MindWorks UK

IPtables and Blocking Network Intruders

Joe R. Doupnik

MindworksUK and Univ of Oxford

joe.doupnik@oucs.ox.ac.uk

Some slides are marked as hidden

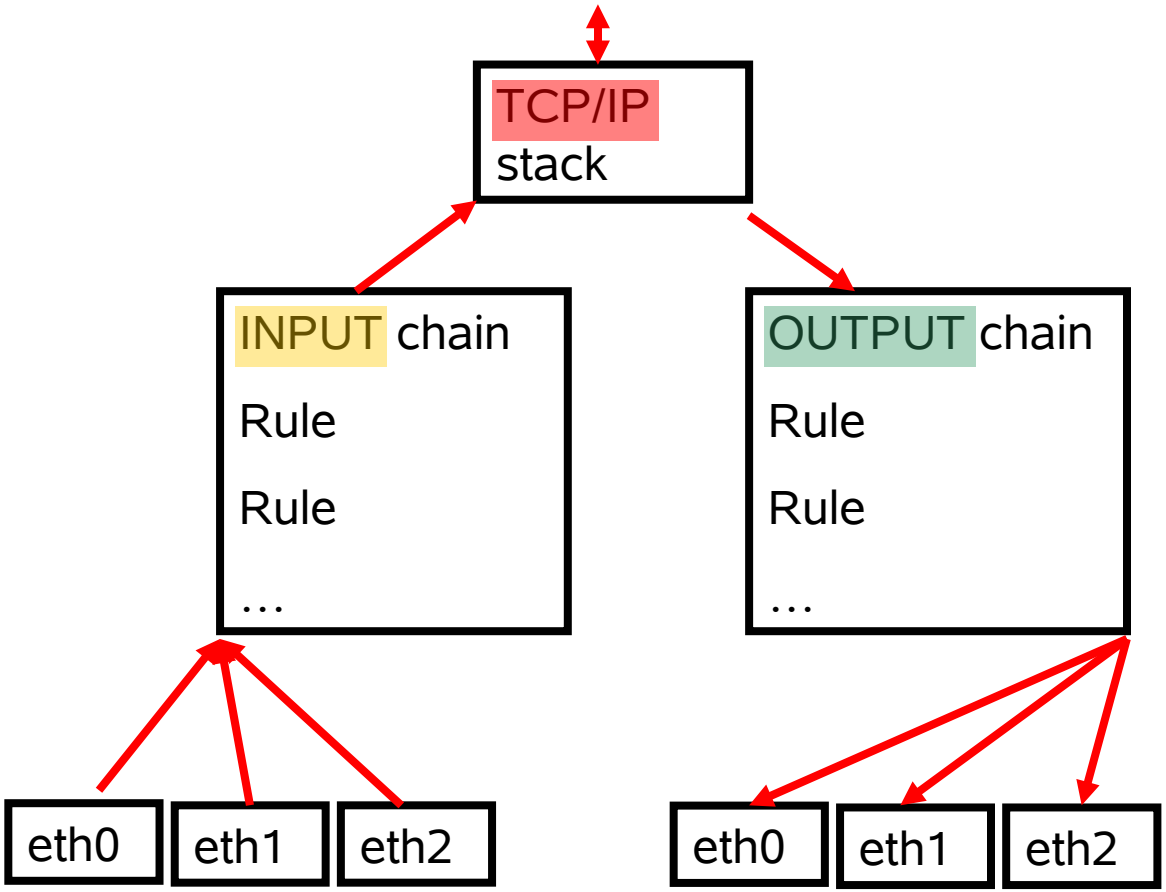
Plan for the presentation

Brief introduction to Linux IPtables filter rules

Two methods of defending against username and password break-in attempts

Simple defence against culling usernames via email

IPtables simple flow diagram



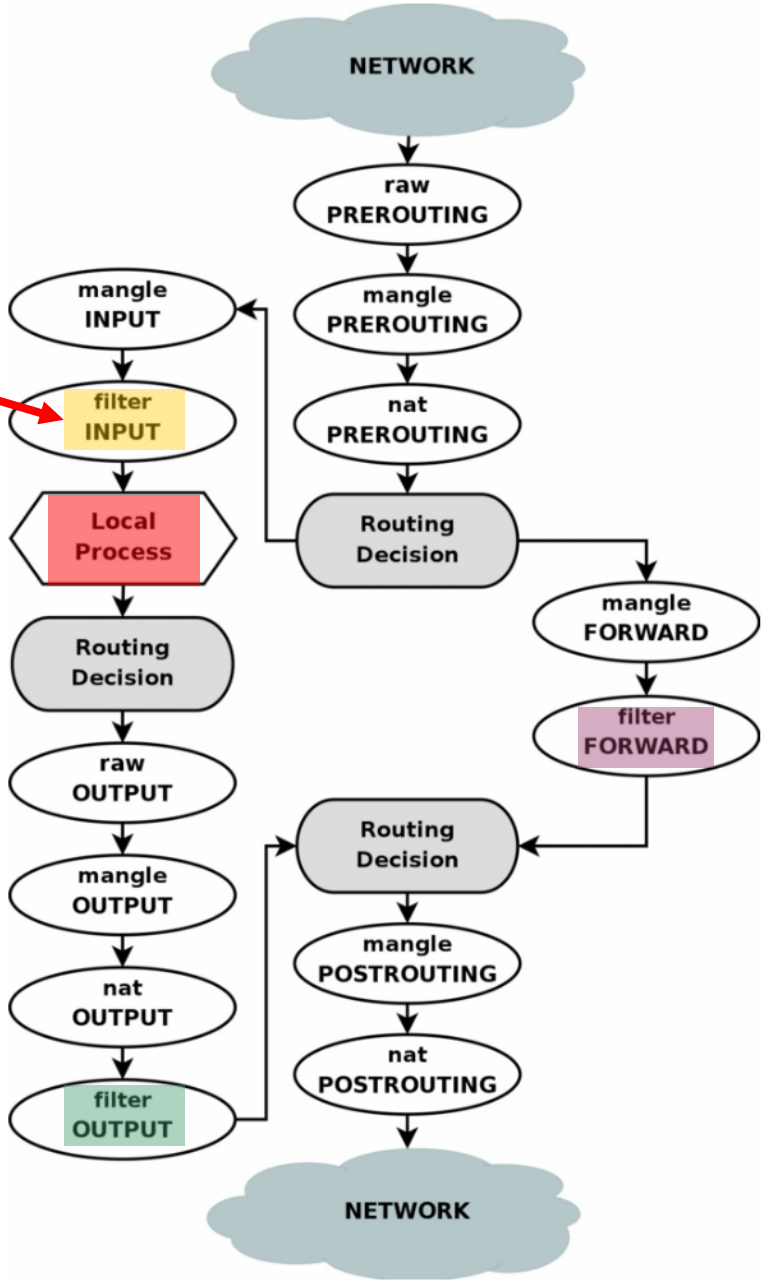
Rules cause packet to:

- go to its destination
- be dropped
- be passed to next rule in this chain
- be passed to another chain of rules

Full Topology

Our focus is here, the INPUT chain

TCP/IP stack



Where rule chains are located

"filter" is default table (has chains INPUT, OUTPUT, FORWARD).
Table "nat" is optional.
Ignore "raw" and "mangle".

http://iptables-tutorial.frozentux.net/chunkyhtml/c962.html

User space to kernel space

Program *`/usr/sbin/iptables`*

- **Converts text form of rules into kernel binary rules**
- **Reads kernel to display current rules as ordinary text**

My scripts usually define a shell variable P to represent *`/usr/sbin/iptables`*, thusly

```
P=/usr/sbin/iptables
```

$P filter-rule **passes that text to the kernel**

Rule syntax

-A INPUT -s 123.67.103.181/22 -p tcp -m tcp --dport 25 -j ACCEPT

Translation:

-A **is what to do with this rule (**-A **append to end of chain, versus**
-I **insert at the beginning)**

Chain-name (INPUT**) where this rule goes**

Matching tests, space separated (-s … 25 items above)

-j **target (what to do if all tests succeed)**

Else upon failure pass packet to next rule in the chain (thus use of the name "chain")

Jump targets (action)

-j ACCEPT **(allow into the TCP/IP stack or network)**

-j DROP **(discard silently)**

-j name **(go to another chain, typically user defined)**

-j LOG **(write message to syslog and continue with next rule)**

Plus some others which we do not need nor use:

-j REJECT **(discard, send back complaint packet)**

-j RETURN **(from user chain to its caller, done automatically at end)**

-j QUEUE **(very special and limited use)**

Choosing lan interface

Choosing an interface for traffic is done by test phrases

-i **interface-name** **for incoming traffic**

-o **interface-name** **for outgoing traffic**

Names of interfaces are viewable using program */sbin/ifconfig*; *lo* and *eth0* are most common, *eth1* is a second Ethernet adapter, etc.

Example:

-A INPUT -i lo -j ACCEPT

allows all traffic to flow through from the loopback interface named *lo*.

Source/destination IP numbers

Source IP number **(be wary if outside of your site)**

-s 1.2.3.4

Destination IP number

-d 1.2.3.4

Using the ! not operator

-A INPUT -d ! 129.67.103.181 -j DROP **(if not for me then drop)**

Netmask for a value range

-A INPUT -s 12.12.3.4/18 -j DROP

-A INPUT -d 23.23.4.5/255.255.254.0 -j ACCEPT

Specifying protocols, -p proto

-A INPUT -s 111.123.224.4 -p tcp -j ACCEPT

-A INPUT -s 111.123.224.4 -p icmp -j DROP

Details <u>within</u> a protocol header (say port number) require a protocol helper module, *-m protocol*

TCP/UDP source or dest port is *--sport #* or *--dest #*

-A INPUT -s 129.67.103.181/16 -p tcp -m tcp --dport 25 -j ACCEPT

Double dash for sub-options

If incoming is from 129.67.x.x and it is for TCP destination port 25 then accept it

More protocol test phrases

Specify many ports within one rule

-m multiport --sport port,port,port or --dport port,port,port

Double dash for sub-options

Replaces the *-m protocol* helper

-A INPUT -p tcp -m multiport --dport 80,443,21,23,25 -j ACCEPT

Setting up default actions

```
# P is the path/name of the iptables program, P6 is for IP v6
P=/usr/sbin/iptables
P6=/usr/sbin/ip6tables

# Set policy and zero counters
        $P -P INPUT DROP
        $P -P OUTPUT ACCEPT
        $P -P FORWARD ACCEPT
        $P -Z
# Flush existing rules in table filter
        $P -F
```

Policy: what to do with unmatched packets in non-user chains

Zero packet/byte counters

Table “filter” is built-in default and establishes chains INPUT, OUTPUT, FORWARD

Note: user-defined chains return to their invoker, like a procedure call, no Policy is possible

More setup: IPv6, FTP traffic

```
# IP v6, drop all traffic
#       $P6 -P INPUT DROP
#       $P6 -P OUTPUT DROP
#       $P6 -P FORWARD DROP
#       $P6 -F
```

Uncomment if IPv6 is active but you do not want its traffic

```
# Passive FTP requires kernel module ip_conntrack_ftp to be
# loaded by program modprobe, like this:

    /sbin/modprobe  ip_conntrack_ftp
```

Policies, zeroing, modprobe are done before specifying rules

Filtering strategy (simple form)

Allow unimpeded access from selected "friends"

Optionally reject multicast & broadcast traffic

Drop spoofed packets (source IP outside legit region)

Allow replies to our existing connections (TCP,UDP, ICMP, all done with one rule)

Be picky about which services (protocol & port) are visible to which source IP numbers/regions of world

- **use concept of regions of trust**

All done in that order, INPUT default action is DROP

Sample rule set

```
# local host, no restriction, bypass other rules
      $P -A INPUT -i lo -j ACCEPT

# Local subnet, trust it   Can use $MYIP/$NETMASK
      $P -A INPUT -s $MYIP/$NETMASK -j ACCEPT
      $P -A INPUT -s 163.1.2.0/24 -j ACCEPT

# Drop traffic not to our specific IP (unicast only)
      $P -A INPUT -d ! $MYIP -j DROP

# Drop spoofed internal addresses as sources
      $P -A INPUT -s $MYIP -j DROP
      $P -A INPUT -s 127.0.0.0/8 -j DROP

# Reverse path for existing TCP, UDP, ICMP connections
      $P -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT

# Services follow below, listed individually for clarity

# SSH world access
     $P -A INPUT -p tcp -m tcp --dport 22 -j ACCEPT

# FTP server, world access
      $P -A INPUT -p tcp -m tcp --dport 21 -j ACCEPT

# Web server, normal and SSL connections, world access
      $P -A INPUT  -p tcp -m multiport --dport 80,443 -j ACCEPT
```

$MYIP and **$NETMASK** are shell variables set earlier in the script

Completely trusted nets (friends)

Broad/multicast blockage

Anti-spoofing tests

Replies for existing connections

Individual services

Viewing active rules & stats

```
# iptables -L -vn
Chain INPUT (policy DROP 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination
278K  352M ACCEPT     all  --  lo     *       0.0.0.0/0            0.0.0.0/0
 121K   16M ACCEPT     all  --  *      *       129.67.100.0/22      0.0.0.0/0
19260 9354K ACCEPT     all  --  *      *       163.1.2.0/24         0.0.0.0/0
  2266  350K DROP       all  --  *      *       0.0.0.0/0           !129.67.103.180
    0     0 DROP       all  --  *      *       129.67.103.180       0.0.0.0/0
    0     0 DROP       all  --  *      *       127.0.0.0/8          0.0.0.0/0
2319K  176M ACCEPT     all  --  *      *       0.0.0.0/0            0.0.0.0/0
          state RELATED,ESTABLISHED
   14   828 fail2ban-SSH  tcp  --  *      *    0.0.0.0/0            0.0.0.0/0
          tcp dpt:22
    6   360 DROP       tcp  --  *      *       0.0.0.0/0            0.0.0.0/0
          tcp dpt:22 recent: UPDATE seconds: 180 hit_count: 3 name: SSH side: source
    8   468 ACCEPT     tcp  --  *      *       0.0.0.0/0            0.0.0.0/0
          tcp dpt:22 recent: SET name: SSH side: source
   33  1840 ACCEPT     tcp  --  *      *       0.0.0.0/0            0.0.0.0/0
          tcp dpt:21
 8353  527K ACCEPT     tcp  --  *      *       0.0.0.0/0            0.0.0.0/0
          multiport dports 80,443

Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination

Chain OUTPUT (policy ACCEPT 964K packets, 6371M bytes)
 pkts bytes target     prot opt in     out     source               destination

Chain fail2ban-SSH (1 references)
 pkts bytes target     prot opt in     out     source               destination
```

List, verbose, numeric form

friends

anti-broad/multicast

anti-spoofing

replies

blocker (log)

blocker (rate)

BBC world service

0.0.0.0/0 means any

blocker (log)

If NAT is active use iptables –L –vn –t nat to see the NAT tables

Includes intruder blockers, discussed later

NAT Topology

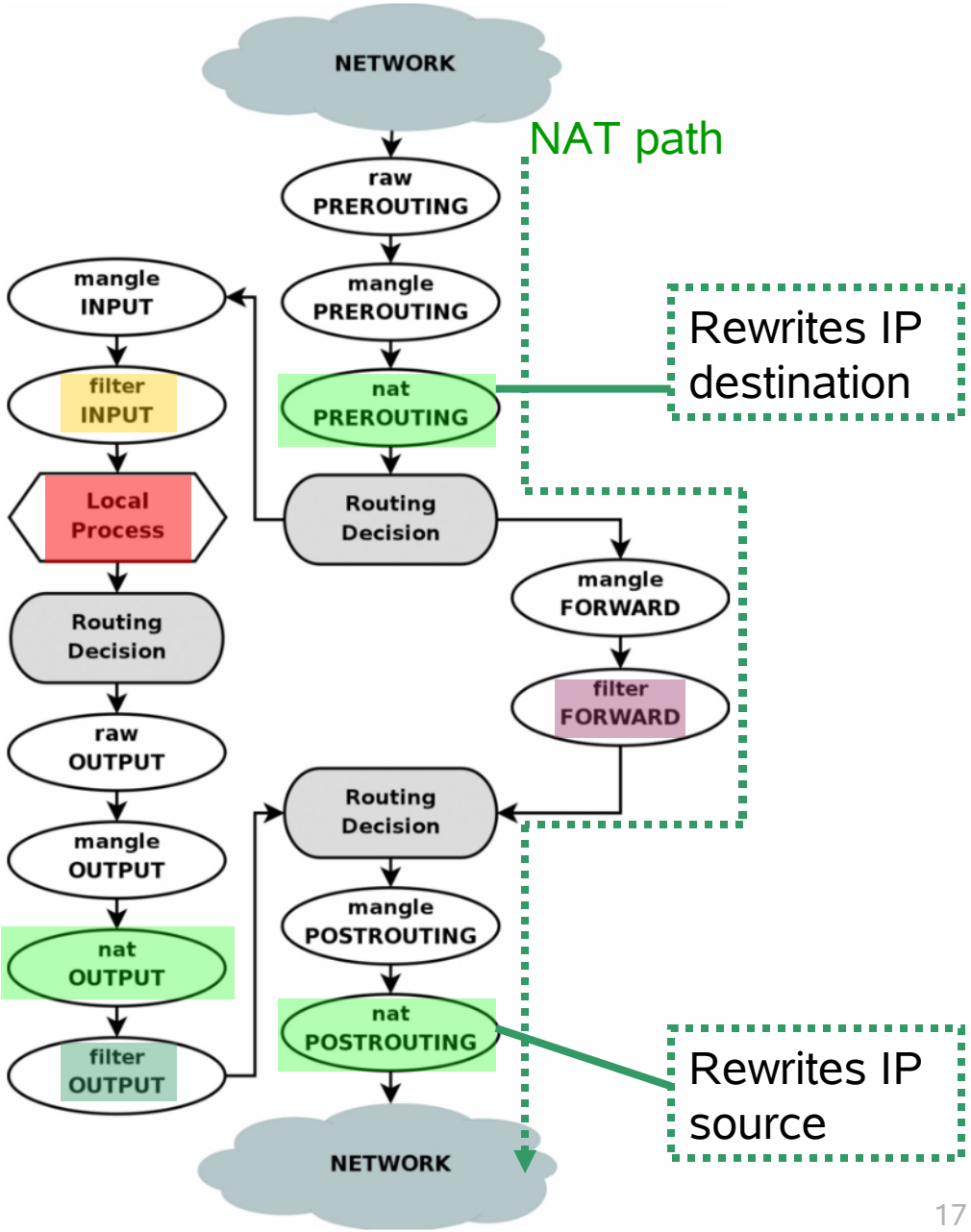
Our focus here is the NAT chain

Linux router

eth0 eth1

Internal network
10.10.0.254
10.10.0.0/24 net

World network
128.123.28.203
129.123.0.0/16 net



Source NAT rewrites packet source IP. Reverse traffic is automatically rewritten.

http://iptables-tutorial.frozentux.net/chunkyhtml/c962.html

Source NAT

Enable IP forwarding between interfaces (YaST, network)

Load NAT support module, in start) section

modprobe ip_nat **(or old name of** iptables_nat**)**

Setup NAT chain policy, in start) section (-t states table name)

```
$P -P FORWARD ACCEPT
$P -t nat -F                    (-F flushes old rules)
$P -t nat -P PREROUTING ACCEPT
$P -t nat -P POSTROUTING ACCEPT
$P -t nat -P OUTPUT ACCEPT
```

Add standalone NAT rule (eth0 is Internet interface):

```
$P -t nat -A POSTROUTING -o eth0 -j SNAT --to-source 129.123.28.203
```

(eth0's IP seen by world)

Add anti-spoofing rules etc with -i **interface, as desired**

A full IPtables script

Seven hidden slides following this one comprise a complete IPtables start/stop script with rules very similar to those shown previously

This script includes intruder blockers and small portions of NAT (full NAT material is in the IPtables product from MindworksUK)

Editing is required to match your local conditions

IPtables script, 1 of 7

```
File /etc/init.d/iptables
#! /bin/sh
# Firewall configuration written by Joe R. Doupnik
# Copyright (c) 2006 Joe R. Doupnik, MindworksUK
#
# Author: Joe R. Doupnik, joe.doupnik@oucs.ox.ac.uk
# Text modelled after /etc/init.d/lisa by SuSE
#
# This start/stop script also contains the filter rules so
# that the station may obtain its IP address from DHCP.
#
### BEGIN INIT INFO
# Provides:          iptables
# Required-Start:  network
# Required-Stop:
# Default-Start:      2 3 5
# Default-Stop:
# Description:       IPTABLES IP filter
### END INIT INFO
```

Chkconfig uses this section to manage rcN symlinks

Watch out for line wrapping in slides

IPtables script, 2 of 7

```
## activate the scripting status functions
. /etc/rc.status

# P is the path/name of the iptables program, P6 is for IP v6
P=/usr/sbin/iptables
# P6=/usr/sbin/ip6tables

rc_reset
case "$1" in
    start)
# Passive FTP requires kernel module ip_conntrack_ftp to be
# loaded by  modprobe ip_conntrack_ftp  before this table
# is loaded. Optional NAT module is ip_nat.
/sbin/modprobe ip_conntrack_ftp
/sbin/modprobe ip_nat

# imply internal table “filter”
# My IP number
##  If the address is static or assigned by dynamic DHCP we can use
#MYIP=`/usr/bin/resolveip -s $HOSTNAME`
## but only if MySQL is installed (resolveip is from it)
# If the above resolvip does not work, as recent SLES9 indicates, then use this
# horrid editing of the output of  ifconfig eth0  for our IP and netmask.
# Note: backtics (`) enclose a command to be executed by the shell
```

IPtables script, 3 of 7

```
MYIP=`ifconfig eth0 | grep 'inet addr:' | sed -e 's/ *inet addr://' \
  -e '1q' | awk '{print $1}'`
NETMASK=`ifconfig eth0 | grep 'inet addr:' | sed -e 's/ *inet addr://' \
  -e 's/Mask://' -e '1q' | awk '{print $3}'`

# Set policy and zero counters
        $P -P INPUT DROP
        $P -P OUTPUT ACCEPT
        $P -P FORWARD ACCEPT
        $P -Z
# Flush existing rules in table filter, and NAT tables
        $P -F
        $P -t nat -F
# IP v6, drop all traffic
#        $P6 -P INPUT DROP
#        $P6 -P OUTPUT DROP
#        $P6 -P FORWARD DROP
#        $P6 -F
# fail2ban SSH chain, create New user chain, Flush any old rules
        $P -N fail2ban-SSH
        $P -F fail2ban-SSH
```

IPtables script, 4 of 7

```
# local host, no restriction, bypass other rules
        $P -A INPUT -i lo -j ACCEPT

# Local subnets, trust them. Be careful here.
        $P -A INPUT -s $MYIP/$NETMASK -j ACCEPT
        $P -A INPUT -s 163.1.2.0/24 -j ACCEPT

# Drop traffic not to our specific IP (unicast only), may add src interface (-i eth0…)
        $P -A INPUT -d ! $MYIP -j DROP

# Drop spoofed internal addresses as sources, may add src interface (-i eth0…)
        $P -A INPUT -s $MYIP -j DROP
        $P -A INPUT -s 127.0.0.0/8 -j DROP

# Reverse path for existing TCP, UDP, ICMP connections
        $P -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT


# Example source NAT to external IP number on eth0
#       $P -t nat -A POSTROUTING  -o eth0  -j SNAT  --to-source $MYIP
```

IPtables script, 5 of 7

```
# Services follow below, listed individually for clarity
## SSH fail2ban form
        $P -A INPUT -p tcp -m tcp --dport 22 -j fail2ban-SSH
#       $P -A INPUT -p tcp -m tcp --dport 22 -j ACCEPT
# SSH world access, new logins added to list "SSH", dropped if over --hitcount
tries in --seconds
        $P -A INPUT -p tcp -m tcp --dport 22 -m recent --update --seconds 180
--hitcount 3 --name SSH -j DROP
        $P -A INPUT -p tcp -m tcp --dport 22 -m recent --set --name SSH -j ACCEPT

# FTP server, world access
        $P -A INPUT -p tcp -m tcp --dport 21 -j ACCEPT

# Web server, normal and SSL connections, world access
        $P -A INPUT  -p tcp -m multiport --dport 80,443 -j ACCEPT


## Start the SSH fail2ban daemon
        echo -n "Starting iptables: "
        rc_status -v
        echo -n "Starting fail2ban daemon: "
        /usr/local/bin/fail2ban-client -x start > /dev/null
        rc_status -v
        ;;
```

IPtables script, 6 of 7

```
stop)
        echo -n "Shutting down IP filter: "
        ## Flush rules from exiting chains (table is default, "filter")
        $P -F
        $P -t nat -F
#        $P6 -F
        ## set default Policy to be ACCEPT to open comms fully
        $P -P INPUT ACCEPT
        $P -P OUTPUT ACCEPT
        $P -P FORWARD ACCEPT
        $P -t nat -P PREROUTING ACCEPT
        $P -t nat -P POSTROUTING ACCEPT
        $P -t nat -P OUTPUT ACCEPT
#        $P6 -P INPUT ACCEPT
#        $P6 -P OUTPUT ACCEPT
#        $P6 -P FORWARD ACCEPT
        rc_status -v
        ## Stop the SSH fail2ban daemon, remove its chain
        echo -n "Shutting down daemon fail2ban: "
        /usr/local/bin/fail2ban-client stop > /dev/null
        $P -X fail2ban-SSH
        rc_status -v
        ;;
```

IPtables script, 7 of 7

```
restart)
        ## If first returns OK call the second, if first or
        ## second command fails, set echo return value.
        $0 stop; $0 start
        rc_status
        ;;
reload)
        $0 stop; $0 start
        rc_status
        ;;
status)
        echo "Checking IPTABLES:"
        /usr/sbin/iptables -L -n
        rc_status -v
        ;;
    *)
        echo "Usage: $0 {start|stop|status|restart|reload}"
        exit 1
        ;;
esac
rc_exit
```

Blocking frequent guessers

Three attack variants

Two tools:

1. **IPtables itself, as a packet rate detector**
2. **Fail2ban, log scanner for multiple services, IPtables rules**

Email, which does not fit the mould above

Simple attack #1: p/w guessing

```
ssh username@example.com
    password foobar1      (fails)
    password foobar2      (fails)
    password foobar3      (fails) ...
```

All in *one* connection to an acceptable username

Detection of p/w failures is by the service, possibly revealed to a blocker via a log file

`man sshd_config` for SLES 10

`MaxAuthTries`

`Specifies the maximum number of authentication attempts permitted per connection. Once the number of failures reaches half this value, additional failures are logged. The default is 6.`

(NOTE: PAM auth logs each password failure on SLES 10)

Simple attack #2: user guessing

```
03:34:14 netlab1 sshd[662]: Invalid user tiger from 64.62.31.170
03:34:22 netlab1 sshd[740]: Invalid user xxx from 64.62.31.170
03:34:25 netlab1 sshd[764]: Invalid user money from 64.62.31.170
03:34:30 netlab1 sshd[796]: Invalid user test from 64.62.31.170
03:34:34 netlab1 sshd[814]: Invalid user teste from 64.62.31.170
```

Each try typically is a *new* TCP connection

Blocker can detect

- **too many connection start ups**
- **too many authentication failures**

over a reasonable period of time

Massive attack: multiple sources

Attacker tries different usernames, each from a different IP address (botnet stuff)

Once a valid username is found then it starts guessing passwords within a single session

Very difficult to defend against this without shutting down a service for a long period

Detecting an attack

Application itself reaches password retry limit

- **Does not retain state to detect repeated attacks**

Blocker reads a log and imposes block on a service

- **Can block very late via tcpwrapper *hosts.allow* (for some services and those run via *xinetd*), not so good**

- **Can block in *IPtables* for a true black hole, using log files to reveal failed connections, keeps state, has duration over time (any number of attempts)**

The defence is to cut off retries and obstruct enough that intruders go away (for awhile)

Advice on IPtables blocking

Blockers using *IPtables* should be restricted to just adding or removing their own DROP rules, nothing else

Ordinary service admission rule:

$P -A INPUT -p tcp -m tcp --dport 22 -j ACCEPT

Precede it with a chain of DROP rules:

$P -A INPUT -p tcp -m tcp --dport 22 -j BLACKLIST

$P -A INPUT -p tcp -m tcp --dport 22 -j ACCEPT

DROP rules go into this user chain, falls through if no match

Debug to log */var/log/firewall*, if desired. Place before rules above

$P -A INPUT -p tcp -m tcp --dport 22 -j LOG --log-prefix “some text”

$P is */usr/sbin/iptables*

Tool #1

IPtables alone:

packet rate detection and blocking

Detecting an attack: hit rate

IPtables itself can detect excessive number of packets in a selected time span and impose a temporary block

Must be careful to detect only new connections (pass through traffic for existing connections prior to rate check)

Technique does not know about usernames and passwords, just packets and IP details

IPtables hit rate filter

<other rules, existing connections passed through up here>

```
# SSH world access, new logins added to list "SSH", dropped if over 5 tries in 60
    seconds
$P -A INPUT -p tcp -m tcp --dport 22    -m recent --update --name SSH
    --seconds 60 --hitcount 5  -j DROP
$P -A INPUT -p tcp -m tcp --dport 22    -m recent --set --name SSH  -j ACCEPT
```

First rule uses module (-m) “recent” to update a counter (SSH) for the source IP address, and if more than “hitcount” visits in past “seconds” then DROP the packet

Second rule adds/updates count & time and accepts the packet. This rule executes only if first rule fails (no DROP).

See man iptables, also http://snowman.net/projects/ipt_recent/ and http://iptables-tutorial.frozentux.net/chunkyhtml/x2702.html#RECENTMATCH

Shell variable $P is defined as */usr/sbin/iptables*

IPtables hit rate filter

This hit counter approach works well to block overly frequent login attempts

It is easily repeated for other services, each operating with an independent counter (--name)

State is kept in the kernel, *proc/net/ipt_recent*, for each --name

echo clear > /proc/net/ipt_recent/SSH	**to clear name SSH**
cat /proc/net/ipt_recent/SSH	**shows state table**

src=**83.96.250.72** ttl: 56 last_seen: 4298867113 oldest_pkt: 7 last_pkts: 4298762657, 4298865910, 4298866611, 4298866737, 4298866861, 4298866987, 4298867113

Details on module recent

Command was iptables -m recent --help

```
Recent v1.3.5 options:
[!] --set                 Add source address to list, always matches.
[!] --rcheck              Match if source address in list.
[!] --update              Match if source address in list, also update last-seen time.
[!] --remove              Match if source address in list, also removes that address from list.
    --seconds seconds     For check and update commands above.
                          Specifies that the match will only occur if source address last seen within
                          the last 'seconds' seconds.
    --hitcount hits       For check and update commands above.
                          Specifies that the match will only occur if source address seen hits times.
                          May be used in conjunction with the seconds option.
    --rttl                For check and update commands above.
                          Specifies that the match will only occur if the source address and the TTL
                          match between this packet and the one which was set.
                          Useful if you have problems with people spoofing their source address in order
                          to DoS you via this module.
    --name name           Name of the recent list to be used.  DEFAULT used if none given.
    --rsource             Match/Save the source address of each packet in the recent list table (default).
    --rdest               Match/Save the destination address of each packet in the recent list table.

ipt_recent v0.3.1: Stephen Frost <sfrost@snowman.net>.  http://snowman.net/projects/ipt_recent/
```

Tool #2

Fail2ban:

read logs, protect multiple services

Fail2ban, multiple services

- Log file patterns for many different services
- Python, be wary of syntax and punctuation
- Client-server model (can change setting in real time)
- Server polls log files, scans for patterns
- Handles log rotation, has white lists
- Pattern match results in an IPtables DROP rule
- Each service has its own configuration file (log to scan, patterns, actions, IPtables chain)
- State kept in server daemon

http://www.fail2ban.org/

Fail2ban

Configuration files have much detail

- **IPtables rules upon startup, match, unblocking, etc**
- **These need local construction refinements**

Service specific files have array of patterns to match

Overall configuration file *`/etc/jail.conf`* has list of services to enable/disable, with filters & actions

Fail2ban, part of file jail.conf

Values for *name, port, protocol* passed to action table **iptables.conf**

```
[ssh-iptables]

enabled  = true
filter   = sshd
action   = iptables[name=SSH, port=ssh, protocol=tcp]
##          sendmail-whois[name=SSH, dest=you@mail.com, sender=fail2ban@mail.com]
logpath  = /var/log/messages
maxretry = 5
```

Filter file

Log file to monitor

Blocking threshold

No mail, tnx

```
[proftpd-iptables]

enabled  = false
filter   = proftpd
action   = iptables[name=ProFTPD, port=ftp, protocol=tcp]
           sendmail-whois[name=ProFTPD, dest=you@mail.com]
logpath  = /var/log/proftpd/proftpd.log
maxretry = 6
```

Please read */etc/fail2ban/jail.conf* for full set of options

Fail2ban main options

```
# "ignoreip" can be an IP address, a CIDR mask or a DNS host, space separated list
ignoreip = 127.0.0.1
```

White list

```
# "bantime" is the number of seconds that a host is banned.
bantime  = 600
```

Lifetime of a DROP rule

```
# A host is banned if it has generated "maxretry" during the last "findtime"
# seconds.
findtime  = 600
```

Interval over which log file is scanned to locate failures

```
# "maxretry" is the number of failures before a host get banned.
maxretry = 3
```

Global values, allows per-service overrides

Revised iptables.conf actions (part)

actionstart =
iptables -N fail2ban-<name>
iptables -A fail2ban-<name> -j RETURN
iptables -I INPUT -p <protocol> --dport <port> -j fail2ban-<name>

<name> <protocol> <port> are passed from jail.conf. <name> is part of chain name (a chain per service)

Do nothing up here

actionstop =
iptables -F fail2ban-<name>
iptables -X fail2ban-<name>
iptables -D INPUT -p <protocol> --dport <port> -j fail2ban-<name>

actionban = iptables -I fail2ban-<name> 1 -s <ip> -j DROP

actionunban = iptables -D fail2ban-<name> -s <ip> -j DROP

Rule insert & delete *IPtables* commands

chain

src IP from log scan

Chain setup, teardown, clearing is done in */etc/init.d/iptables*
Manager specifies exactly where the DROP chain is placed

A variety of service filter patterns

ls /etc/fail2ban/filter.d

apache-auth.conf
apache-badbots.conf
apache-noscript.conf
apache-overflows.conf
common.conf
courierlogin.conf
couriersmtp.conf
exim.conf
named-refused.conf
postfix.conf
proftpd.conf
pure-ftpd.conf
qmail.conf
sasl.conf
sshd-ddos.conf
sshd.conf
vsftpd.conf
webmin-auth.conf
wuftpd.conf
xinetd-fail.conf

Others are easily constructed from these examples

sshd.conf filter file

```
# Option:  failregex
# Notes.:  regex to match the password failures messages in the logfile. The
#          host must be matched by a group named "host". The tag "<HOST>" can
#          be used for standard IP/hostname matching and is only an alias for
#          (?:::f{4,6}:)?(?P<host>\S+)
# Values:  TEXT
#
failregex = ^%(__prefix_line)s(?:error: PAM: )?Authentication failure for .* from <HOST>\s*$
            ^%(__prefix_line)sFailed [-/\w]+ for .* from <HOST>(?: port \d*)?(?:ssh\d*)?$
            ^%(__prefix_line)sROOT LOGIN REFUSED.* FROM <HOST>\s*$
            ^%(__prefix_line)s[iI](?:llegal|nvalid) user .* from <HOST>\s*$
            ^%(__prefix_line)sUser \S+ from <HOST> not allowed because not listed in AllowUsers$
            ^%(__prefix_line)sauthentication failure; logname=\S* uid=\S* euid=\S* tty=\S* ruser=\S*rhost=<HOST>(?:\s+user=.*)?\s*$
            ^%(__prefix_line)srefused connect from \S+ \(<HOST>\)\s*$
            ^%(__prefix_line)sAddress <HOST> .* POSSIBLE BREAK-IN ATTEMPT\s*$
            ^%(__prefix_line)sDid not receive identification string from <HOST>\s*$

# Option:  ignoreregex
# Notes.:  regex to ignore. If this regex matches, the line is ignored.
# Values:  TEXT
ignoreregex=
```

Pattern added by me

Some experience with Python regular expressions and syntax can be helpful

%(__prefix_line)s has protocol name picking and much other common pattern stuff

My /etc/init.d/iptables snippets

Start) area -

```
# fail2ban-SSH chain
        $P -N fail2ban-SSH                                  setup and flush new (-N) chain fail2ban-SSH
        $P -F fail2ban-SSH

                <Trusted hosts, anti-spoofing, pass existing sessions, other services appear here>
                Rules below here refer to starting new connections

## SSH fail2ban form
        $P -A INPUT -p tcp -m tcp --dport 22 -j fail2ban-SSH    holds DROP rules, done if a match,
        $P -A INPUT -p tcp -m tcp --dport 22 -j ACCEPT          else come here to accept packet

## Start the SSH fail2ban daemon
        echo -n "Starting fail2ban daemon: "
        /usr/local/bin/fail2ban-client –x start > /dev/null        start file2ban server via the client
        rc_status –v

Stop) area -
## Stop the SSH fail2ban daemon, remove its chain
        echo -n "Shutting down fail2ban daemon: "
        /usr/local/bin/fail2ban-client stop > /dev/null     stop fail2ban server via the client
        $P -X fail2ban-SSH                                  remove the user chain
        rc_status –v
```

IPtables after one SSH block

```
# iptables -L -vn
Chain INPUT (policy DROP 3 packets, 144 bytes)
 pkts bytes target     prot opt in     out     source               destination

  541  105K ACCEPT     all  --  lo     *       0.0.0.0/0            0.0.0.0/0
    1    76 ACCEPT     all  --  *      *       162.5.2.17           0.0.0.0/0
    0     0 ACCEPT     all  --  *      *       86.97.123.0/29       0.0.0.0/0
    0     0 ACCEPT     all  --  *      *       62.4.220.0/24        0.0.0.0/0
    0     0 ACCEPT     all  --  *      *       10.0.0.0/24          0.0.0.0/0
  328 54891 DROP       all  --  *      *       0.0.0.0/0           !129.67.101.41
    0     0 DROP       all  --  *      *       129.67.101.41        0.0.0.0/0
    0     0 DROP       all  --  *      *       127.0.0.0/8          0.0.0.0/0
  259 20283 ACCEPT     all  --  *      *       0.0.0.0/0            0.0.0.0/0 state RELATED,ESTABLISHED
    2   108 fail2ban-SSH  tcp  --  *      *    0.0.0.0/0            0.0.0.0/0           tcp dpt:22
    2   108 ACCEPT     tcp  --  *      *       0.0.0.0/0            0.0.0.0/            tcp dpt:22
    0     0 ACCEPT     tcp  --  *      *       0.0.0.0/0            0.0.0.0/            tcp dpt:21
    0     0 ACCEPT     tcp  --  *      *       129.67.1.0/24        0.0.0.0/            tcp dpt:25
    0     0 ACCEPT     tcp  --  *      *       0.0.0.0/0            0.0.0.0/0 multiport dports 80,443

Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination


Chain OUTPUT (policy ACCEPT 739 packets, 129K bytes)
 pkts bytes target     prot opt in     out     source               destination


Chain fail2ban-SSH (1 references)
 pkts bytes target     prot opt in     out     source               destination

    0     0 DROP       all  --  *      *       129.67.101.23        0.0.0.0/0
```

trusted hosts

unicast anti-spoofing

services

Rule goes away after timeout interval *bantime*

But is this enough? Sigh, no

No.	Time	Source	Destination	Protocol	Info
76	4.670845	88.96.125.1	205.158.108.254	TCP	ssh > 41500 [ACK] Seq=1190 Ack=385 Win=7936 Len=0 TSV=2325170 TSER=1300301194
77	4.670918	88.96.125.1	205.158.108.254	SSHv2	Server: Unknown (86)
78	4.855308	205.158.108.254	88.96.125.1	SSHv2	Encrypted request packet len=84
79	4.856057	88.96.125.1	205.158.108.254	SSHv2	Encrypted response packet len=68
80	5.034901	205.158.108.254	88.96.125.1	SSHv2	Encrypted request packet len=52
81	5.035338	205.158.108.254	88.96.125.1	TCP	41500 > ssh [FIN, ACK] Seq=521 Ack=1310 Win=8704 Len=0 TSV=1300301558 TSER=2325216
82	5.035998	205.158.108.254	88.96.125.1	TCP	41563 > ssh [SYN] Seq=0 Ack=0 Win=5840 Len=0 MSS=1300 TSV=1300301558 TSER=0 WS=7
83	5.036027	88.96.125.1	205.158.108.254	TCP	ssh > 41563 [SYN, ACK] Seq=0 Ack=1 Win=5792 Len=0 MSS=1460 TSV=2325261 TSER=1300301558 WS=2
84	5.036033	88.96.125.1	205.158.108.254	TCP	ssh > 41500 [FIN, ACK] Seq=1310 Ack=522 Win=7936 Len=0 TSV=2325261 TSER=1300301558
85	5.213818	205.158.108.254	88.96.125.1	TCP	41563 > ssh [ACK] Seq=1 Ack=1 Win=5888 Len=0 TSV=1300301737 TSER=2325261
86	5.217000	205.158.108.254	88.96.125.1	TCP	41500 > ssh [ACK] Seq=522 Ack=1311 Win=8704 Len=0 TSV=1300301740 TSER=2325261
87	5.225891	88.96.125.1	205.158.108.254	SSH	Server Protocol: SSH-1.99-OpenSSH_4.2
88	5.404522	205.158.108.254	88.96.125.1	TCP	41563 > ssh [ACK] Seq=1 Ack=22 Win=5888 Len=0 TSV=1300301927 TSER=2325309
89	5.404728	205.158.108.254	88.96.125.1	SSH	Client Protocol: SSH-2.0-libssh-0.1\r
90	5.404758	88.96.125.1	205.158.108.254	TCP	ssh > 41563 [ACK] Seq=22 Ack=21 Win=5792 Len=0 TSV=2325354 TSER=1300301927
91	5.405958	88.96.125.1	205.158.108.254	SSHv2	Server: Key Exchange Init
92	5.600386	205.158.108.254	88.96.125.1	SSHv2	Client: Key Exchange Init
93	5.637609	88.96.125.1	205.158.108.254	TCP	ssh > 41563 [ACK] Seq=726 Ack=173 Win=6864 Len=0 TSV=2325412 TSER=1300302123
94	5.816473	205.158.108.254	88.96.125.1	SSHv2	Client: Diffie-Hellman Key Exchange Init
95	5.816492	88.96.125.1	205.158.108.254	TCP	ssh > 41563 [ACK] Seq=726 Ack=317 Win=7936 Len=0 TSV=2325456 TSER=1300302340
96	5.820832	88.96.125.1	205.158.108.254	SSHv2	Server: New Keys
97	6.011302	205.158.108.254	88.96.125.1	SSHv2	Client: New Keys
98	6.049606	88.96.125.1	205.158.108.254	TCP	ssh > 41563 [ACK] Seq=1190 Ack=333 Win=7936 Len=0 TSV=2325515 TSER=1300302535
99	6.228892	205.158.108.254	88.96.125.1	SSHv2	41563 > ssh [PSH, ACK] Seq=333 Ack=1190 Win=8704 Len=52 TSV=1300302750 TSER=2325515[Malformed Pack
100	6.228932	88.96.125.1	205.158.108.254	TCP	ssh > 41563 [ACK] Seq=1190 Ack=385 Win=7936 Len=0 TSV=2325560 TSER=1300302750
101	6.229023	88.96.125.1	205.158.108.254	SSHv2	Server: Unknown (250)
102	6.410966	205.158.108.254	88.96.125.1	SSHv2	Encrypted request packet len=84
103	6.411787	88.96.125.1	205.158.108.254	SSHv2	Encrypted response packet len=68
104	6.591281	205.158.108.254	88.96.125.1	SSHv2	Encrypted request packet len=52
105	6.591682	205.158.108.254	88.96.125.1	TCP	41563 > ssh [FIN, ACK] Seq=521 Ack=1310 Win=8704 Len=0 TSV=1300303115 TSER=2325605
106	6.592276	205.158.108.254	88.96.125.1	TCP	41621 > ssh [SYN] Seq=0 Ack=0 Win=5840 Len=0 MSS=1300 TSV=1300303116 TSER=0 WS=7
107	6.592302	88.96.125.1	205.158.108.254	TCP	ssh > 41621 [SYN, ACK] Seq=0 Ack=1 Win=5792 Len=0 MSS=1460 TSV=2325650 TSER=1300303116 WS=2
108	6.592377	88.96.125.1	205.158.108.254	TCP	ssh > 41563 [FIN, ACK] Seq=1310 Ack=522 Win=7936 Len=0 TSV=2325650 TSER=1300303115
109	6.769879	205.158.108.254	88.96.125.1	TCP	41621 > ssh [ACK] Seq=1 Ack=1 Win=5888 Len=0 TSV=1300303294 TSER=2325650

1.5 sec

SSH probes, 1.5 seconds between TCP connections

Only 1 log entry over 60++ sec of busy traffic (saturated SSH crypto)

Normal SSH login

No.	Time	Source	Destination	Protocol	Info
1	0.000000	10.0.0.1	10.0.0.20	TCP	48651 > ssh [SYN] Seq=0 Ack=0 Win=5840 Len=0 MSS=1460 TSV=2514513 TSER=0 WS=2
2	0.000036	10.0.0.20	10.0.0.1	TCP	ssh > 48651 [SYN, ACK] Seq=0 Ack=1 Win=5792 Len=0 MSS=1460 TSV=2422385 TSER=2514513 WS=2
3	0.000264	10.0.0.1	10.0.0.20	TCP	48651 > ssh [ACK] Seq=1 Ack=1 Win=5840 Len=0 TSV=2514513 TSER=2422385
4	0.030495	10.0.0.20	10.0.0.1	SSH	Server Protocol: SSH-1.99-OpenSSH_4.2
5	0.020707	10.0.0.1	10.0.0.20	TCP	48651 > ssh [ACK] Seq=1 Ack=22 Win=5840 Len=0 TSV=2514518 TSER=2422390
6	0.020811	10.0.0.1	10.0.0.20	SSH	Client Protocol: SSH-2.0-OpenSSH_4.2
7	0.020823	10.0.0.20	10.0.0.1	TCP	ssh > 48651 [ACK] Seq=22 Ack=21 Win=5792 Len=0 TSV=2422390 TSER=2514518
8	0.022694	10.0.0.1	10.0.0.20	SSHv2	Client: Key Exchange Init
9	0.022715	10.0.0.20	10.0.0.1	TCP	ssh > 48651 [ACK] Seq=22 Ack=733 Win=7216 Len=0 TSV=2422390 TSER=2514519
10	0.022904	10.0.0.20	10.0.0.1	SSHv2	Server: Key Exchange Init
11	0.023072	10.0.0.1	10.0.0.20	SSHv2	Client: Diffie-Hellman GEX Request
12	0.025705	10.0.0.20	10.0.0.1	SSHv2	Server: Diffie-Hellman Key Exchange Reply
13	0.027381	10.0.0.1	10.0.0.20	SSHv2	Client: Diffie-Hellman GEX Init
14	0.029489	10.0.0.20	10.0.0.1	SSHv2	Server: Diffie-Hellman GEX Reply
15	0.030669	10.0.0.1	10.0.0.20	SSHv2	Client: New Keys
16	0.067560	10.0.0.20	10.0.0.1	TCP	ssh > 48651 [ACK] Seq=1342 Ack=917 Win=8640 Len=0 TSV=2422402 TSER=2514521
17	0.067802	10.0.0.1	10.0.0.20	SSHv2	Encrypted request packet len=48
18	0.067816	10.0.0.20	10.0.0.1	TCP	ssh > 48651 [ACK] Seq=1342 Ack=965 Win=8640 Len=0 TSV=2422402 TSER=2514530
19	0.067938	10.0.0.20	10.0.0.1	SSHv2	Encrypted response packet len=48
20	0.068108	10.0.0.1	10.0.0.20	SSHv2	Encrypted request packet len=64
21	0.079933	10.0.0.20	10.0.0.2	DNS	Standard query PTR 1.0.0.10.in-addr.arpa
22	0.080324	10.0.0.20	10.0.0.1	SSHv2	Encrypted response packet len=64
23	0.080906	10.0.0.1	10.0.0.20	SSHv2	Encrypted request packet len=96
24	0.119532	10.0.0.20	10.0.0.1	TCP	ssh > 48651 [ACK] Seq=1454 Ack=1125 Win=8640 Len=0 TSV=2422415 TSER=2514531
25	0.132077	10.0.0.2	10.0.0.20	DNS	Standard query response, No such name
26	0.140594	10.0.0.20	10.0.0.1	SSHv2	Encrypted response packet len=64
27	0.179562	10.0.0.1	10.0.0.20	TCP	48651 > ssh [ACK] Seq=1125 Ack=1518 Win=10064 Len=0 TSV=2514548 TSER=2422420
28	4.537560	10.0.0.1	10.0.0.20	SSHv2	Encrypted request packet len=80
29	4.537581	10.0.0.20	10.0.0.1	TCP	ssh > 48651 [ACK] Seq=1518 Ack=1205 Win=8640 Len=0 TSV=2423520 TSER=2515643
30	4.541942	10.0.0.20	10.0.0.1	SSHv2	Encrypted response packet len=48
31	4.542178	10.0.0.1	10.0.0.20	TCP	48651 > ssh [ACK] Seq=1205 Ack=1566 Win=10064 Len=0 TSV=2515644 TSER=2423521
32	4.542210	10.0.0.1	10.0.0.20	SSHv2	Encrypted request packet len=80
33	4.542843	10.0.0.20	10.0.0.1	SSHv2	Encrypted response packet len=32
34	4.544321	10.0.0.1	10.0.0.20	SSHv2	Encrypted request packet len=64
35	4.544860	10.0.0.20	10.0.0.1	SSHv2	Encrypted response packet len=48
36	4.545247	10.0.0.1	10.0.0.20	SSHv2	Encrypted request packet len=528
37	4.548788	10.0.0.20	10.0.0.1	SSHv2	Encrypted response packet len=48
38	4.548860	10.0.0.20	10.0.0.1	SSHv2	Encrypted response packet len=112
39	4.549023	10.0.0.1	10.0.0.20	TCP	48651 > ssh [ACK] Seq=1877 Ack=1806 Win=10064 Len=0 TSV=2515646 TSER=2423522
40	4.652428	10.0.0.20	10.0.0.1	SSHv2	Encrypted response packet len=48
41	4.685245	10.0.0.1	10.0.0.20	TCP	48651 > ssh [ACK] Seq=1877 Ack=1854 Win=10064 Len=0 TSV=2515683 TSER=2423548
42	5.077041	linux.jrdresearch	10.0.0.2	ARP	Who has 10.0.0.2? Tell 10.0.0.20
43	5.077267	10.0.0.2	linux.jrdresearch	ARP	10.0.0.2 is at 00:50:56:f1:db:8a

Short (70ms), only one TCP connection

Two can be better than one

Combine log file reader with hit rate counter

```
## fail2ban SSH form
$P -A INPUT -p tcp -m tcp --dport 22 -j fail2ban-SSH
##  $P -A INPUT -p tcp -m tcp --dport 22 -j ACCEPT

# SSH world access, new logins added to list "SSH", dropped if over 5 tries in 60
    seconds
$P -A INPUT -p tcp -m tcp --dport 22 -m recent --update --seconds 60
    --hitcount 5 --name SSH -j DROP
$P -A INPUT -p tcp -m tcp --dport 22 -m recent --set --name SSH -j ACCEPT
```

Hit rate counter catches items too quick or obscure for logging

Example: hitcount 15 + logging

 # cat /proc/net/ipt_recent/SSH — **IPtables "recent" kernel table**

src=61.237.230.10 ttl: 43 last_seen: 4839785126 oldest_pkt: 6 last_pkts: 4839639844, 4839780290, 4839781641, 4839782441, 4839784202, 4839785126

 # cat /var/log/firewall — **IPtables LOG command to syslog**

Poke & wait

Jun 13 15:37:37 netlab1 kernel: SSH before hitcounter IN=eth0 OUT= MAC=00:0d:60:16:8e:72:00:d0:ff:ef:a8:00:08:00 SRC=61.237.230.10 DST=129.67.103.180 LEN=60 TOS=0x00 PREC=0x00 TTL=43 ID=54906 PROTO=TCP SPT=36208 DPT=22 WINDOW=5840 RES=0x00 SYN URGP=0

Break in attempts

Jun 13 15:46:59 netlab1 kernel: SSH before hitcounter IN=eth0 OUT= MAC=00:0d:60:16:8e:72:00:d0:ff:ef:a8:00:08:00 SRC=61.237.230.10 DST=129.67.103.180 LEN=60 TOS=0x00 PREC=0x00 TTL=43 ID=39250 PROTO=TCP SPT=53239 DPT=22 WINDOW=5840 RES=0x00 SYN URGP=0

Jun 13 15:47:04 netlab1 kernel: SSH before hitcounter IN=eth0 OUT= MAC=00:0d:60:16:8e:72:00:d0:ff:ef:a8:00:08:00 SRC=61.237.230.10 DST=129.67.103.180 LEN=60 TOS=0x00 PREC=0x00 TTL=43 ID=16186 PROTO=TCP SPT=53526 DPT=22 WINDOW=5840 RES=0x00 SYN URGP=0

Jun 13 15:47:07 netlab1 kernel: SSH before hitcounter IN=eth0 OUT= MAC=00:0d:60:16:8e:72:00:d0:ff:ef:a8:00:08:00 SRC=61.237.230.10 DST=129.67.103.180 LEN=60 TOS=0x00 PREC=0x00 TTL=43 ID=29431 PROTO=TCP SPT=38613 DPT=22 WINDOW=5840 RES=0x00 SYN URGP=0

Jun 13 15:47:14 netlab1 kernel: SSH before hitcounter IN=eth0 OUT= MAC=00:0d:60:16:8e:72:00:d0:ff:ef:a8:00:08:00 SRC=61.237.230.10 DST=129.67.103.180 LEN=60 TOS=0x00 PREC=0x00 TTL=43 ID=40536 PROTO=TCP SPT=38972 DPT=22 WINDOW=5840 RES=0x00 SYN URGP=0

Jun 13 15:47:18 netlab1 kernel: SSH before hitcounter IN=eth0 OUT= MAC=00:0d:60:16:8e:72:00:d0:ff:ef:a8:00:08:00 SRC=61.237.230.10 DST=129.67.103.180 LEN=60 TOS=0x00 PREC=0x00 TTL=43 ID=62007 PROTO=TCP SPT=39153 DPT=22 WINDOW=5840 RES=0x00 SYN URGP=0

 # grep 61.237.230.10 /var/log/messages — **SSHD reports to syslog**

Jun 13 15:37:37 netlab1 sshd[1038]: Did not receive identification string from 61.237.230.10 — Crypto probe

Jun 13 15:47:13 netlab1 sshd[5816]: Invalid user stud from 61.237.230.10

Jun 13 15:47:17 netlab1 sshd[6200]: Invalid user trash from 61.237.230.10 — Username guessing

Jun 13 15:47:22 netlab1 sshd[6270]: Invalid user aaron from 61.237.230.10

 # tail /var/log/fail2ban.log — **Fail2ban reports to its log**

2008-06-13 15:47:23,277 fail2ban.actions: WARNING [ssh-iptables] Ban 61.237.230.10 ← This is the block

2008-06-13 15:57:23,356 fail2ban.actions: WARNING [ssh-iptables] Unban 61.237.230.10

Note lag time in the log reports, small TTL from canned tool

SMTP mail, a special case

Bad persons can easily discover all the usernames in your system, just by sending email messages

IP filtering will not help

SPAM filters will not help

SMTP mail, a special case

SMTP always responds with recipient known or unknown, thus allowing bad guys to discover all usernames in the system

SMTP operates as a relay scheme. We cannot just block an IP number because it is for the last relay in the chain, not the originator of traffic.

The solution is to accept all email, and internally redirect that for unknown recipients to a bin

Postfix email solution

1. File */etc/postfix/main.cf*, at the end, add these two lines

```
luser_relay = notauser
local_recipient_maps =
```

2. Create user *notauser* with no login shell etc

Line within file */etc/passwd*:

```
notauser:x:1001:65534:Not A. User, Dummy user to black hole email to non-
   existant users:/var/lib/empty:/bin/false
```

3. Look in */var/spool/mail* to see the rubbish bin

```
-rw------- 1 notauser  nogroup  8387 Jun 5 02:38 notauser
```

Head of notauser email file above:

Return-Path: <precolorings@itrco.com>
X-Original-To: 74628542@netlab1.oucs.ox.ac.uk
Delivered-To: notauser@netlab1.oucs.ox.ac.uk

Motivations for this work were

- Normal requirements to protect many services, yet have rules be readily verifiable (to my eyes) and be in one place (no surprises)
- Annoyance with zealots requiring huge obscure passwords, changed frequently

Now it is your turn

MindWorks UK

MindWorks Inc. Ltd
210 Burnley Road
Weir
Bacup
OL13 8QE UK

Telephone: +44 (0) 170 687 1900

Fax: +44 (0) 170 687 8203

Web: www.mindworksuk.com

Email: training@mindworksuk.com