Fail2Ban Setup

Introduction

When operating web and email servers, it is important to implement security measures to protect your site and users. Any publicly accessible password prompt is likely to attract brute force attempts from malicious users and bots. Setting up fail2ban can help alleviate this problem. When users repeatedly fail to authenticate to a service (or engage in other suspicious activity), fail2ban can issue a temporary bans on the offending IP address by dynamically modifying the running firewall policy. Each fail2ban “jail” operates by checking the logs written by a service for patterns which indicate failed attempts. Setting up fail2ban to monitor Nginx logs is fairly easy using the some of included configuration filters and some we will create ourselves.

Install

Note: we are installing fail2ban in the individual LXC containers and not on the host. Because fail2ban need IP-tables working to exclude attackers and IP-tables seem to work on LXC container level. Install fail2ban into the Mail and Web Server containers:

```
apt-get update
apt-get install fail2ban
```

This will install the software. It starts automatically after installation. Also nftables will be installed. To check the services status use this command:

```
systemctl status fail2ban
systemctl status nftables
```

Configuration

The fail2ban.conf file configures some basic operational settings like the way the daemon logs info, and the socket and pid file it will use. The supplied /etc/fail2ban/jail.conf file is the main provided resource for this. The remaining configuration, however takes place in the files that define the “jails”.

/etc/fail2ban/fail2ban.local

To make modifications, we need to copy this file to /etc/fail2ban/fail2ban.local. This will prevent our changes from being overwritten if a package update provides a new default file:

```
sudo cp /etc/fail2ban/fail2ban.conf /etc/fail2ban/fail2ban.local
```

Common options to change in the default fail2ban.local file are:

1. Logtarget: we prefer Fail2ban to write logs to: /var/log/fail2ban.log. Change the line: logtarget = SYSLOG to l**ogtarget = /var/log/fail2ban.log**
2. To avoid startup warnings on ipv6 we added option: **allowipv6 = auto**

```
[DEFAULT]

# OdG: added 03/09/2023 to avoid startup warning on ipv6
allowipv6 = auto

# Option: loglevel
# Notes.: Set the log level output.
#         CRITICAL
#         ERROR
#         WARNING
#         NOTICE
#         INFO
#         DEBUG
# Values: [ LEVEL ]  Default: INFO
#
loglevel = WARNING

# Option: logtarget
# Notes.: Set the log target. This could be a file, SYSTEMD-JOURNAL, SYSLOG,
STDERR or STDOUT.
#         Only one log target can be specified.
#         If you change logtarget from the default value and you are
#         using logrotate -- also adjust or disable rotation in the
#         corresponding configuration file
#         (e.g. /etc/logrotate.d/fail2ban on Debian systems)
# Values: [ STDOUT | STDERR | SYSLOG | SYSOUT | SYSTEMD-JOURNAL | FILE ]
Default: STDERR
#
logtarget = /var/log/fail2ban.log

# Option: syslogsocket
# Notes: Set the syslog socket file. Only used when logtarget is SYSLOG
#        auto uses platform.system() to determine predefined paths
# Values: [ auto | FILE ]  Default: auto
syslogsocket = auto

# Option: socket
# Notes.: Set the socket file. This is used to communicate with the daemon.
Do
#         not remove this file when Fail2ban runs. It will not be possible
to
#         communicate with the server afterwards.
# Values: [ FILE ]  Default: /var/run/fail2ban/fail2ban.sock
#
socket = /var/run/fail2ban/fail2ban.sock

# Option: pidfile
# Notes.: Set the PID file. This is used to store the process ID of the
#         fail2ban server.
# Values: [ FILE ]  Default: /var/run/fail2ban/fail2ban.pid
#
pidfile = /var/run/fail2ban/fail2ban.pid

# Option: allowipv6
# Notes.: Allows IPv6 interface:
#         Default: auto
# Values: [ auto yes (on, true, 1) no (off, false, 0) ] Default: auto
#allowipv6 = auto

# Options: dbfile
# Notes.: Set the file for the fail2ban persistent data to be stored.
#         A value of ":memory:" means database is only stored in memory
#         and data is lost when fail2ban is stopped.
#         A value of "None" disables the database.
# Values: [ None :memory: FILE ] Default: /var/lib/fail2ban/fail2ban.sqlite3
dbfile = /var/lib/fail2ban/fail2ban.sqlite3

# Options: dbpurgeage
# Notes.: Sets age at which bans should be purged from the database
# Values: [ SECONDS ] Default: 86400 (24hours)
dbpurgeage = 1d

# Options: dbmaxmatches
# Notes.: Number of matches stored in database per ticket (resolvable via
#         tags <ipmatches>/<ipjailmatches> in actions)
# Values: [ INT ] Default: 10
dbmaxmatches = 10

[Definition]

[Thread]

# Options: stacksize
# Notes.: Specifies the stack size (in KiB) to be used for subsequently
created threads,
#         and must be 0 or a positive integer value of at least 32.
# Values: [ SIZE ] Default: 0 (use platform or configured default)
#stacksize = 0
```

Jails

The jail config file also specifies the available jails that will monitor our application logs for specific behavior patterns.

/etc/fail2ban/jail.local

The supplied **/etc/fail2ban/jail.conf** file is the main provided resource for this. To make modifications, we need to copy this file to **/etc/fail2ban/jail.local.** This will prevent our changes from being overwritten if a package update provides a new default file:

```
  # cp /etc/fail2ban/jail.conf /etc/fail2ban/jail.local
  # nano /etc/fail2ban/jail.local
```

Each jail within the configuration file is marked by a header containing the jail name in square brackets (every section but the [DEFAULT] section indicates a specific jail's configuration). By default, only the **[ssh]** jail is enabled. We should start by evaluating the defaults set within the file to see if they suit our needs. These will be found under the **[DEFAULT]** section within the file. These items set the general policy and can each be overridden in specific jails. Look at the following directives:

- **ignoreip:**It is a good idea to add your own IP address or network to the list of exceptions to avoid locking yourself out. This is less of an issue with web server logins though if you are able to maintain shell access, since you can always manually reverse the ban. You can add additional IP addresses or networks delimited by a space, to the existing list.
- **bantime:** controls how many seconds an offending member is banned for. It is ideal to set this to a long enough time to be disruptive to a malicious actor's efforts, while short enough to allow legitimate users to rectify mistakes. By default, this is set to (10 minutes). Increase or decrease this value as you see fit.
- **findtime:** specifies an amount of time in seconds that maxretry attemps are allowed before blocking
- **maxretry:** directive indicates the number of attempts to be tolerated within that time. If a client makes more than maxretry attempts within the amount of time set by findtime, they will be banned.
- **backend:** This entry specifies how fail2ban will monitor log files. The setting of auto means that fail2ban will try pyinotify, then gamin, and then a polling algorithm based on what's available.
- **usedns:** This defines whether reverse DNS is used to help implement bans. Setting this to "no" will ban IPs themselves instead of hostnames. The "warn" setting will attempt to use reverse-DNS to look up the hostname and ban that way, but will log the activity for review.
- **destemail:** This is the address that will be sent notification mail if configured your action to mail alerts.
- **sendername:** This will be used in the email from field for generated notification emails
- **banaction:** This sets the action that will be used when the threshold is reached. There is actually the name of a file located in /etc/fail2ban/action.d/ called iptables-multiport.conf. This handles the actual iptables manipulation to ban an IP address. We will look at this later.
- **mta:** This is the mail transfer agent that will be used to send notification emails.
- **protocol:** This is the type of traffic that will be dropped when an IP ban is implemented. This is also the type of traffic that is sent to the new iptables chain.
- **chain:** This is the chain that will be configured with a jump rule to send traffic to the fail2ban funnel.

```
[DEFAULT]
. . .
ignoreip = 127.0.0.1/8 your_home_IP
ignoreip = 127.0.0.1/8 54.15.55.01
bantime  = 3600
```

```
findtime = 3600
maxretry = 6
. . .
banaction = nftables-multiport
banaction_allports = nftables-allports
. . .
```

Any of these parameters in the **[DEFAULT]** section can be overruled in the individual [JAIL] sections.

Debian default Jails

On Debian only **ssh** is enabled via a file in: **/etc/fail2ban/jail.d/defaults-debian.conf**. Since we prefer to maintain our jails in jail.local. Remove this file.

```
# cat /etc/fail2ban/jail.d/defaults-debian.conf/defaults-debian.conf
------------------------------------------------------------------
[sshd]
enabled = true

# rm /etc/fail2ban/jail.d/defaults-debian.conf/defaults-debian.conf
```

Systemd Journal

On modern systemd-based distros, like newer releases of Debian services like sshd logs to the systemd journal. There is no logfile anymore in /var/log. This can cause jail2ban.service to terminate with an error: *“ERROR Failed during configuration: Have not found any log file for sshd jail”*. To solve this tell fail2ban that it should use systemd logging (backend = systemd):

```
# nano /etc/fail2ban/jail.local
------------------------------
...
[sshd]
port    = ssh
logpath = %(sshd_log)s
#backend = %(sshd_backend)s
backend = systemd
...
```

Filters

The jails section in */etc/fail2ban/jail.local* file enumerates the available jails. The names between brackets ***[Jail-name]*** refer to a filter file in the **/etc/fail2ban/filter.d** directory. Files in this directory have the ***.conf** extension and contain **failregex** specifications that are used to scan the logs.

www-login-fail.conf

We have developed one custom filter file to monitor failed login attempts in our website.

```
# nano /etc/fail2ban/filter.d/www-login-fail.conf
-------------------------------------------------
# Fail2Ban filter to catch:
#   1)  Unknown user login attempts
#   2)  Incorrect password attempts www.oscardegroot.nl
#
[INCLUDES]
# Load regexes for filtering
before =

[Definition]

failregex =
^.+Login\sattempt\suser.+incorrect\spassword.+client:\s<HOST>.+,\sserver.+$
            ^.+Login\sattempt\sip\[<HOST>\].+$

ignoreregex =

datepattern =
```

fail2ban-regex

The correctness of filter files with regex expressions can be tested with:

- fail2ban-regex
- fail2ban-regex inline

fail2ban-regex

The fail2ban-regex command line tool offers a way of testing if a regex is working as expected. You can use it as follows:

```
fail2ban-regex <logpath> <filterpath>
```

Here is a sample of it in use.

```
fail2ban-regex /var/lib/docker/containers/7c2442*/*-json.log
/etc/fail2ban/filter.d/nginx-noscript.conf
```

And a sample output report is shown below.

```
Running tests
=============
Use   failregex filter file : nginx-noscript, basedir: /etc/fail2ban
Use         log file : /var/lib/docker/containers/94dc5*/94dc5*-json.log
Use         encoding : UTF-8

Results
=======
Failregex: 233 total
|-  #) [# of hits] regular expression
|   1) [233] ^{"log":"<HOST> -.*GET.*(\.php|\.asp|\.exe|\.pl|\.cgi|\.scgi)
`-

Ignoreregex: 0 total

Date template hits:
|- [# of hits] date format
|  [724] Day(?P<_sep>[-/])MON(?P=_sep)ExYear[
:]?24hour:Minute:Second(?:\.Microseconds)?(?: Zone offset)?
`-

Lines: 724 lines, 0 ignored, 233 matched, 491 missed
[processed in 0.10 sec]
```

Notice towards the end of the report, it states 233 matched, 491 missed. This means that the filter was able to match 233 lines and 491 were not applicable. This is a good indication that the regex is matching and working.

fail2ban-regex inline

Below is the syntax of how to use it.

```
fail2ban-regex '<logline>' '<regex>'
```

And below is a real life example.

```
fail2ban-regex '{"log":"111.22.333.444 47.95.1.195 - - [05/Jul/2018:21:42:40
+0000] \"GET /phpMyadmin_bak/index.php HTTP/1.1\" 503 213 \"-\"
\"Mozilla/5.0\"\n","stream":"stdout","time":"2018-07-05T21:42:
40.24318153Z"}' '{"log":"<HOST>.*GET.*.php.*'
```

The sample below specifies a single log file entry:

'{“log”:“111.22.333.444 47.95.1.195 - - [05/Jul/2018:21:42:40 +0000] \”GET /phpMyadmin_bak/index.php HTTP/1.1\“ 503 213 \”-\“ \”Mozilla/5.0\“\n”,“stream”:“stdout”,“time”:“2018-07-05T21:42:40.24318153Z”}'

and the regex {“log”:“<HOST>.*GET.*.php.*' is used to verify if it works.

Below is the sample report:

```
Running tests
```

```
=============
Use   failregex line : {"log":"<HOST>.*GET.*.php.*
Use      single line : {"log":"111.22.333.444 47.95.1.195 - - [05/Jul/201...

Results
=======
Failregex: 1 total
|-  #) [# of hits] regular expression
|   1) [1] {"log":"<HOST>.*GET.*.php.*
`-

Ignoreregex: 0 total

Date template hits:
|- [# of hits] date format
|  [1] Day(?P<_sep>[-/])MON(?P=_sep)ExYear[
:]?24hour:Minute:Second(?:\.Microseconds)?(?: Zone offset)?
`-

Lines: 1 lines, 0 ignored, 1 matched, 0 missed
[processed in 0.03 sec]
```

Actions

This file is responsible for setting up the firewall with a structure that allows easy modifications for banning malicious hosts, and for adding and removing those hosts as necessary. It is located in the following directory: **/etc/fail2ban/action.d**. E.g. the action that our SSH service invokes is called iptables-multiport. Open the associated file now:

```
sudo nano /etc/fail2ban/action.d/iptables-multiport.conf
```

With the comments removed, this file looks something like this:

```
[INCLUDES]
before = iptables-blocktype.conf

[Definition]
actionstart = iptables -N fail2ban-<name>
              iptables -A fail2ban-<name> -j RETURN
              iptables -I <chain> -p <protocol> -m multiport --dports <port>
-j fail2ban-<name>

actionstop = iptables -D <chain> -p <protocol> -m multiport --dports <port>
-j fail2ban-<name>

actioncheck = iptables -n -L <chain> | grep -a 'fail2ban-<name>[ \t]'

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

```

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

[Init]
name = default
port = ssh
protocol = tcp
chain = INPUT
```

Ban IP Range (subnet)

In some cases spammers have several systems infected in a specific subnet. By using different alternating IP nummers in this subnet, they can avoid Fail2Ban detection. In these cases it could be helpful to block a whole subnet range at once. We created an adapted action file for **nftables-subnet.conf** that was derived from *nftables.conf*.

```

# cp /etc/fail2ban/action.d/nftables-subnet.conf
/etc/fail2ban/action.d/nftables-subnet.conf
# nano /etc/fail2ban/action.d/nftables-subnet.conf
----------------------------------------------------------------------------
----------------
The following changes have been applied
<code>
. . .
Add the "flags interval" needed by nftables to allow specifying a subnet
mask
   _nft_add_set = <nftables> add set <table_family> <table> <addr_set> \{
type <addr_type>\; \}
into
   _nft_add_set = <nftables> add set <table_family> <table> <addr_set> \{
type <addr_type>\;flags interval\;\}
. . .
Add the <addr_subnet> in the line that adds records/elements in the nft set
   actionban = <nftables> add element <table_family> <table> <addr_set> \{
<ip> \}
into
   actionban = <nftables> add element <table_family> <table> <addr_set> \{
<ip>\/<addr_subnet> \}
. . .
At the bottom of the file add the following subnet specifiers to the [INIT]
and [Init?family=inet6]:
[INIT] add:
   addr_subnet = 24
[Init?family=inet6]
   addr_subnet = 56
```

Now we need to add this custom action to our jail in: **/etc/fail2ban/jail.local**. Add the following line to the specific jail: **banaction = nftables-subnet[type=multiport]**. For example:

```
[www-login-fail]
```

```
enabled = true
port = http,https
logpath = /var/log/nginx/error.www.log
findtime  = 900
maxretry = 3
banaction = nftables-subnet[type=multiport]
```

Restart & activating Changes

Any changes made in jail.local a restart of Fail2ban is required. Below are commands which will restart the service.

```
  systemctl restart fail2ban
  or
  fail2ban-client reload
```

If there is anything wrong with the fail or filter, an error would be reported.

Monitor Status

There are 2 options to check the current status of jails and banned clients:

1. fail2ban-client
2. nft list ruleset

fail2ban-client

You can see an overview of your enabled jails by using the fail2ban-client command. You should see a list of all of the jails you enabled:

```
# fail2ban-client status
------------------------
Status
|- Number of jail:  5
`- Jail list:   nextcloud, nginx-bad-request, nginx-botsearch, sshd, www-login-fail
```

If you want to see the details of the bans being enforced by any one jail, it is probably easier to use the fail2ban-client again:

```
# fail2ban-client status www-login-fail
--------------------------------------
Status for the jail: www-login-fail
|- Filter
|  |- Currently failed: 0
|  |- Total failed: 3
|  `- File list:    /var/log/nginx/error.www.log
`- Actions
   |- Currently banned: 0
   |- Total banned: 2
   `- Banned IP list:
```

NFtables

You can look at nftables to see that fail2ban has modified your firewall rules to create a framework for banning clients. Even with no previous firewall rules, you would now have a framework enabled that allows fail2ban to selectively ban clients by adding them to purpose-built chains:

```
#nft list ruleset
-----------------
table inet f2b-table {
    set addr-set-www-login-fail {
        type ipv4_addr
        flags interval
        elements = { 192.168.178.0/24 }
    }

    set addr6-set-www-login-fail {
        type ipv6_addr
        flags interval
        elements = { fdaa:66:67::/56 }
    }

    chain f2b-chain {
        type filter hook input priority filter - 1; policy accept;
        tcp dport { 80, 443 } ip saddr @addr-set-www-login-fail reject with
icmp port-unreachable
        tcp dport { 80, 443 } ip6 saddr @addr6-set-www-login-fail reject
with icmpv6 port-unreachable
    }
}
```

Unbanning

you can manually un-ban your IP address with the fail2ban-client by typing:

```
# fail2ban-client set nginx-http-auth unbanip 111.111.111.111
```

Or for unbanning all jails at once:

```
# fail2ban-client unban --all
```

Cleaning

The above unbanning commands will remove the IP's from the nftables firewall, but the rule and table structures will remain in the nfttables firewall. If you want to clean / reset everything:

```
# fail2ban-client stop www-login-fail
```

The following command will only clear the nftables structures, but this could lead to errors in jail2bin logs.

```
# nft clear ruleset
```

Optional: Setting Up Mail Notifications

You can enable email notifications if you wish to receive mail whenever a ban takes place. To do so, you will have to first set up an MTA on your server so that it can send out email. To learn how to use Postfix for this task, follow this guide. Once you have your MTA set up, you will have to adjust some additional settings within the **[DEFAULT]** section of the **/etc/fail2ban/jail.local** file. Start by setting the **mta** directive. If you set up Postfix, like the above tutorial demonstrates, change this value to “mail”. You need to select the email address that will be sent notifications. Modify the **destemail** directive with this value. The **sendername** directive can be used to modify the “Sender” field in the notification emails. A fail2ban “action” is the procedure followed when a client fails authentication too many times. The default action (called **action_**) is to simply ban the IP address from the port in question. However, there are two other pre-made actions that can be used if you have mail set up. You can use the **action_mw** action to ban the client and send an email notification to your configured account with a “whois” report on the offending address. You could also use the **action_mwl** action, which does the same thing, but also includes the offending log lines that triggered the ban:

```
[DEFAULT]
. . .
mta = mail
. . .
destemail = youraccount@email.com
sendername = Fail2BanAlerts
. . .
action = %(action_mwl)s
. . .
```

NGINX

We enables the following 2 default Debian fail2ban installation jails for Nginx:

- **nginx-bad-request.conf**: filter to match bad requests to nginx
- **nginx-botsearch.conf**: filter to match web requests for selected URLs that don't exist

The following have been added by ourselves:

- **www-login-fail.conf**: filter to detect failed login attempts on our website.

The following default is not used, because we have no basic auth enables in Nginx

- **nginx-http-auth.conf**: filter for http basic authentication failures
- **nginx-limit-req.conf**: filter to ban hosts, that fail through nginx by limit request processing rate

/etc/fail2ban/jail.local

```
#
# HTTP servers
#
[nginx-botsearch]
enabled = true
port     = http,https
logpath = /var/log/nginx/access.*.log
findtime  = 900
maxretry = 3
banaction = nftables-subnet[type=multiport]

[nginx-bad-request]
enabled = true
port    = http,https
logpath = /var/log/nginx/access.*.log
findtime  = 900
maxretry = 3
banaction = nftables-subnet[type=multiport]

[www-login-fail]
enabled = true
port = http,https
logpath = /var/log/nginx/error.www.log
findtime  = 900
maxretry = 3
banaction = nftables-subnet[type=multiport]

# To use more aggressive http-auth modes set filter parameter "mode" in
jail.local:
# normal (default), aggressive (combines all), auth or fallback
# See "tests/files/logs/nginx-http-auth" or "filter.d/nginx-http-auth.conf"
for usage example and details.
[nginx-http-auth]
# mode = normal
port    = http,https
logpath = %(nginx_error_log)s

# To use 'nginx-limit-req' jail you should have `ngx_http_limit_req_module`
# and define `limit_req` and `limit_req_zone` as described in nginx
documentation
```

```
# http://nginx.org/en/docs/http/ngx_http_limit_req_module.html
# or for example see in 'config/filter.d/nginx-limit-req.conf'
[nginx-limit-req]
port    = http,https
logpath = %(nginx_error_log)s
```

/etc/fail2ban/filter.d/nginx-bad-request.conf

```
[Definition]
# The request often doesn't contain a method, only some encoded garbage
# This will also match requests that are entirely empty
failregex = ^<HOST> - \S+ \[\] "[^"]*" 400

datepattern = {^LN-BEG}%%ExY(?P<_sep>[-/.])%%m(?P=_sep)%%d[T
]%%H:%%M:%%S(?:[.,]%%f)?(?:\s*%%z)?
              ^[^\[]*\[({DATE})
              {^LN-BEG}

journalmatch = _SYSTEMD_UNIT=nginx.service + _COMM=nginx
```

/etc/fail2ban/filter.d/nginx-botsearch.conf

```
INCLUDES]
# Load regexes for filtering
before = botsearch-common.conf

[Definition]
failregex = ^<HOST> \- \S+ \[\] \"(GET|POST|HEAD) \/<block> \S+\" 404 .+$
           ^ \[error\] \d+#\d+: \*\d+ (\S+ )?\"\S+\" (failed|is not found)
\(2\: No such file or directory\), client\: <HOST>\, server\: \S*\, request:
\"(GET|POST|HEAD) \/<block> \S+\"\, .*?$

ignoreregex =

datepattern = {^LN-BEG}%%ExY(?P<_sep>[-/.])%%m(?P=_sep)%%d[T
]%%H:%%M:%%S(?:[.,]%%f)?(?:\s*%%z)?
              ^[^\[]*\[({DATE})
              {^LN-BEG}

journalmatch = _SYSTEMD_UNIT=nginx.service + _COMM=nginx
```

</code>

/etc/fail2ban/filter.d/www-login-fail.conf

```
[INCLUDES]
# Load regexes for filtering
```

```
before =

[Definition]

failregex =
^.+Login\sattempt\suser.+incorrect\spassword.+client:\s<HOST>.+,\sserver.+$
            ^.+Login\sattempt\sip\[<HOST>\].+$

ignoreregex =

datepattern =
```

Links

- How Fail2Ban Works
- How To Protect an Nginx Server with Fail2Ban

From:
https://wiki.oscardegroot.nl/ - **HomeWiki**

Permanent link:
https://wiki.oscardegroot.nl/doku.php?id=linux:apps:fail2ban&rev=1693940762

Last update: **2023/09/05 19:06**