

Basic Concept of Firewall

jnlin (2020)

? (~ 2019)

交大資工系資訊中心

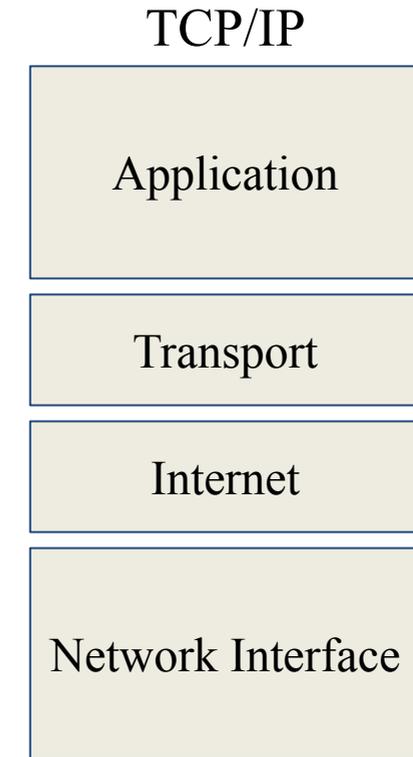
Computer Center of Department of Computer Science, NCTU

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
- The usage
 - **Incoming**: protect and insulate the applications, services and machines
 - Such as ssh, NFS, telnet, NetBIOS(samba), internal web servers
 - **Outgoing**: limit or disable access from the internal network
 - Such as Line, ssh, ftp, Facebook, Online Games
 - **NAT** (Network Address Translation)

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
 - hosts.allow, hosts.deny
 - In FreeBSD: tcpd(8)



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 – Packages

- Linux
 - iptables (kernel 2.4+)
 - ipchains (kernel < 2.4)
 - firewalld
 - ufw (ubuntu)
- FreeBSD
 - IPFILTER (known as IPF)
 - IPFWALL (known as IPFW) + Dummynet
 - **Packet Filter (known as PF)+ ALTQ**
 - Migrated from OpenBSD
 - <http://www.openbsd.org/faq/pf/>

Basic PF in FreeBSD

Packet Filter (PF)

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

Not covered today

PF in FreeBSD – Enable pf*

- In /etc/rc.conf
 - 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 and Config

- /etc/rc.d/pf
 - **start** / stop / **restart** / status / check / **reload** / resync
 - reboot if kernel modules is not loaded
- /etc/pf.conf
 - rules for PF
 - traffics to block/pass
 - tables to lookup
 - ...

PF in FreeBSD – Example

```
# 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 in log on $extdev proto tcp from any to any port {139, 445}
block in log on $extdev proto udp from any to any port {137, 138}
block quick on $extdev 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 – Tool

- pfctl
 - -e / -d
 - Enable/disable
 - -F {nat | rules | state | info | Tables | all | ...}
 - Flush rules
 - -v -s {nat | rules | state | info | all | Anchors | Tables | ...}
 - Show current rules
 - -v -n -f /etc/pf.conf
 - Parse the rule file without actually take effect
 - Suitable for testing marcos

PF in FreeBSD – Tool

- pfctl
 - `-t table_name -T {add | delete| test} {ip ...}`
 - Modify lookup table, add/remove IP addresses
 - `-t table_name -T {show | kill | flush | ...}`
 - Show/disable/reload tables
 - `-k {host | network} [-k {host | network}]`
 - Kill internal state entries for given host/network

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.
 - E.g.
 - 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 }

PF in FreeBSD – Lists

- Lists
 - Pitfall
 - A lists will be expanded into rules.
 - Last matched rule takes effect
 - 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_]
 - E.g.
 - `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 }"`
 - Macros are not expanded within quotes!

PF in FreeBSD – Tables (1)

- Tables
 - Used to hold a group of IPv4 and/or IPv6 addresses
 - Supports address lookup and query
 - Hostname, interface 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
 - E.g.
 - `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

- E.g.

- `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.
 - E.g. 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 – 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 *passed*, **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)

- Block policy
 - drop
 - Incoming packet is silently dropped.
 - return
 - Incoming packet is dropped
 - For TCP packets
 - TCP RST is returned
 - For UDP packets
 - ICMP UNREACHABLE is returned
 - For other packets
 - No response is sent

PF in FreeBSD – Packet Filtering (3)

- Parameters
 - *in* | *out* – apply to incoming 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(:, ><, <>)
- E.g.
 - 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>/ | 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 , and see if the flags (not) in <a> is (not) set
 - E.g.
 - 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 – Stateful tracking

- Stateful tracking options
 - *keep state*, *modulate state*, and *synproxy state* support these options
 - *keep state* must be specified explicitly to apply options to a rule
 - E.g.
 - `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 – Example

```
# 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 in log on $extdev proto tcp from any to any port {139, 445}
block in log on $extdev proto udp from any to any port {137, 138}
block quick on $extdev 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_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/*

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

iptables

- In SA, we only cover high level idea of iptables
- Detailed configuration and usage are covered in NA

iptables - filtering

- Main command: iptables
- Almost everything is done by it
- iptables content for new machine (ubuntu)
 - iptables -L

```
Chain INPUT (policy ACCEPT)
target      prot opt source                destination

Chain FORWARD (policy ACCEPT)
target      prot opt source                destination

Chain OUTPUT (policy ACCEPT)
target      prot opt source                destination
```

iptables – List

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

```
$ sudo iptables -L -n
Chain INPUT (policy ACCEPT)
target      prot opt source                destination

Chain FORWARD (policy ACCEPT)
target      prot opt source                destination
ACCEPT     all  --  0.0.0.0/0             0.0.0.0/0        ctstate RELATED,ESTABLISHED
DOCKER     all  --  0.0.0.0/0             0.0.0.0/0
ACCEPT     all  --  0.0.0.0/0             0.0.0.0/0
ACCEPT     all  --  0.0.0.0/0             0.0.0.0/0

Chain OUTPUT (policy ACCEPT)
target      prot opt source                destination
```

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
 - E.g.: CentOS /etc/sysconfig/iptables /etc/sysconfig/ip6tables
- Restore file syntax
 - # comments
 - * table name
 - : chain default-policy [pkt:byte]
 - Rules
 - COMMIT (End of file)

```
*filter
:INPUT ACCEPT [8:1468]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [855:500357]
:BLOCK - [0:0]
:WORKSTON-INPUT - [0:0]
:cs-firewall - [0:0]
-A INPUT -i lo -j ACCEPT
-A INPUT -s 10.1.0.0/16 -j ACCEPT
-A INPUT -m conntrack --ctstate RELATED,ESTABLISHED -j ACCEPT
COMMIT
```

iptables – Rules (1/2)

- Modify
 - -A, --append
 - -C, --check
 - -D, --delete
 - -I, --insert
 - -R, --replace

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`
 - `-j [target]`: target for matched packets
 - `-j ACCEPT, -j DROP`
 - `!` (not) : Invert matching
 - `! -s 140.113.1.0/24`
 - `! -i eth0`
 - `! -p udp`

Example

- Allow all packets from 192.168.1.0/24 on eth0
 - iptables -A INPUT -i eth0 -p tcp -s 192.168.1.0/24 -j ACCEPT
- Drop packets from 192.168.1.25
 - iptables -A INPUT -i eth0 -p tcp -s 192.168.1.25 -j DROP

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