Fail2Ban Demystified

A Beginner's Guide to Configuring Fail2Ban

Custom Fail2Ban Actions

Fail2Ban is a powerful security tool that bridges your public-facing network services (ssh, mail server, web server, *etc.*) and your firewall. The Fail2Ban daemon constantly monitors your server logs, looking for troublemakers. When an intruder or suspicious activity is detected, it takes action! By default, Fail2Ban writes a temporary firewall rule that blocks the offender's IP address on selected ports for a short period of time.

The real power in Fail2Ban, though, lies in its customizability. Using custom action scripts, admins can configure how Fail2Ban responds to threats, extending its abilities far beyond simply interfacing with the firewall. For example, you can define actions that will send an email or notification, write to a log file or database, invoke a web API, react differently in different contexts, and so on. The possibilities are unlimited!

This article explains how actions in Fail2Ban are defined and customized so that you can get the most out of Fail2Ban.

Fail2Ban Conventions

If you're not yet experienced with Fail2Ban configuration, it will be helpful for you to be familiar with these terms:

filter

A *filter* is a set of customized regular expressions and properties that is used to match threatening log entries. When a filter is matched, the actions in the corresponding *jail* are invoked. Filter definitions are normally found in the *filter.d* subdirectory. This article touches only briefly on filters, but a companion article will describe them in more detail.

action

An *action* is a shell command (or commands) that Fail2Ban executes at appropriate times. For example a *ban action* is executed when Fail2Ban invokes a ban against an IP address. Actions are normally defined by configuration files in the *action.d* subdirectory.

jail

A *jail* is a set of actions and parameters that Fail2Ban applies when its *filter* is matched. A jail can have multiple actions of the same type, but can only have one filter. Jails are normally defined in *jail.conf* and *jail.local* in the configuration directory, and by files in the *jail.d* subdirectory.

Action Configuration Files

This article assumes that you already have Fail2Ban installed and configured correctly to work with your firewall and now you want to start customizing what it does. Action configuration files are found in the *action.d* subdirectory of your Fail2Ban configuration directory. By default, this is */etc/fail2ban/action.d*. In this directory, you'll find a few default templates for sending email alerts, alerting AbuseIPDB.com or Blocklist.de, writing to hosts.deny, and a few other actions, but without further configuration, Fail2Ban only invokes the actions for writing (and unwriting) firewall rules.

By convention, all Fail2Ban configuration files end in *.conf* (for default templates provided with the Fail2Ban package) or *.local* (for user-defined configurations). When a configuration is parsed by Fail2Ban, it will first load the *.conf* file, and then load the corresponding *.local* file, updating any parameters that differ between the two files.

Important: Since *.conf* files may be overwritten by new versions when Fail2Ban is updated, you should not make any changes directly to these files. Instead, make a copy with the *.local* suffix and edit the copy:

```
$ sudo cp mail-whois.conf mail-whois.local
```

If you are only adding or changing a small number of parameters, it may be more efficient to just create a new *.local* file with those parameters:

```
$ echo dest=flathead@example.com | sudo tee mail-whois.local
```

Sections

Action configuration files can have up to three sections, identified by the name of the section in square brackets.

[INCLUDES]

Configuration files listed in this optional section will be parsed either before or after the current configuration file. This is useful for importing or updating common variables. Separate multiple files with spaces or include them on separate lines beginning with a space or tab.

```
before = file(s)
```

Parse these files *before* the current configuration file.

```
after = file(s)
```

Parse these files *after* the current configuration file, updating variables with new values.

In the following example, the parameters in *mail-addresses.local* and *mail-options.local* will be read, then those parameters will be updated and amended by the parameters in this file, then finally *mail-overrides.local* will be parsed.

```
[INCLUDES]

before =  mail-addresses.local
          mail-options.local

after =   mail-overrides.local
```

[Definition]

This section defines the various actions that Fail2Ban will take, for example, when banning or unbanning an IP address. See the section *Fail2Ban Actions* below for details.

Note that you can specify that the actions for banning and unbanning should be ignored for restored bans (for example, bans that are restored when Fail2Ban is restarted) by including:

```
  norestored = 1
```

[Init]

This optional section defines the default values for variables used by the actions in this configuration. The default values will be overridden when either:

- A new value is assigned in a configuration file parsed *after* this one in the [INCLUDES] section (see above), or
- A value is explicitly assigned in the *jail* definition.

For example,

```
[INCLUDES]
after =   mail-overrides.local

[Definition]
norestored = 1
actionban =    writelog.sh <message>

[Init]
message = An IP address has been banned
```

In this example, a shell script called “writelog.sh” is executed with the argument *message* when a scofflaw is banned. (Note the angle brackets around the variable name.) The default value for *message* is “An IP address has been banned” but if *message* is redefined by *mail-overrides.local*, or by the jail’s definition, the updated value will be used instead.

Fail2Ban Actions

These are the actions that can be included in the [Definition] section of an action configuration file. Each specifies a command or script to execute in a certain context. By default, the Fail2Ban daemon runs as root, but if you have changed that behavior, you’ll need to make sure the daemon’s user has permissions for the desired commands.

Action scripts can be multiple lines so long as each new line begins with a space or tab, similar to shell scripts.

Most custom actions will only need an *actionban* command or script. The list of valid actions includes:

actionban

As the name implies, Fail2Ban will try to run this command or script when it normally bans an IP address. If the command fails (returns an exit code other than 0), *actioncheck* will be run next, otherwise the ban is considered enforced. You can instruct Fail2Ban to ignore this action for restored bans by setting *norestored=1* in the [Definition] section of the file.

actionunban

This defines the command or script that Fail2Ban will run when an IP address is unbanned, either because it has reached its expiration time or an unban has been forced, such as when the service is stopped. If the command fails (returns an exit code other than 0), *actioncheck* will run next. You can instruct Fail2Ban to ignore this action for restored bans by setting *norestored=1* in the [Definition] section of the file.

actionprolong

This defines the command that Fail2Ban will run when it extends an IP's ban time due to repeated offenses. This occurs when (1) the jail's *bantime.increment* property is "true" and (2) there is at least one previous ban for this IP in the database. This action is run in addition to and after *actionban* has completed, which may be a few seconds later.

actioncheck

This command or script is invoked only if *actionban* or *actionunban* fails (returns an exit code other than 0). If *actioncheck* also returns an exit code other than 0, then *actionrepair* will be called next. For example, if *actionban* tries to write to a file but fails, *actioncheck* could verify the existence and permissions of the file, then, if it doesn't exist, *actionrepair* could recreate the file and set permissions. Note that in early versions of Fail2Ban, *actioncheck* was called before each *actionban*, but this behavior was changed as of version 1.0.1.

actionrepair

This command or script is invoked only if *actioncheck* returns an error code other than 0.

actionreban

This command or script runs when a ban is reinstated after a daemon restart. If it does not exist, actionban runs instead.

actionstart

This command or script runs when a jail is started.

actionstop

This command or script runs when a jail is stopped.

actionflush

This command or script runs when several or all bans are removed in one fell swoop, such as when Fail2Ban is stopped or is reloaded with a previously-active jail removed, or with *fail2ban-client unban –all*. By default, *actionunban* is called for each banned IP.

Action Tags

Fail2Ban pre-defines several action tags with useful values. To use them in commands or scripts, enclose the variable name in angle brackets:

```
actionban = banomatic.sh <ip> <bancount>
```

will invoke the "banomatic.sh" script with the banned IP address and the number of times that IP address has been banned as arguments.

You can also use angle brackets to refer to variables defined within your configuration file.

These action tags are pre-defined:

name	The name of the jail for this ban
fq-hostname	The fully-qualified hostname of your host (*e.g.* mail.example.com)
fq-shortname	The short hostname of your host (*e.g.* mail)
ip	The banned IP address
ip-rev	The IP address, in reversed order, suitable for reverse DNS lookups. For example, if <ip> is "111.222.333.444", then <ip-rev> will be "444.333.222.111."
ip-host	The banned IP's hostname, if it has a PTR record (which it probably does not, in which case it defaults to "None")
fid	The failure ID. Defaults to the banned IP address unless it has been changed by the filter. This can be

	used to give an ID to general failures that do not include an IP address
family	The IP address family: INET4 or INET6
time	The unix epoch time of the ban. If you use this, you'll probably want to convert it to a human-readable format using the *date* shell command, for example `date -d '@<time>'`
bancount	The total number of bans in the database for the IP address in this jail
jail.banned	The number of IP addresses currently banned for this jail
jail.banned_total	The total number of IP addresses banned for this jail since the last restart
bantime	The length of the ban, in seconds
failures	The number of times the filter matched a log line for this ban
ipjailfailures	The total number of filter matches in the database for this IP in this jail
ipfailures	The total number of filter matches in the database for this IP across all jails
jail.found	The current number of filter matches for this jail, for all IP addresses
jail.found_total	The total number of filter matches since restart for this jail for all IP addresses
matches	A list of the log lines that were matched for this ban
ipjailmatches	A list of all matches in the database for this IP address in this jail
ipmatches	A list of all matches in the database for this IP address across all jails
raw-ticket	An object containing data relevant to this ban, including ip, time, bantime, bancount, failures, and matches.
F-*name*	Custom tags that can be set in the filter regex. See below.
restored	1 if the ban has been restored, for example after a daemon restart, or 0 otherwise.

br	A newline (“\n”) character
sp	A literal space (“ “) character

For cumulative historical variables, such as *bancount* and *ipjailmatches*, the lookback period depends on the database’s retention length, which defaults to 1 day. The retention length is defined by the *dbpurgeage* property in *fail2ban.conf*. (Remember to copy the file to *fail2ban.local* if you make changes).

The “F-*name*” tags are custom tags that can be defined in a jail’s filter (in the filter configuration file in *filter.d*) by enclosing the desired pattern in a pair of `<F-name>. . .</F-name>` tags. For example, if the failregex includes this snippet at an appropriate place:

```
failregex = . . . <F-EMAIL>\w+@(\w+\.?)+</F-EMAIL> . . .
```

then the *F-EMAIL* variable will be pre-populated with the matching email address and can be used with <F-EMAIL> in actions. More details about custom filter tags will be discussed in another article about Fail2Ban filters.

Action scripts can also refer to variables, including variables that define scripts, using python-like format:

```
actionban = %(variable)s
```

The variable “known/*parameter*” is particularly useful. It is pre-defined as the last-known definition for an action or parameter and can be used to refer to an inherited definition.

For example, suppose you want to use the existing *mail.conf* configuration (which uses the “mail” command to send an email notification each time an IP is banned), but you only want to send the email when an IP has been banned at least three times. To do that, you could create a configuration file *mail.local* that looks like this:

```
# mail.local
# Send an email notification for three or more bans from
# the same IP

[Definition]
actionban =    if [<bancount> -ge <toomany>]; then
                    %(known/actionban)s
               fi

[Init]
```

```
toomany = 3
```

Linking Actions to Jails

The actions associated with each jail are defined in *jail.local* with the *action* property. You can add a new action to existing ones in the [DEFAULT] section or in individual jails. Each jail can have more than one action, with each one on a new line beginning with whitespace. When adding a new action, append it to any that already exist so you don't override the existing firewall actions.

For example, if you want to add *mail.local* to a jail (or default) and it already has

```
action = %(action_)s
```

then add your action beneath it, not including the *.local* extension:

```
action = %(action_)s
         mail
```

If you are adding a custom action to an individual jail and there is no existing *action* property, the jail is currently using the default action. You can change the default higher up in *jail.local* or you can add the action property to the jail to retain the default:

```
action =  %(known/action)s
          your_action_here
```

Passing Variables into Actions

Sometimes you may want to override variables defined in your actions' [Init] sections individually per jail. To do this, you can include the variables as comma-separated *key=value* pairs inside square brackets after an action name.

```
action = someaction[key1=value1, key2="value2", …]
```

In this example, two variables are passed into the "notifyme" action for the sshd jail:

```
[sshd]
port     = ssh
```

```
logpath  = %(sshd_log)s
backend  = %(sshd_backend)s
action   = %(known/action)s
           notifyme[token="ABCDEF123456", priority=5]
```

After adding new actions in jail.local, it may not be sufficient to just reload the Fail2Ban daemon, so you should completely restart it:

```
sudo systemctl restart fail2ban
```

And check it's status:

```
sudo systemctl status fail2ban
```

And there you have it. Happy banning!