Fail2ban

STLLUG
17 April 2014

Fail2ban blocks repetitive unauthorized access attempts by adding timed drop rules to the iptables INPUT chain for IP addresses found in logfile messages.

A slight simplification, but it conveys the essentials. The details will be covered in a bit.

Fail2ban searches specified logfiles for messages recording failed access attempts.

The IP addresses from those messages are banned if they appear ‘too often’.

Drop rules are added to the iptables chains matching the intruder’s IP address.

Fail2ban removes those drop rules after the corresponding timer expires.

About Me

I work as an independent consultant performing system and small network administration, and writing specialized technical documentation.

I use fail2ban on a client's mail server (exim4, UW IMAP/POP3, Debian stable) to drop about 100 hostile packets/hour.

XML or SGML based mil-spec documents

Run the Visual C++ debugger or read a switchbox schematic.

Generate 'picture books' for custom test program sets using Python and MS Word.

Questions to be Answered

- Why would you want it?
- What do you need to know to make it work?
- How do you get it?
- How can you learn more?

What is it? – answered two slides back.

Why would you want it?

- Reduce noise in log files
- Impede unauthorized access
- Reduce spam filtering load
- Reduce webserver load
- Easy to get, setup, and use

Because you want to accomplish these things

Because it has these desirable features

What do you need to know to make it work?

- Jails
- Filters
- Actions
- ~~iptables~~
- Examples and results
- Bonus python script

A jail defines where and how to look for repetitive access failures and what to do about them.

A filter is the regular expression (or > one) that matches or rejects log file messages to identify unauthorized access attempts. Most work for me.

The default action when a match occurs is to add a drop rule to the iptables chains. You can define your own actions. I have not.

Iptables is a powerful package from netfilter.org that lets you specify match rules and actions on network packets. Iptables is probably installed by default. Fail2ban uses iptables to drop 'hostile' packets.

Jails

/etc/fail2ban/jail.local

```
[Default]
ignoreip = 127.0.0.1/8 10.1.1.0/24
[eximt]
enabled   = true
port      = smtp,ssmtp,submission
filter    = target-exim
```


A jail defines where and how to look for repetitive access failures and what to do about them. A filter, an action, and some data to make them work.

(Don't confuse this with a chroot or FreeBSD jail)

The [Default] section specifies global values

Enabled – the obvious

Port – these ports are used in the default banaction

Filter – the file with the regular expression(s)

Jails

```
banaction = iptables-multiport[name=EXIMT,
port="smtp,ssmtp,submission", protocol=tcp]
logpath   = /var/log/exim4/rejectlog
maxretry  = 1
findtime  = 3600
bantime   = 14400
```


Banaction – do this when an attempt is found

Logpath – the log file to look in

Maxretry – how many matches for an IP address trigger the banaction

Findtime – the window for counting maxretry

Bantime – how long that IP address is dropped; how long before fail2ban removes the the drop rule from iptables (assuming default actions)

Filters

/etc/fail2ban/filter.d

Fail2ban strips the date from the logfile entry before applying regular expressions.

Test your filter files with fail2ban-regex.

A filter is the regular expression (or more than one) that matches or rejects log file messages to identify unauthorized access attempts.

/etc/fail2ban/filter.d

The fail2ban regular expressions work on the logfile entry after the date has been stripped off

You can test your filter files with fail2ban-regex

I use regular expressions often enough to create this one, but rarely enough that I keep a copy of 'Mastering Regular Expressions' handy. 2Nd edition is $10 w/ shipping, used.

Filters

failregex = ^ login_server authenticator failed
for .* ?\((ylmf\-pc|User|\[192\.168\.2\.33\])\)
\[<HOST>\]: 535 Incorrect authentication data
\(set_id=.*\)$

The fail2ban regular expressions work on the logfile entry after the date has been stripped off

You can test your filter files with fail2ban-regex

I use regular expressions often enough to create this one, but rarely enough that I keep a copy of 'Mastering Regular Expressions' handy. 2Nd edition is $10 w/ shipping, used.

This is a filter I created when I observed that most targeted (using real usernames) attempts had some common features.

Actions

- The default actions ban an ipaddr/port(s) combination for a limited time.
- You can define custom actions.

The default action when a match occurs is to add a drop rule to the iptables chains. You can define your own actions.

Iptables is a powerful package from netfilter.org that lets you specify match rules and actions on network packets. Iptables is probably installed by default. Fail2ban uses iptables to drop 'hostile' packets.

iptables

- Match rules and actions on network packets
- Probably already installed
- Fail2ban knows how to use it!
- “iptables -vnxL” displays rules, chains and counters

Iptables is a powerful package from netfilter.org that lets you specify match rules and actions on network packets. Iptables is probably installed by default. Fail2ban uses iptables to drop 'hostile' packets.

Examples – Standard Filters

- [exim4]
- [ssh]
- [uw-imap]

I used these filters mostly 'out of the box', though it turned out I needed to modify the exim4 filter to match the log message format on the system where I use fail2ban.

Examples – Custom Filters

- [eximt]
- [flag]
- [spam]
- [ssh-root]

These are custom filters I set up.

Eximt limits many targeted attack

Flag filters repetitive senders of flagged email.

Spam filters senders of spam

Ssh-root filters any attempt to use an account on the deny list.

Results (uptime 30 days)

Linux2:/etc/cron.daily# iptables -vnxL | /etc/cron.daily/f2bcounts.py

- ('Filter/Jail', 'Pkts', 'Bytes')
- ('fail2ban-FLAG', 2016, 192078L)
- ('fail2ban-SPAM', 33849, 1789535L)
- ('fail2ban-EXIM4', 9706, 464767L)
- ('fail2ban-UW-IMAP', 21264, 1055070)
- ('fail2ban-EXIMT', 1120, 61914L)
- ('fail2ban-ssh-root', 1448, 127360L)
- ('fail2ban-ssh', 0, 0L)
- ('TOTALS', 69403, 3690724L)

Bonus Python Script

(In the notes, in the PDF)


```
# copyright Ken Johnson, 2014
# This is free software; you can redistribute it and/or modify it under
# the terms of version 2 of the GNU General Public License as
# published by the Free Software Foundation.

# This software is distributed in the hope that it will be useful, but
# WITHOUT ANY WARRANTY;without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

# The INPUT chain is the one we care about.  Packets work down the
# chain until accepted or dropped.  The fail2ban rules are jump rules,
# that is, matching packets jump to the named chain.  If they match
# a rule in that named chain, they are dropped.  Otherwise, they return
# to the next rule in the INPUT chain.  So, to find out how many packets
# are dropped by a f2b rule, subtract the returned packet count of the
# f2b chain from the match packet count shown for that f2b rule in the
# INPUT chain.

import fileinput

seenChainINPUT = False
seenChainFORWARD = False
getReturnCount = False
chain = ""
beforeList = []
afterList = []
totPkts = 0
totBytes = 0
```

Bonus Python Script

(In the notes, in the PDF)


```
for line in fileinput.input():
  if not seenChainINPUT:
    if "Chain INPUT" in line:
      seenChainINPUT = True

  elif not seenChainFORWARD:
    if "Chain FORWARD" in line:
      seenChainFORWARD = True

    elif "fail2ban-" in line:
      slist = line.split()
      beforeList.append((slist[2], slist[0], slist[1]))

  elif "Chain fail2ban-" in line:
    slist = line.split()
    chain = slist[1]

  elif "RETURN" in line:
    slist = line.split()
    afterList.append((chain, slist[0], slist[1]))

for bsl in beforeList:
  for asl in afterList:
    if bsl[0] == asl[0]:
      print (bsl[0], int(bsl[1])-int(asl[1]), int(bsl[2])-int(asl[2]))
      totPkts = totPkts + int(bsl[1])-int(asl[1])
      totBytes = totBytes + int(bsl[2])-int(asl[2])
print ("TOTALS", totPkts, totBytes)
```

Other resources

- www.fail2ban.org – wiki
- sourceforge.net/p/fail2ban/mailman/fail2ban-users/
- https://github.com/fail2ban/fail2ban/issues
- http://www.onerussian.com/tmp/fail2ban-pycon2014.pdf

Wiki

Mailing list

Issue tracker

Presentation at Pycon