Firewalls

Firewalls

- Firewall
  - A piece of hardware and/or software which functions in a networked environment to prevent some communications forbidden by the security policy.
  - Choke point between secured and unsecured network
  - Filter incoming and outgoing traffic that flows through your system
- What it can be used to do
  - To protect and insulate the applications, services and machines of your internal network from unwanted traffic coming in from the public Internet
    - Such as telnet, NetBIOS
  - To limit or disable access from hosts of the internal network to services of the public Internet
    - Such as MSN, ssh, ftp
  - To support NAT (Network Address Translation)

Firewalls – Layers of Firewalls

- 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.

Firewall Rules

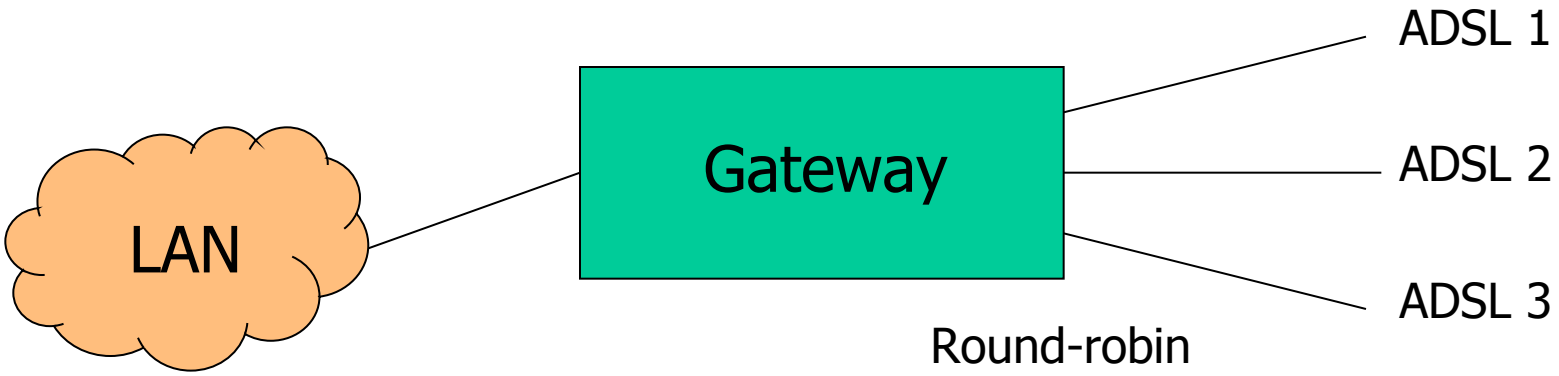
- Two ways to create firewall rulesets
  - Exclusive
    - Allow all traffic through except for the traffic matching the rulesets
  - Inclusive
    - Allow traffic matching the rulesets and blocks everything else
    - Offer much better control of the outgoing traffic
    - Control the type of traffic originating from the public Internet that can gain access to your private network
    - Safer than exclusive one
      - reduce the risk of allowing unwanted traffic to pass
      - Increase the risk to block yourself with wrong configuration
- Stateful firewall
  - Keep track of which connections are opened through the firewall
  - Be vulnerable to Denial of Service (DoS) attacks

Firewall Packages

- FreeBSD
  - IPFILTER (known as IPF)
  - IPFIREWALL (known as IPFW) + Dummynet
  - *Packet Filter (known as PF)+ ALTQ*
- Solaris
  - IPF
- Linux
  - ipchains
  - iptables

Packet Filter (PF)

- Introduction
  - Packet filtering
  - Translation (NAT)
  - Alternate Queuing (ALTQ) for QoS , bandwidth limit
  - Load balance
  - Failover (pfsync + carp)
  - Firewall migrated from OpenBSD
    - http://www.openbsd.org/faq/pf/



PF in FreeBSD (1) – enabling pf

- Enable pf in /etc/rc.conf (pf.ko loaded automatically)

```
pf_enable="YES"
```

- Rebuild Kernel (if pfsync, ALTQ is needed)

```
device     pf      # Enable "Packet Filter" firewall
device     pflog   # pseudo device to log traffic
# device   pfsync  # pseudo device to monitor "state changes"
options    ALTQ
options    ALTQ_CBQ      # Class based queueing
options    ALTQ_PRIQ     # Priority queueing
options    ALTQ_{RED | RIO} # Avoid network congestion
options    ALTQ_HFSC     # Hierarchical Fair Service Curve
```

Ref: http://www-2.cs.cmu.edu/~hzhang/HFSC/main.html

PF in FreeBSD (2) – enabling 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/*

PF in FreeBSD (3) – related commands

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

PF in FreeBSD (4) – 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 (5) – 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 (6) – 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 (7) – Tables

- 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 (8) – Tables

- 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 (9) – 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 (10) – 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 (11) – Queueing

- ALTQ

```
altq on interface type [options ... ] main_queue { sub_q1, sub_q2 ..}
        queue sub_q1 [ options ... ]
        queue sub_q2 [ options ... ] { subA, subB, ... }
        [...]

pass [ ... ] queue sub_q1
pass [ ... ] queue sub_q2
```

PF in FreeBSD (11) – Queueing

- Queue scheduler (a.k.a. Queue disciplines)
  - Default: FIFO (without ALTQ)
  - priq - Priority-based Queueing
  - cbq - Class-based Queueing
  - hfsc - Hierarchical Fair Service Curve

PF in FreeBSD (11) – Queueing

- priq - Priority-based queue
  - defined purely in terms of priority within total declared bandwidth
  - priority range:
    - 0 ~ 15 (packets with high priority is served first)

PF in FreeBSD (11) – Queueing

- Example with priq:

```
$ext_bw = “64Kb”

altq on $ext_if priq bandwidth $ext_bw queue { q_pri, q_def }
        queue q_pri priority 7
        queue q_def priority 1

pass out on $ext_if queue (q_def, q_pri)
```

PF in FreeBSD (11) – Queueing

- cbq - Class-based queue
  - defined as constant-sized bandwidth allocation
    - ➢ percentage of total available bandwidth
    - ➢ bandwidth in units of kilobits, megabits or gigabits
  - can be subdivded into queues that are assigned priority
    - ➢ priority range:
      - 0 ~ 7 (packets with high priority is served first)

PF in FreeBSD (11) – Queueing

- Example with cbq:

```
altq on $ext_if cbq bandwidth 2Mb queue { main, ftp, udp, web, ssh, icmp }
        queue main bandwidth 18% cbq(default borrow red)
        queue ftp bandwidth 10% cbq(borrow red)
        queue udp bandwidth 30% cbq(borrow red)
        queue web bandwidth 20% cbq(borrow red)
        queue ssh bandwidth 20% cbq(borrow red) { ssh_interactive, ssh_bulk }
                queue ssh_interactive priority 7 bandwidth 20%
                queue ssh_bulk priority 5 bandwidth 80%
        queue icmp bandwidth 2% cbq

pass log quick on $ext_if proto tcp to port ssh queue (ssh_bulk, ssh_interactive)
pass in quick on $ext_if proto tcp to port ftp queue ftp
pass in quick on $ext_if proto tcp to port www queue http
pass out on $ext_if proto udp queue udp
pass out on $ext_if proto icmp queue icmp
pass out on $ext_if proto tcp from $localnet to port $client_out
```

PF in FreeBSD (11) – Queueing

- hfsc - Hierarchical Packet Scheduler
  - uses HFSC algorithm to ensure fair allocation of the bandwidth among queues in the hierarchy
  - can setup guaranteed minimum allocations and hard upper limits
  - allocations can vary over time
  - can have priority
    - 0 - 7

PF in FreeBSD (11) – Queueing

- Example with hfsc:

```
altq on $ext_if bandwidth $ext_bw hfsc queue { main, spamd }
        queue main bandwidth 99% priority 7 qlimit 100 hfsc (realtime 20%, linkshare 99%) \
                                                { q_pri, q_def, q_web, q_dns }
                queue q_pri bandwidth 3% priority 7 hfsc (realtime 0, linkshare 3% red )
                queue q_def bandwidth 47% priority 1 hfsc (default realtime 30% linkshare 47% red)
                queue q_web bandwidth 47% priority 1 hfsc (realtime 30% linkshare 47% red)
                queue q_dns bandwidth 3% priority 7 qlimit 100 hfsc (realtime (30Kb 3000 12Kb), \
                                                                linkshare 3%)
        queue spamd bandwidth 0% priority 0 qlimit 300 hfsc (realtime 0, upperlimit 1%, \
                                                        linkshare 1%)
```

PF in FreeBSD (12) – Translation

- Translation
  - Modify either the source or destination address of the packets
  - The translation engine modifies the specified address and/or port in the packet, and then 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 (13) – Translation

- 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 (14) – Translation

- 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 (15) – Packet filtering

- 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 (16) – Packet filtering

- 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 (17) – Packet filtering

- 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 (18) – Packet filtering

- 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 (19) – Packet filtering

- 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 (20) - 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 (22) – Security

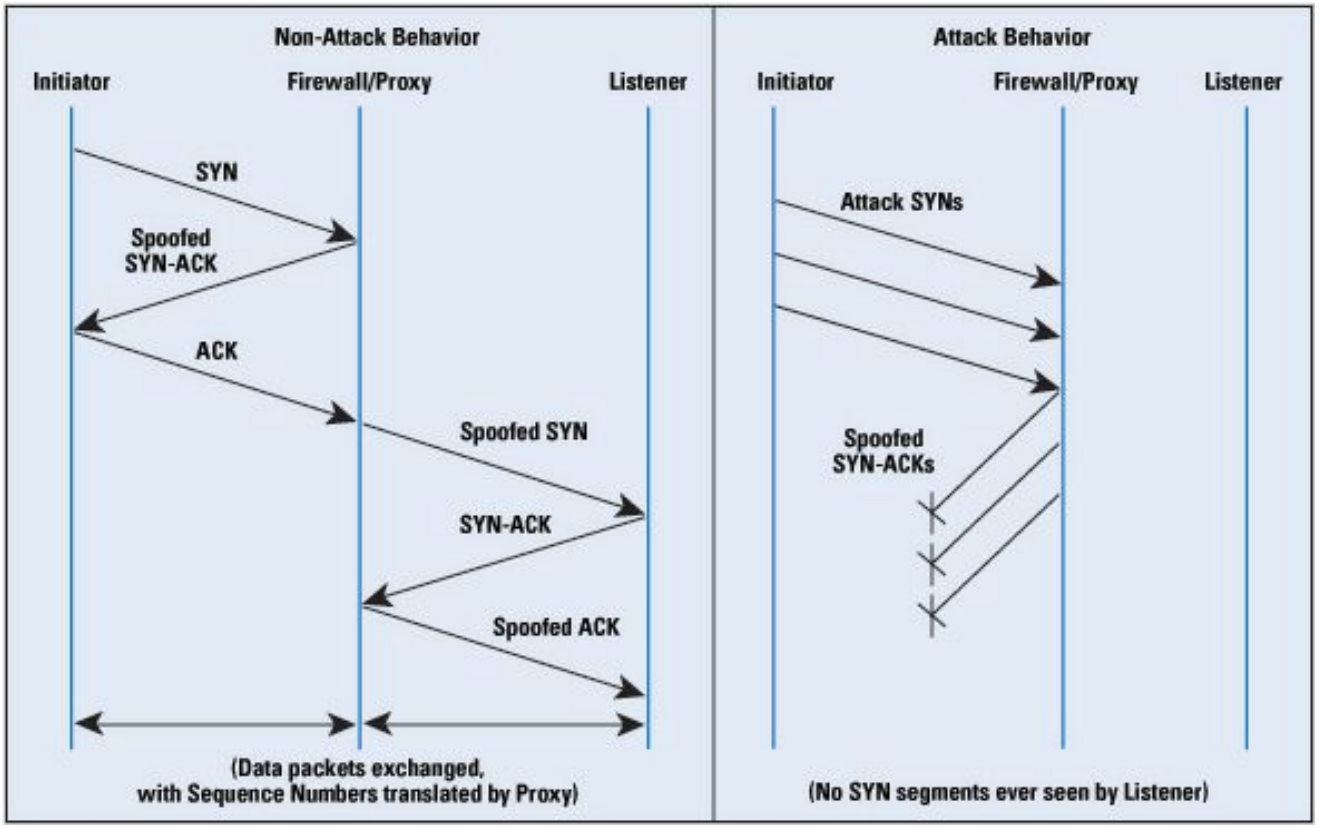
- For security consideration
  - state modulation
    - ➢ Applying *modulate state* parameter to a TCP connection
  - syn proxy
    - ➢ Applying *synproxy state* parameter to a TCP connection
      - – Include modulate state

PF in FreeBSD (22) – 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 (22) – Stateful tracking

- Synproxy state



PF in FreeBSD (23) – 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 (24) – 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>

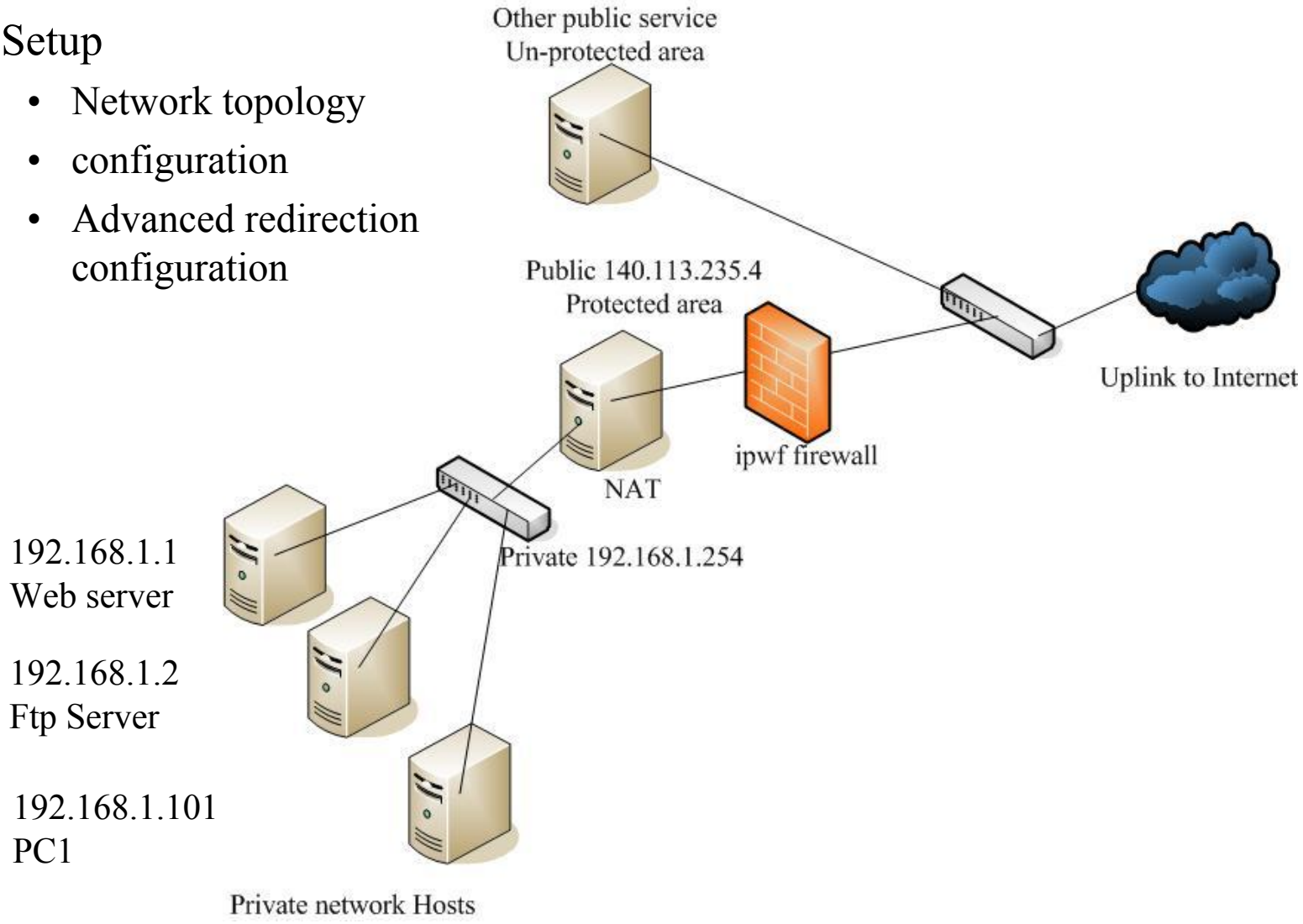
PF in FreeBSD (25)

- 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}
```

NAT on FreeBSD (1)

- Setup
  - Network topology
  - configuration
  - Advanced redirection configuration



NAT on FreeBSD (2)

- IP configuration (in /etc/rc.conf)

  ifconfig_fxp0="inet 140.113.235.4 netmask 255.255.255.0 media autoselect"

  ifconfig_fxp1="inet 192.168.1.254 netmask 255.255.255.0 media autoselect“

  defaultrouter="140.113.235.254“

- Enable NAT
  - Here we use Packet Filter (PF) as our NAT server
  - Configuration file: /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’
pc1=‘192.168.1.101’

# 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
```

NAT on FreeBSD (3)

```
# macro definitions
extdev='fxp0‘
intranet='192.168.219.0/24‘
winxp=‘192.168.219.1’
server_int=‘192.168.219.2’
server_ext=‘140.113.214.13’

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

Redundancy and Failover - CARP and pfsync

- Common Address Redundancy Protocol
  - non-patent-encumbered alternative to the Virtual Router Redundancy Protocol (VRRP)
  - ensure firewall/services will keep functioning under
    - errors
    - planned maintenence
  - authenticated redundancy

Redundancy and Failover - CARP and pfsync

- Common Address Redundancy Protocol
  - parameters
    - vhid (virtual host id)
      - consistent vhid for each machine participating in the virtual group
    - advbase (sec): interval of the advertisement (default: 1sec)
    - advskew (1/256 sec): added to the base advertisement interval to make one host advertise a bit slower
  - demotion
    - indicate the readiness of a particular host
  - the advertisement interval will be:
    - advbase + (advskew+demotion)/256 (secs)

Setting up CARP

- In /boot/loader.conf:
  - carp_load=”YES”
- Load the module without rebooting
  - kldload carp
- In /etc/rc.conf

```
# Host A
ifconfig_em0=”inet 10.0.0.2 netmask 255.255.2jjjjkk55.0”
ifconfig_em0_alias0=”vhid 1 advskew 100 pass youneverknow 10.0.0.1/24”

# Host B
ifconfig_em0=”inet 10.0.0.3 netmask 255.255.255.0”
ifconfig_em0_alias0=”vhid 1 advskew 200 pass youneverknow 10.0.0.1/24”
```

Setting up CARP

- In /etc/pf.conf:
  - pass proto carp

Redundancy and Failover - CARP and pfsync

- Common Address Redundancy Protocol
  - global parameters set using sysctl
    - net.inet.carp.allow (default to 1)
    - net.inet.carp.preempt (default to 0)
    - net.inet.carp.log (default to 1)
    - net.inet.carp.log.demotion (default to 0)
    - net.inet.carp.ifdown_demotion_factor (default to 240)
    - net.inet.carp.senderr_demotion_factor (default to 240)
  - force preemption
    - ifconfig em0 vhid 1 state master

Redundancy and Failover - CARP and pfsync

- pfsync
  - synchronize the active connections state to redundant firewall
  - caveat! the protocol is unecrypted itself, you must either:
    - setup an encrypted tunnel (e.g. ipsec, openvpn…)
    - use a crosswire to connect the redundant pair directly

Setting up pfsync

- In /boot/loader.conf:
  - pfsync_load=”YES”
- In /etc/rc.conf:
  - ifconfig_em1=”inet 192.168.0.253/24”
  - pfsync_enable=”YES”
  - pfsync_syncdev=”em1”
- Enable using command line
  - kldload pfsync
  - ifconfig pfsync0 syncdev em1
- Sysctl tunable
  - net.pfsync.carp_demotion_factor