMTP
    - POP3
    - HTTP
    - VPN Servers
    - …
- A layer of security
  - Limit the damage if system is compromised



Firewalls – Bastion Host

- A workstation allow users connect to internal service
  - Limit the entry point of the internal network
  - Do logging and auditing on it
  - Located in DMZ or behind VPN service



Firewalls – Packages

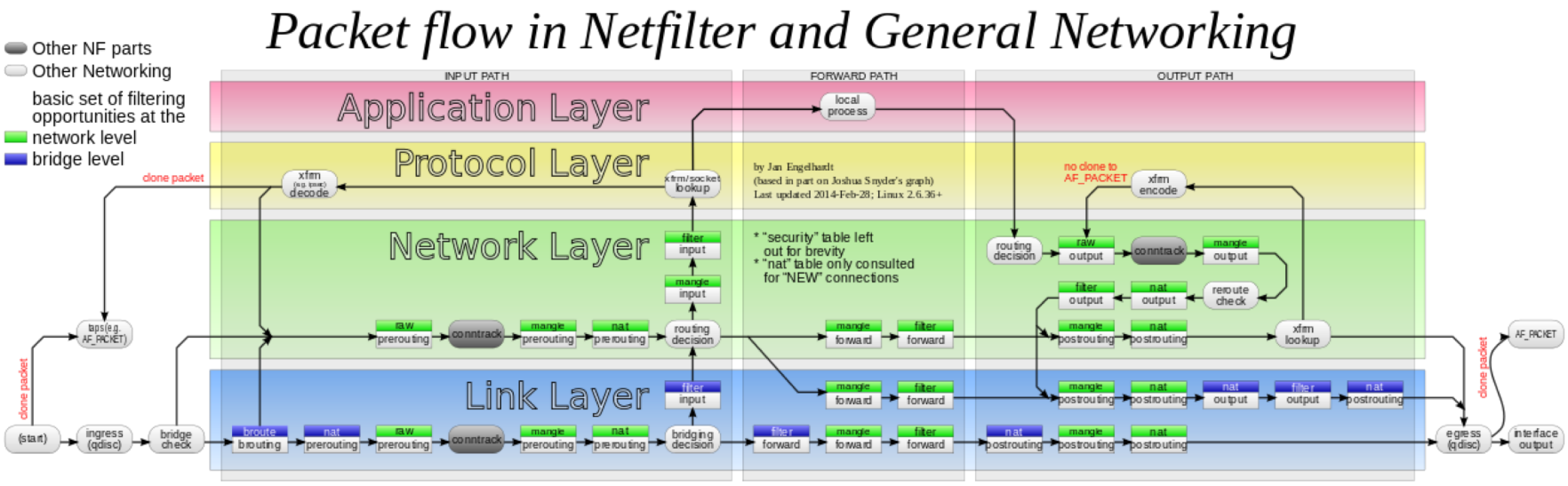
- Linux
  - iptables (kernel 2.4+)
  - ipchains (kernel < 2.4)
  - Firewalld
  - ufw
- FreeBSD
  - IPFILTER (known as IPF)
  - IPFIREWALL (known as IPFW) + Dummynet
  - *Packet Filter (known as PF)+ ALTQ*
    - migrated from OpenBSD
    - v4.5 (In FreeBSD 9.0)
    - http://www.openbsd.org/faq/pf/ v5.0

iptables in Linux

iptables

- User-space software that control Linux kernel firewall
  - Control Linux kernel Netfilter modules
- Support kernel version 2.4+
  - Replace ipchains and ipfwadm
- iptables allows system administrators to define *tables* containing *chains* of *rules* for the treatment of packets

Packet flow in Netfilter



Xtables Architecture

- Xtables
  - v4, v6, arp, eb
  - IPv4, IPv6 are different tables
- Tables
  - filter, nat, mangle
- Chains
  - PREROUTING, OUTPUT, FORWARD, INPUT, POSTROUTING
- Rules
  - e.g., iptables -A INPUT -i lo -j ACCEPT

Xtables Architecture – Filter

- Filter Table

  The default table of iptables command

  For packets filter

  - INPUT
    - Packets that come in (to local)
  - OUTPUT
    - Packets that go out (from local)
  - FORWARD
    - Packets that pass through (from others to others)

Xtables Architecture – NAT

- NAT tables

  For IP masquerade

  - PREROUTING
    - Packets that will go into the routing tables
  - POSTROUTING
    - Packets that have left the routing tables
  - OUTPUT
    - Packets that go out (from local)

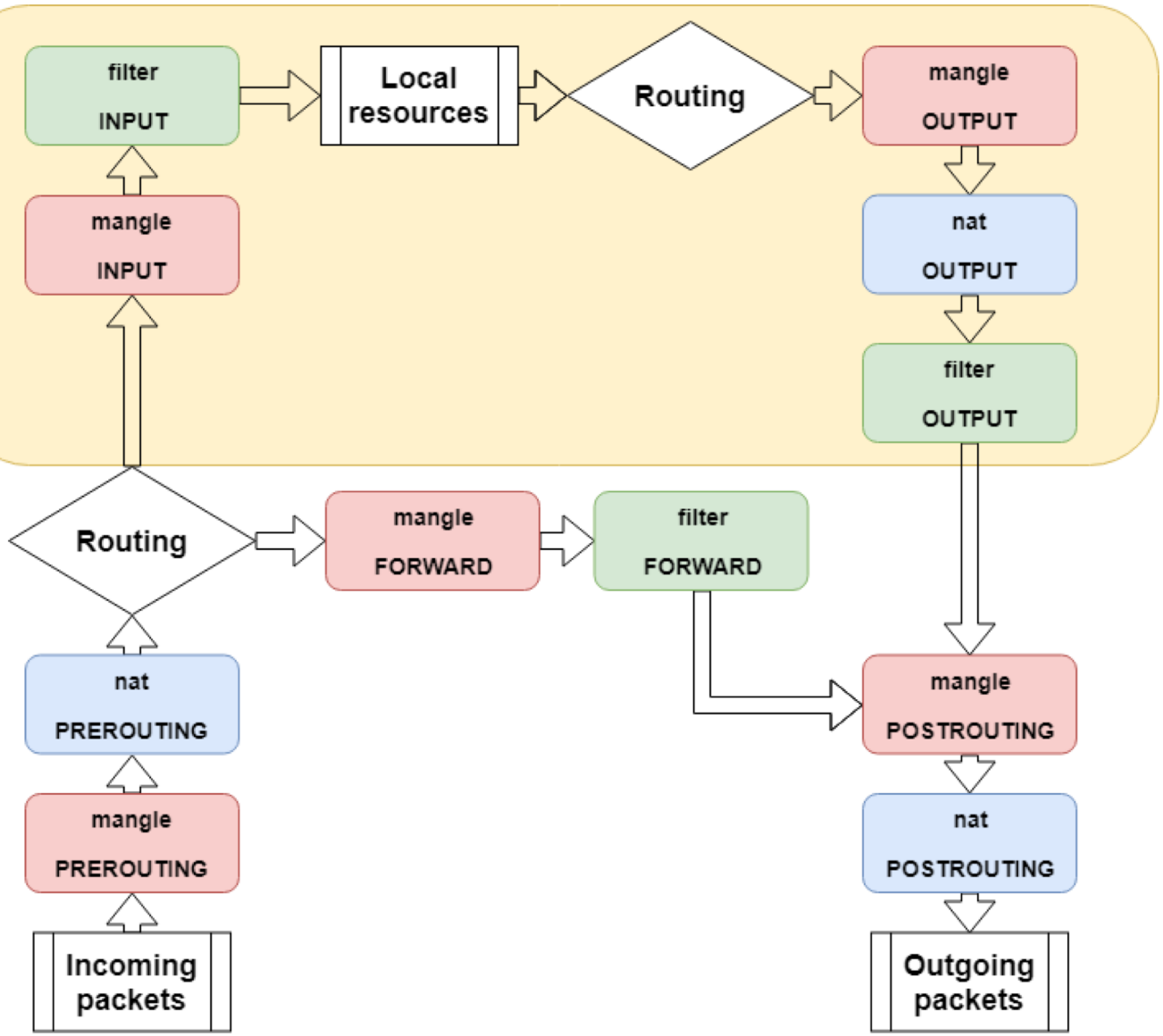
Xtables Architecture – Mangle

- Mangle Table

  For special purpose, e.g., add or remove some special tags from packets

  - PREROUTING
  - OUTPUT
  - FORWARD
  - INPUT
  - POSTROUTING

iptables Flowchart



iptables – List

- iptables
  - -t tables : Target table
  - -L : List all rules
  - -n : Don't lookup domain names
  - -v : Show details

```
zswu@ linux1 (/bin/bash): ~
^_^(04:57 PM)$ sudo iptables -L -n
Chain INPUT (policy ACCEPT)
target     prot opt source               destination
ACCEPT     all  --  0.0.0.0/0            0.0.0.0/0
ACCEPT     all  --  [redacted]/16        0.0.0.0/0
ACCEPT     all  --  0.0.0.0/0            0.0.0.0/0
WORKSTATON-INPUT  tcp  --  0.0.0.0/0            0.0.0.0/0
WORKSTATON-INPUT  icmp --  0.0.0.0/0            0.0.0.0/0

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination

Chain BLOCK (1 references)
target     prot opt source               destination
DROP       all  --  0.0.0.0/0            0.0.0.0/0

Chain WORKSTATON-INPUT (2 references)
target     prot opt source               destination
DROP       all  --  0.0.0.0/0            0.0.0.0/0
cs-firewall  all  --  0.0.0.0/0            0.0.0.0/0
           tcp  --  0.0.0.0/0            0.0.0.0/0
BLOCK      tcp  --  0.0.0.0/0            0.0.0.0/0
ACCEPT     all  --  0.0.0.0/0            0.0.0.0/0

Chain cs-firewall (1 references)
target     prot opt source               destination
DROP       all  --  [redacted]           0.0.0.0/0
DROP       all  --  [redacted]           0.0.0.0/0
DROP       all  --  [redacted]           0.0.0.0/0
DROP       all  --  [redacted]           0.0.0.0/0
DROP       all  --  [redacted]           0.0.0.0/0
DROP       all  --  [redacted]           0.0.0.0/0
```

iptables – Init

- iptables
  - -F : Flush all rules
  - -X : Flush all custom chains
  - -Z : Flush all statistics data for all chains
- iptables
  - -P [INPUT,OUTPUT,FORWARD] [ACCEPT, DROP]
    - Change the default policy of the target chain

iptables – Save and Restore

- iptables-restore
  - Restore from restore file
- iptables-save
  - Export all rules and generate restore file
  - Some system will load restore file at boot
    - Ex: CentOS /etc/sysconfig/iptables /etc/sysconfig/ip6tables
- Restore file syntax
  - # comments
  - * table name
  - : chain default-policy [pkt:byte]
  - Rules
  - COMMIT (End of file)

```
1  *filter
2  :INPUT DROP [0:0]
3  :FORWARD DROP [0:0]
4  :OUTPUT ACCEPT [0:0]
5  -A INPUT -p icmp -j ACCEPT
6  -A INPUT -m conntrack --ctstate RELATED,ESTABLISHED -j ACCEPT
7  -A INPUT -i lo -j ACCEPT
8  -A INPUT -p tcp -j REJECT --reject-with tcp-reset
9  -A INPUT -p udp -j REJECT --reject-with icmp-port-unreachable
10 -A INPUT -j REJECT --reject-with icmp-proto-unreachable
11 COMMIT
```

iptables – Module

- User may need special rule to filter packets
- Split several feature into different module
- Stateful
  - Packets states tracking
  - Traffic statistics
- Use -m to access module
  - iptables -A INPUT -m conntrack …
  - iptables -A INPUT -m recent …

- http://ipset.netfilter.org/iptables-extensions.man.html

iptables – Rules (1/2)

- Modify
  - -A, --append
  - -C, --check
  - -D, --delete
  - -I, --insert
  - -R, --replace
- Jump
  - -j, --jump
    - To user-defined chain
    - ACCEPT, DROP, REJECT, RETURN, SNAT, DNAT, MASQUERADE
  - -g, --goto
    - Unlike the --jump option return will not continue processing in this chain but instead in the chain that called us via --jump.

iptables – Rules (2/2)

- Filter
  - -i, -o [if] : incoming interface / outgoing interface
    - -i ens192 -o docker0
  - -s, -d [net] : Source / Destination
    - -s 192.168.0.1/24 –d 140.113.1.1
  - --sport, --dport [port] : Source port / Destination port
    - --sport 22 --dport 80
  - -p [protocol] : tcp, udp, icmp, all
    - -p icmp
  - ! (not) : Invert matching
    - ! -s 140.113.1.0/24
    - ! -i eth0
    - ! -p udp

iptables – Custom chain

- Create
  - -N my-chain
  - Define in restore file
- When iptables reaches the end of user-defined chain, flow returns to the next rule in the calling chain
- Ex
  - -A INPUT -j badguy
  - -A INPUT -j ACCEPT
  - -A badguy -s 1.2.3.4 -j DROP
  - -A badguy -s 140.112.0.0/24 -j DROP
  - …

Example: Hello world

- Allow outgoing packets but deny all incoming packets, except the packets that reply users requests
  - -A INPUT -i lo -j ACCEPT
  - -A INPUT -m conntrack --ctstate RELATED,ESTABLISHED -j ACCEPT
- State
  - NEW : New connection
  - ESTABLISHED : Old connection
  - RELATED : New connection create by ESTABLISHED session
  - INVALID

Example: NAT

- Provides NAT from eth0 to eth1
  - sysctl -w net.ipv4.ip_forward=1
  - -t NAT -A POSTROUTING -i eth0 -o eth1 -j MASQUERADE
- Nat
  - SNAT --to-source : Change Source IP Address
  - DNAT --to-destination : Change Destination IP Address
  - MASQUERADE : Change Source IP Address (based on outgoing device IP Address)

Example: Prevent DDoS Attack

- Append traffic limit (10 times / 60 sec) to SSH services
  - -A INPUT -p tcp --dport 22 -m state --state NEW -m recent --set --name RECENT --rsource
  - -A INPUT -p tcp --dport 22 -m state --state NEW -m recent --rcheck --seconds 60 --hitcount 10 --name RECENT --rsource -j DROP
- xt_recent
  - Record every connection
  - Filter connection by connecting history

Other tools

- These tools help user to manage iptables rules
  - UFW (Uncomplicated Firewall) (Ubuntu)
    - Easy to use
    - Hard to customize
  - Firewalld (Redhat)
    - Another way to manage your firewall
- Sometime even with these tools, you still need to understand iptables, otherwise you cannot manage complicated firewall rules like docker network, kubernetes

PF in FreeBSD

Packet Filter (PF)

- Functionality
  - Filtering packets
  - NAT
  - Load balance
  - QoS: (ALTQ: Alternate Queuing)
  - Failover (pfsync + carp)

PF in FreeBSD – Enable pf*

- In /etc/rc.conf (kernel modules loaded automatically)

  pf_enable="YES"

  pflog_enable="YES"

  pfsync_enable="YES"

- Kernel configurations

  device pf

  device pflog

  device pfsync

```
# The pf packet filter consists of three devices:
#  The `pf' device provides /dev/pf and the firewall code itself.
#  The `pflog' device provides the pflog0 interface which logs packets.
#  The `pfsync' device provides the pfsync0 interface used for
#   synchronization of firewall state tables (over the net).
device          pf
device          pflog
device          pfsync
```

PF in FreeBSD – Commands

- /etc/rc.d/pf
  - start / stop / restart / status / check / reload / resync
- pfctl
  - -e / -d
  - -F {nat | rules | state | info | Tables | all | …}
  - -v -s {nat | rules | state | info | all | Anchors | Tables | …}
  - -v -n -f /etc/pf.conf
  - -t <table> -T {add | delete| test} {ip …}
  - -t <table> -T {show | kill | flush | …}
  - -k {host | network} [-k {host | network}]
  - -a {anchor} …
    - Default anchor: -a '*'
    - Ex. -a ‘ftp-proxy/*’

PF in FreeBSD – Config ordering

- Macros
  - user-defined variables, so they can be referenced and changed easily.
- Tables "table"
  - similar to macros, but efficient and more flexible for many addresses.
- Options "set"
  - tune the behavior of pf, default values are given.
- Normalization "scrub"
  - reassemble fragments and resolve or reduce traffic ambiguities.
- Queueing "altq", "queue"
  - rule-based bandwidth control.
- Translation (NAT) "rdr", "nat", "binat"
  - specify how addresses are to be mapped or redirected to other addresses
  - First match rules
- Filtering "antispoof", "block", "pass"
  - rule-based blocking or passing packets
  - Last match rules

PF in FreeBSD – Lists

❑ Lists

- Allow the specification of multiple similar criteria within a rule
  - ➢ multiple protocols, port numbers, addresses, etc.
- defined by specifying items within { } brackets.
- eg.
  - ➢ pass out on rl0 proto { tcp, udp } from { 192.168.0.1, 10.5.32.6 } to any
  - ➢ pass in on fxp0 proto tcp to port { 22 80 }
- Pitfall
  - ➢ pass in on fxp0 from { 10.0.0.0/8, !10.1.2.3 }
  - ➢ You mean (It means)
    1. pass in on fxp0 from 10.0.0.0/8
    2. block in on fxp0 from 10.1.2.3
    2. pass in on fxp0 from !10.1.2.3
  - ➢ Use table, instead.

PF in FreeBSD – Macros

❑ Macros

- user-defined variables that can hold IP addresses, port numbers, interface names, etc.
- reduce the complexity of a pf ruleset and also make maintaining a ruleset much easier.
- Naming: start with [a-zA-Z] and may contain [a-zA-Z0-9_]
- eg.
  - ext_if = "fxp0“
  - block in on $ext_if from any to any
- Macro of macros
  - host1 = "192.168.1.1“
  - host2 = "192.168.1.2“
  - all_hosts = "{" $host1 $host2 "}"

PF in FreeBSD – Tables (1)

- Tables
  - used to hold a group of IPv4 and/or IPv6 addresses
    - hostname, inteface name, and keyword *self*
  - Lookups against a table are very fast and consume less memory and processor time than lists
  - Two attributes
    - persist: keep the table in memory even when no rules refer to it
    - const: cannot be changed once the table is created
  - eg.
    - table <private> const { 10/8, 172.16/12, 192.168/16 }
    - table <badhosts> persist
    - block on fxp0 from { <private>, <badhosts> } to any
    - table <spam> persist file "/etc/spammers" file "/etc/openrelays"

PF in FreeBSD – Tables (2)

❑ Tables – Address Matching

- An address lookup against a table will return the most narrowly matching entry
- eg.
  - table <goodguys> { 172.16.0.0/16, !172.16.1.0/24, 172.16.1.100 }
  - block in on dc0
  - pass in on dc0 from <goodguys>
- Result
  - 172.16.50.5 passed
  - 172.16.1.25 blocked
  - 172.16.1.100 passed
  - 10.1.4.55 blocked

PF in FreeBSD – Options

- Format
  - control pf's operation, and specified in pf.conf using "set"
    - Format: set option [sub-ops] value
- Options
  - *loginterface* – collect packets and gather byte count statistics
  - *ruleset-optimization* – ruleset optimizer
    - none, basic, profile
    - basic: remove dups, remove subs, combine into a table, re-order rules
  - *block-policy* – default behavior for blocked packets
    - drop, return
  - *skip on* {ifname} – interfaces for which packets should not be filtered.
    - eg. set skip on lo0
  - *timeout, limit, optimization, state-policy, hostid, require-order, fingerprints, debug*

PF in FreeBSD – Normalization

- Traffic Normalization
  - IP fragment reassembly
    - scrub in all
  - Default behavior
    - Fragments are buffered until they form a complete packet, and only the completed packet is passed on to the filter.
    - Advantage: filter rules have to deal only with complete packets, and ignore fragments.
    - Disadvantage: caching fragments is the additional memory cost
    - The full reassembly method is the only method that currently works with NAT.

PF in FreeBSD – Translation (1)

- Translation
  - Modify either the source or destination address of the packets
  - The translation engine
    1. modifies the specified address and/or port in the packet
    2. passes it to the packet filter for evaluation
  - Filter rules filter based on the translated address and port number
  - Packets passed directly if the *pass* modifier is given in the rule

PF in FreeBSD – Translation (2)

- Various types of translation
  - binat – bidirectional mapping between an external IP netblock and an internal IP netblock
    - binat on $ext_if from 10.1.2.150 to any -> 140.113.235.123
    - binat on $ext_if from 192.168.1.0/28 to any -> 140.113.24.0/28
  - nat – IP addresses are to be changes as the packet traverses the given interface
    - no nat on $ext_if from 192.168.123.234 to any
    - nat pass on $ext_if from 192.168.123.0/24 to any -> 140.113.235.21
  - rdr – redirect packets to another destination and possibly different port
    - no rdr on $int_if proto tcp from any to $server port 80
    - rdr on $int_if proto tcp from any to any port 80 -> 127.0.0.1 port 80

PF in FreeBSD – Translation (3)

- Evaluation
  - Evaluation order of translation rules depends on the type
    - *binat* rules first, and then either *rdr* rules for inbound packets or *nat* rules for outbound packets
  - Rules of the same type are evaluated in the order of appearing in the ruleset
  - The first matching rule decides what action is taken
  - If no rule matches the packet, it is passed to the filter unmodified

PF in FreeBSD – Packet Filtering (1)

- pf has the ability to *block* and *pass* packets based on
  - layer 3(ip, ip6) and layer 4(icmp, icmp6, tcp, udp) headers
- Each packet processed by the filter
  - The filter rules are evaluated in sequential order
  - The last matching rule decides what action is taken
  - If no rule matches the packet, the default action is to pass
- Format
  - {pass | block [drop | return]} [in | out] [log] [quick] [on ifname] … {hosts} …
  - The simplest to block everything by default: specify the first filter rule
    - block all

PF in FreeBSD – Packet Filtering (2)

- States
  - If the packet is *pass*ed, state is created unless the *no state* is specified
    - The first time a packet matches *pass*, a state entry is created
    - For subsequent packets, the filter checks whether each matches any state
    - For TCP, also check its sequence numbers
    - pf knows how to match ICMP replies to states
      - Port unreachable for UDP
      - ICMP echo reply for echo request
      - …
    - Stores in BST for efficiency

PF in FreeBSD – Packet Filtering (3)

- Parameters
  - *in* | *out* – apply to imcoming or outgoing packets
  - *log* - generate log messages to pflog (pflog0, /var/log/pflog)
    - Default: the packet that establishes the state is logged
  - *quick* – the rule is considered the last matching rule
  - *on ifname* – apply only on the particular interface
  - *inet* | *inet6* – apply only on this address family
  - *proto* {*tcp* | *udp* | *icmp* | *icmp6*} – apply only on this protocol

PF in FreeBSD – Packet Filtering (4)

- Parameters
  - *hosts* : { *from* host [*port* [*op*] #] *to* host [*port* [*op*] #] | *all* }
  - host:
    - host can be specified in CIDR notation, hostnames, interface names, table, or keywords *any*, *self*, …
    - Hostnames are translated to address(es) at ruleset load time.
    - When the address of an interface or hostname changes, the ruleset must be reloaded
    - When interface name is surrounded by (), the rule is automatically updated whenever the interface changes its address
  - port:
    - ops: unary(=, !=, <, <=, >, >=), and binary(:, ><, <>)
  - eg.
    - block in all
    - pass in proto tcp from any port < 1024 to self port 33333:44444

PF in FreeBSD – Packet Filtering (5)

- Parameters
  - *flags* {<a>/<b> | *any*} – only apply to TCP packets
    - Flags: (F)IN, (S)YN, (R)ST, (P)USH, (A)CK, (U)RG, (E)CE, C(W)R
    - Check flags listed in <b>, and see if the flags (not) in <a> is (not) set
    - eg.
      - flags S/S : check SYN is set, ignore others.
      - flags S/SA: check SYN is set and ACK is unset., ignore others
    - Default *flags S/SA* for TCP
  - *icmp-type type code code*
  - *icmp6-type type code code*
    - Apply to ICMP and ICMP6 packets
  - *label* – for per-rule statistics
  - {*tag* | *tagged*} *string*
    - tag by nat, rdr, or binat, and identify by filter rules.

PF in FreeBSD – Load Balance

- Load balance
  - For *nat* and *rdr* rules
  - eg.
    - rdr on $ext_if proto tcp from any to any port 80 \
      -> {10.1.2.155, 10.1.2.160, 10.1.2.161} round-robin

PF in FreeBSD – Security

- For security consideration
  - state modulation
    - Create a high quality random sequence number
    - Applying *modulate state* parameter to a TCP connection
  - syn proxy
    - pf itself completes the handshake
    - Applying *synproxy state* parameter to a TCP connection
      - Include modulate state

PF in FreeBSD – Stateful tracking

- Stateful tracking options
  - *keep state*, *modulate state*, and *synproxy state* support these options
    - keep state must be specidied explicitly to apply options to a rule
  - eg.
    - table <bad_hosts> persist
    - block quick from <bad_hosts>
    - pass in on $ext_if proto tcp to ($ext_if) port ssh keep state \
      (max-src-conn-rate 5/30, overload <bad_hosts> flush global)

PF in FreeBSD – Blocking spoofed

- Blocking spoofed traffic
  - *antispoof for ifname*
  - antispoof for lo0
    - block drop in on ! lo0 inet from 127.0.0.1/8 to any
    - block drop in on ! lo0 inet6 from ::1 to any
  - antispoof for wi0 inet (IP: 10.0.0.1, netmask 255.255.255.0)
    - block drop in on ! wi0 inet from 10.0.0.0/24 to any
    - block drop in inet from 10.0.0.1 to any
  - Pitfall:
    - Rules created by the *antispoof* interfere with packets sent over loopback interfaces to local addresses. One should pass these explicitly.
    - set skip on lo0

PF in FreeBSD – Anchors

- Besides the main ruleset, pf can load rulesets into anchor attachment points
  - An anchor is a container that can hold rules, address tables, and other anchors
  - The main ruleset is actually the default anchor
  - An anchor can reference another anchor attachment point using
    - nat-anchor
    - rdr-anchor
    - binat-anchor
    - anchor
    - load anchor <name> from <file>

PF in FreeBSD – Example

- Ex.

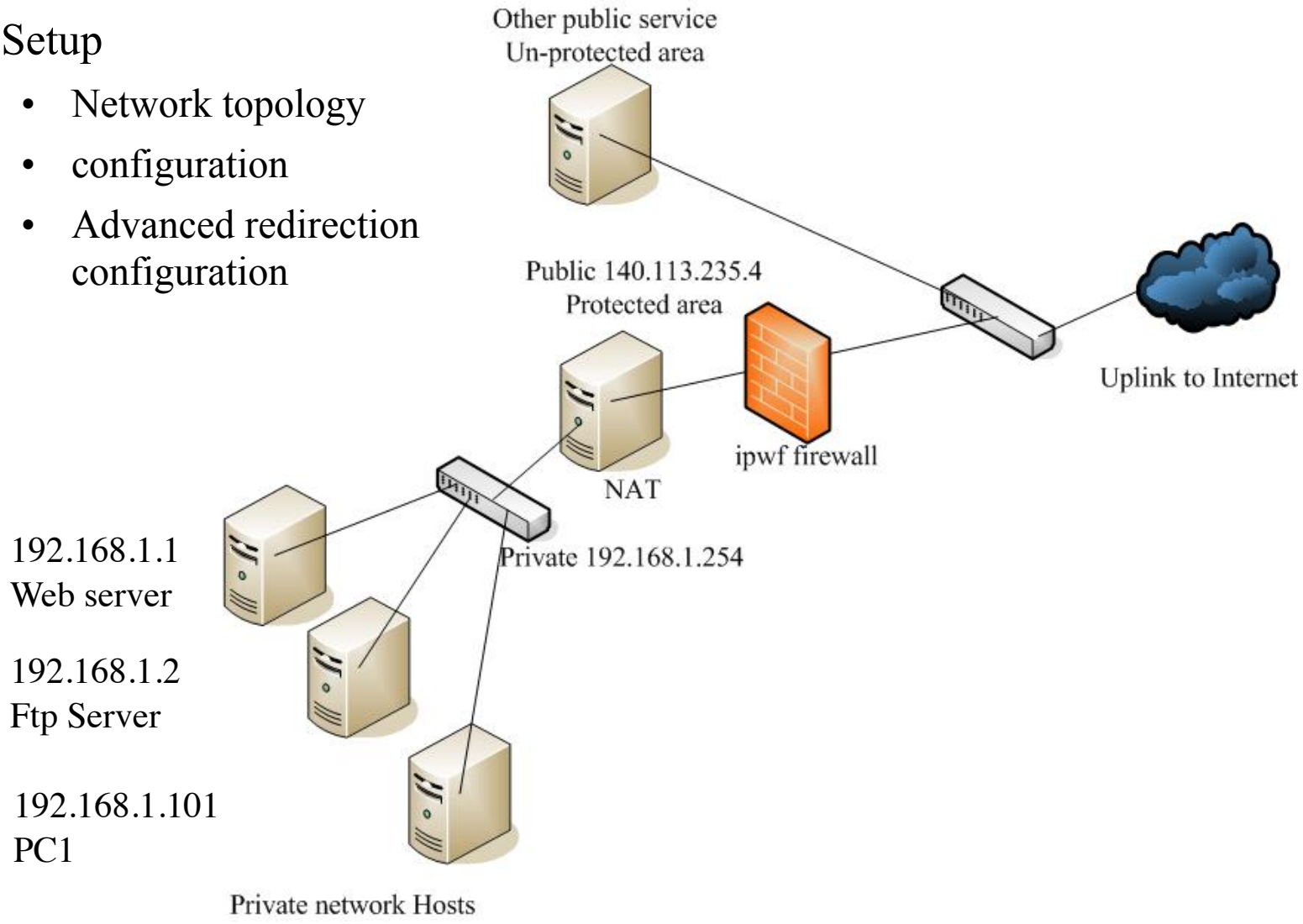
```
# macro definitions
extdev='fxp0‘
server_ext=‘140.113.214.13’

# options
set limit { states 10000, frags 5000 }
set loginterface $extdev
set block-policy drop
set skip on lo0

# tables
table <badhosts> persist file “/etc/badhosts.list”

# filtering rules
block in  all
pass out all
antispoof for $extdev
block log in on $extdev proto tcp from any to any port {139, 445}
block log in on $extdev proto udp from any to any port {137, 138}
block on $extdev quick from <badhosts> to any
pass in on $extdev proto tcp from 140.113.0.0/16 to any port {139, 445}
pass in on $extdev proto udp from 140.113.0.0/16 to any port {137, 138}
```

PF in FreeBSD – Debug by pflog

- Enable pflog in /etc/rc.conf (pflog.ko loaded automatically)
  - pflog_enable="YES"
    - Log to pflog0 interface
    - tcpdump -i pflog0
  - pflog_logfile="/var/log/pflog"
    - tcpdump -r /var/log/pflog
- Create firewall rules
  - Default configuration rules
    - pf_rules="/etc/pf.conf"
  - Sample files
    - /usr/share/examples/pf/*

NAT on FreeBSD (1)

- Setup
  - Network topology
  - configuration
  - Advanced redirection configuration



NAT on FreeBSD (2)

- In /etc/rc.conf

```
ifconfig_fxp0="inet 140.113.235.4"
ifconfig_fxp1="inet 192.168.1.254/24"
defaultrouter="140.113.235.254"
gateway_enable="YES"
```

- In /etc/pf.conf
  - nat
  - rdr
  - binat

```
# macro definitions
extdev='fxp0‘
intranet='192.168.1.0/24‘
webserver=‘192.168.1.1’
ftpserver=‘192.168.1.2’
winxp=‘192.168.1.101’
server_int=‘192.168.1.88’
server_ext=‘140.113.235.13’

# nat rules
nat on $extdev inet from $intranet to any -> $extdev
rdr on $extdev inet proto tcp to port 80 -> $webserver port 80
rdr on $extdev inet proto tcp to port 443 -> $webserver port 443
rdr on $extdev inet proto tcp to port 21 -> $ftpserver port 21
rdr on $extdev inet proto tcp to port 3389 -> $winxp port 3389
binat on $extdev inet from $server_int to any -> $server_ext
```

ALTQ: Alternate Queue – (1)

- Rebuild Kernel is needed
  - http://www.freebsd.org/doc/handbook/firewalls-pf.html
  - ALTQ related kernel options and supported devices
    - man 4 altq

```
# altq(9). Enable the base part of the hooks with the ALTQ option.
# Individual disciplines must be built into the base system and can not be
# loaded as modules at this point. ALTQ requires a stable TSC so if yours is
# broken or changes with CPU throttling then you must also have the ALTQ_NOPCC
# option.
options         ALTQ
options         ALTQ_CBQ        # Class Based Queueing
options         ALTQ_RED        # Random Early Detection
options         ALTQ_RIO        # RED In/Out
options         ALTQ_HFSC       # Hierarchical Packet Scheduler
options         ALTQ_CDNR       # Traffic conditioner
options         ALTQ_PRIQ       # Priority Queueing
options         ALTQ_NOPCC      # Required if the TSC is unusable
options         ALTQ_DEBUG
```

ALTQ: Alternate Queue – (2)

- altq on dc0 cbq bandwidth 5Mb queue {std, http}
- queue std bandwidth 10% cbq(default)
- queue http bandwidth 60% priority 2 cbq(borrow) {employee,developer}
- queue developers bandwidth 75% cbq(borrow)
- queue employees bandwidth 15%

- block return out on dc0 inet all queue std
- pass out on dc0 inet proto tcp from $developerhosts to any port 80 queue developers
- pass out on dc0 inet proto tcp from $employeehosts to any port 80 queue employees
- pass out on dc0 inet proto tcp from any to any port 22
- pass out on dc0 inet proto tcp from any to any port 25