CONTENTS



// TUTORIAL //

# How Fail2Ban Works to Protect Services on a Linux Server

Updated on August 2, 2022

Security Firewall Monitoring Conceptual

Justin Ellingwood and Alex Garnett


## Introduction

SSH is the de facto method of connecting to a cloud server. It is durable, and it is extensible — as new encryption standards are developed, they can be used to generate new SSH keys, ensuring that the core protocol remains secure. However, no protocol or software stack is totally foolproof, and SSH being so widely deployed across the internet means that it represents a very predictable *attack surface* or *attack vector* through which people can try to gain access.

Any service that is exposed to the network is a potential target in this way. If you review the logs for your SSH service running on any widely trafficked server, you will often see repeated, systematic login

attempts that represent brute force attacks by users and bots alike. Although you can make some optimizations to your SSH service to reduce the chance of these attacks succeeding to near-zero, such as disabling password authentication in favor of SSH keys, they can still pose a minor, ongoing liability.

Large-scale production deployments for whom this liability is completely unacceptable will usually implement a VPN such as WireGuard in front of their SSH service, so that it is impossible to connect directly to the default SSH port 22 from the outside internet without additional software abstraction or gateways. These VPN solutions are widely trusted, but will add complexity, and can break some automations or other small software hooks.

Prior to or in addition to committing to a full VPN setup, you can implement a tool called **Fail2ban**. Fail2ban can significantly mitigate brute force attacks by creating rules that automatically alter your firewall configuration to ban specific IPs after a certain number of unsuccessful login attempts. This will allow your server to harden itself against these access attempts without intervention from you.

In another tutorial, we discussed How to protect SSH with Fail2ban. In this guide, we'll discuss in more depth how Fail2ban actually works and how you can use this knowledge to modify or extend the behavior of this service.

## The Fundamentals of Fail2ban

The purpose of Fail2ban is to monitor the logs of common services to spot patterns in authentication failures.

When fail2ban is configured to monitor the logs of a service, it looks at a **filter** that has been configured specific to that service. The filter is designed to identify authentication failures for that specific service through the use of complex regular expressions. Regular expressions are a common templating language used for pattern matching. It defines these regular expression patterns into an internal variable called `failregex`.

By default, Fail2ban includes filter files for common services. When a log from any service, like a web server, matches the `failregex` in its filter, a predefined action is executed for that service. The `action` is a variable that can be configured to do many different things, depending on the preferences of the administrator.

The default action is to ban the offending host/IP address by modifying the local firewall rules. You can expand this action to, for example, send an email to your system administrator.

By default, action will be taken when three authentication failures have been detected in 10 minutes, and the default ban time is for 10 minutes. This is configurable.

When using the default `iptables` firewall, `fail2ban` creates a new set of firewall rules, also called a chain, when the service is started. It adds a new rule to the INPUT chain that sends all TCP traffic directed at port 22 to the new chain. In the new chain, it inserts a single rule that returns to the INPUT chain. The chain and associated rules are removed if the Fail2ban service is stopped.

## Exploring Fail2ban Service Settings

Fail2ban is configured through several files located within a hierarchy under the `/etc/fail2ban/` directory.

The `fail2ban.conf` file configures some operational settings like the way the daemon logs info, and the socket and pid file it will use. The main configuration, however, is specified in the files that define the per-application "jails".

By default, fail2ban ships with a `jail.conf` file. However, this can be overwritten in updates, so you should copy this file to a `jail.local` file and make adjustments there.

If you already have a `jail.local` file, open it using `nano` or your favorite text editor: Copy

```
$ sudo nano /etc/fail2ban/jail.local
```

If you don't have a `jail.local` file already, or the file you opened was blank, copy over the `jail.conf`

file and then open the new file:

Copy

```
$ sudo cp /etc/fail2ban/jail.conf /etc/fail2ban/jail.local
$ sudo nano /etc/fail2ban/jail.local
```

We will take a look at the options available here and see how this file interacts with other configuration files on the system.

## The Default Section

The first portion of the file will define the defaults for fail2ban policy. These options can be overridden in each individual service's configuration section.

With the comments removed, the entirety of the default section looks something like this:

/etc/fail2ban/jail.local

```
[DEFAULT]

ignoreip = 127.0.0.1/8
bantime = 10m
findtime = 10m
maxretry = 3
backend = auto
usedns = warn
destemail = root@localhost
sendername = Fail2Ban
banaction = iptables-multiport
mta = sendmail
protocol = tcp
chain = INPUT
action_ = %(banaction)s[name=%(__name__)s, port="%(port)s", protocol="%(protocol)s", chain="%
action_mw = %(banaction)s[name=%(__name__)s, port="%(port)s", protocol="%(protocol)s", chain=
            %(mta)s-whois[name=%(__name__)s, dest="%(destemail)s", protocol="%(protocol)s", c
action_mwl = %(banaction)s[name=%(__name__)s, port="%(port)s", protocol="%(protocol)s", chain
            %(mta)s-whois-lines[name=%(__name__)s, dest="%(destemail)s", logpath=%(logpath)s,
action = %(action_)s
```

Let's go over what some of this means:

- **ignoreip**: This parameter identifies IP addresses that should be ignored by the banning system. By default, this is just set to ignore traffic coming from the machine itself, so that you don't fill up your own logs or lock yourself out.
- **bantime**: This parameter sets the length of a ban, in seconds. The default is 10 minutes.
- **findtime**: This parameter sets the window that Fail2ban will pay attention to when looking for repeated failed authentication attempts. The default is set to 10 minutes, which means that the software will count the number of failed attempts in the last 10 minutes.
- **maxretry**: This sets the number of failed attempts that will be tolerated within the `findtime` window before a ban is instituted.
- **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. `inotify` is a built-in Linux kernel feature for tracking when files are accessed, and `pyinotify` is a Python interface to `inotify`, used by Fail2ban.
- **usedns**: This defines whether reverse DNS is used to help implement bans. Setting this to "no" will ban IPs themselves instead of their domain hostnames. The `warn` setting will attempt to look up a 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. This is actually a path to a file located in `/etc/fail2ban/action.d/` called `iptables-multiport.conf`. This handles the actual `iptables` firewall 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.

The rest of the parameters define different actions that can be specified. They pass in some of the parameters that we've defined above using variable substitution within text strings like this:

`%(var_name)s`

The line above would be replaced with the contents of `var_name`. Using this, we can tell that the `action` variable is set to the `action_` definition by default (ban only, no mail alerts).

This, in turn, is configured by calling the `iptables-multiport` action with a list of parameters (service name, port, protocol, and chain) that is needed to perform the ban. The `__name__` is substituted with the name of the service as specified by the section headers below.

### Service Specific Sections

Beneath the default section, there are sections for specific services that can be used to override the default settings. This follows a convention of only modifying the parameters that differ from the normal values (convention over configuration).

Each section header is specified like this:

`[service_name]`

Any section that has the line `enabled = true` will be read and enabled.

Within each section, the parameters are configured, including the filter file that should be used to parse the logs (minus the file extension) and the location of the log files themselves.

Keeping this in mind, the section that specifies the actions for the SSH service looks like this:

/etc/fail2ban/jail.local

```
[SSH]

enabled     = true
port        = ssh
filter      = sshd
logpath     = /var/log/auth.log
maxretry    = 6
```

This enables this section and sets the port to the default "ssh" port (port 22). It tells Fail2ban to look at the log located at `/var/log/auth.log` for this section and to parse the log using the filtering mechanisms defined in the `/etc/fail2ban/filters.d` directory in a file called `sshd.conf`.

All of the other pieces of information that it needs are taken from the parameters defined in the `[DEFAULT]` section. For instance, the action will be set to `action_` which will ban the offending IP address using the `iptables-multiport` banaction, which references a file called `iptables-multiport.conf` found in `/etc/fail2ban/action.d`.

As you can see, the actions in the `[DEFAULT]` section should be general and flexible. Using parameter substitution along with parameters that provide sensible defaults will make it possible to override definitions when necessary.

## Examining the Filter File

In order to understand what is going on in our configuration, we need to understand the filter and action files, which do the bulk of the work.

The filter file will determine the lines that fail2ban will look for in the log files to identify offending characteristics. The action file implements all of the actions required, from building up a firewall structure when the service starts, to adding and deleting rules, and tearing down the firewall structure when the service stops.

Let's look at the filter file that our SSH service called for in the configuration above:

```
$ sudo nano /etc/fail2ban/filter.d/sshd.conf
```

/etc/fail2ban/sshd.conf

```
[INCLUDES]

before = common.conf

[Definition]

_daemon = sshd
failregex = ^%(__prefix_line)s(?:error: PAM: )?[aA]uthentication (?:failure|error) for .* fro
        ^%(__prefix_line)s(?:error: PAM: )?User not known to the underlying authentication mo
        ^%(__prefix_line)sFailed \S+ for .*? from <HOST>(?: port \d*)?(?: ssh\d*)?(: (ruser .
        ^%(__prefix_line)sROOT LOGIN REFUSED.* FROM <HOST>\s*$
        ^%(__prefix_line)s[iI](?:llegal|nvalid) user .* from <HOST>\s*$
        ^%(__prefix_line)sUser .+ from <HOST> not allowed because not listed in AllowUsers\s*
        ^%(__prefix_line)sUser .+ from <HOST> not allowed because listed in DenyUsers\s*$
        ^%(__prefix_line)sUser .+ from <HOST> not allowed because not in any group\s*$
        ^%(__prefix_line)srefused connect from \S+ \(<HOST>\)\s*$
        ^%(__prefix_line)sUser .+ from <HOST> not allowed because a group is listed in DenyGr
        ^%(__prefix_line)sUser .+ from <HOST> not allowed because none of user's groups are l
ignoreregex =
```

The `[INCLUDES]` section header specifies other filter files that are read in before or after this file. In our example, the `common.conf` file is read in and placed before the other lines in this file. This sets up some parameters that we will be using in our configuration.

Next, we have a `[Definition]` section that defines the actual rules for our filter matches. First, we set the name of the daemon we are monitoring by using the `_daemon` parameter.

After that, we go through the actual `failregex` definition, which sets the patterns that will trigger when a matching line in the log file is found. These are regular expressions that match based on the different errors and failures that can be thrown when a user does not authenticate correctly.

Portions of the line like `%(__prefix_line)s` will be substituted with the value of a parameter setup in the `common.conf` file that we sourced. This is used to match the different leading information that operating systems write to log files when they use standard methods. For instance, some lines from the `/var/log/auth.log` might look something like this:

/var/log/auth.log

```
May  6 18:18:52 localhost sshd[3534]: pam_unix(sshd:auth): authentication failure; logname=
May  6 18:18:54 localhost sshd[3534]: Failed password for invalid user phil from 101.79.130
May  6 18:18:54 localhost sshd[3534]: Received disconnect from 101.79.130.213: 11: Bye Bye
```

The highlighted portion is a standard pattern that the operating system inserts to provide more context. After that, there are quite a few different ways that the iptables firewall service writes failure attempts to the log.

We see two separate failures in the first two lines above (a PAM authentication error and a password error). The regular expressions defined in the filter are designed to match any of the possible failure lines. You should not have to adjust any of these lines, but you should be aware of the need to catch all of the log entries that signify an unauthorized use error for the application you are trying to protect if you ever have to create a filter file yourself.

At the bottom, you can see an `ignoreregex` parameter, which is currently blank. This can be used to exclude more specific patterns that would typically match a failure condition in case you want to negate the failure trigger for fail2ban for certain scenarios. We won't be adjusting this.

Save and close the file when you are finished examining it.

## Examining the Action File

Now, let's take a look at the action file. This file is responsible for setting up the firewall with a structure that allows modifications for banning malicious hosts, and for adding and removing those hosts as necessary.

The action that our SSH service invokes is called `iptables-multiport`. Open the associated file Copy

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

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

/etc/fail2ban/action.d/iptables-multiport.conf

```
[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-<n

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
```

The file starts off by sourcing another action file called `iptables-blocktype.conf` that defines the `blocktype` parameter, which configures the restriction that will be set when a client is banned. By default the `blocktype` is set to reject packets and reply to pings sent by banned clients with a rejection message that the port is unreachable. We will use this in our ban rules below.

Next, we get to the rule definitions themselves. The `actionstart` action sets up the iptables firewall when the fail2ban service is started. It creates a new chain, adds a rule to that chain to return to the calling chain, and then inserts a rule at the beginning of the INPUT chain that passes traffic matching the correct protocol and port destinations to the new chain.

It does this by using the values we passed in with the `action` that we defined in our `jail.local` file. The `name` is taken from the section header for each service. The `chain`, `protocol`, and `port` are taken from the `action` line itself in that file.

Here, all of the parameters that are set by the other file are referenced by including the parameter name in angle brackets:

`<param_name>`

When we move down to the companion `actionstop` definition, we can see that the firewall commands are implementing a reversal of the `actionstart` commands. When the Fail2ban service stops, it cleanly removes any firewall rules that it added.

Another action called `actioncheck` makes sure that the proper chain has been created prior to attempting to add ban rules.

Next, we get to the actual banning rule, called `actionban`. This rule works by adding a new rule to our created chain. The rule matches the source IP address of the offending client – this parameter is read in from the authorization logs when the `maxretry` limit is reached. It institutes the block defined by the `blocktype` parameter that we sourced in the `[INCLUDE]` section at the top of the file.

The `actionunban` rule removes this rule. This is done automatically by fail2ban when the ban time has

elapsed.

Finally, we get to the `[Init]` section. This just provides some defaults in case the action file is called without passing in all of the appropriate values.

## How the Fail2ban Service Processes Configuration Files to Implement Bans

Now that we've seen the specifics, let's go over the process that happens when fail2ban starts.

### Loading the Initial Configuration Files

First, the main `fail2ban.conf` file is read to determine the conditions that the main process should operate under. It creates the socket, pid, and log files if necessary and begins to use them.

Next, fail2ban reads the `jail.conf` file for configuration details. It follows this by reading, in alphabetical order, any files found in the `jail.d` directory that end in `.conf`. It adds the settings found in these files to its internal configuration, giving new values preference over the values described in the `jail.conf` file.

It then searches for a `jail.local` file and repeats this process, adapting the new values. Finally, it searches the `jail.d` directory again, reading in alphabetical order files ending in `.local`.

In our case, we only have a `jail.conf` file and a `jail.local` file. In our `jail.local` file, we only need to define the values that differ from the `jail.conf` file. The fail2ban process now has a set of directives loaded into memory that represent a combination of all of the files that it found.

It examines each section and searches for an `enabled = true` directive. If it finds one, it uses the parameters defined under that section to build a policy and decide what actions are required. Any parameters that are not found in the service's section use the parameters defined in the `[DEFAULT]` section.

### Parsing the Action Files to Determine Starting Actions

Fail2ban looks for an `action` directive to figure out what action script to call to implement the banning/unbanning policies. If one is not found, it falls back on the default action determined above.

The action directive consists of the name of the action file(s) that will be read, as well as a key-value dictionary that passes the parameters needed by those files. The values of these often take the form of parameter substitutions by referencing the settings configured in the service's section. The "name" key is usually passed the value of the special `__name__` variable that will be set to the value of the section's header.

Fail2ban then uses this information to find the associated files in the `action.d` directory. It first looks for the associated action file ending in `.conf` and then amends the information found there with any settings contained in an accompanying `.local` file also found in the `action.d` directory.

It parses those files to determine the actions that it needs to take. It reads the `actionstart` value to see the actions it should take to set up the environment. This often includes creating a firewall structure to accommodate banning rules in the future.

The actions defined in this file use the parameters passed to it from the `action` directive. It will use these values to dynamically create the appropriate rules. If a certain variable wasn't set, it can look at the default values set in the action file to fill in the blanks.

### Parsing the Filter Files to Determine Filtering Rules

The parameters for the service in the `jail.*` files also include the location of the log file as well as the polling mechanism that should be used to check the file (this is defined by the `backend` parameter). It also includes a filter that should be used to determine whether a line in the log represents a failure.

Fail2ban looks in the `filter.d` directory to find the matching filter file that ends with `.conf`. It reads this file to define the patterns that can be used to match offending lines. It then searches for a matching filter file ending with `.local` to see if any of the default parameters were overwritten.

It uses the regular expressions defined in these files as it reads the service's log file. It tries each

`failregex` line defined in the `filter.d` files against every new line written to the service's log file.

If the regular expression returns a match, it checks the line against the regular expressions defined by the `ignoreregex`. If this also matches, fail2ban ignores it. If the line matches an expression in the `failregex` but does not match an expression in the `ignoreregex`, an internal counter is incremented for the client that caused the line and an associated timestamp is created for the event.

As the window of time set by the `findtime` parameter in the `jail.*` files is reached (as determined by the event timestamp), the internal counter is decremented again and the event is no longer considered relevant to the banning policy.

If, over the course of time, additional authentication failures are logged, each attempt increments the counter. If the counter reaches the value set by the `maxretry` parameter within the configured window of time, fail2ban institutes a ban by calling the `actioncheck` action for the service as defined in the `action.d/` files for the service. This is to determine whether the `actionstart` action set up the necessary structure. It then calls the `actionban` action to ban the offending client. It sets a timestamp for this event as well.

When the amount of time has elapsed that was specified by the `bantime` parameter, fail2ban unbans the client by calling the `actionunban` action.

## Conclusion

By now you have a fairly in-depth understanding of how fail2ban operates. When you deviate from the standard configuration, it is helpful to know how fail2ban functions in order to manipulate its behavior in a predictable way.

To learn about how to protect other services with fail2ban, you can read How To Protect an Nginx Server with Fail2Ban on Ubuntu 22.04.



## About the authors

Justin Ellingwood Author

Alex Garnett Author
Senior DevOps Technical Writer



Comments

## 2 Comments

Leave a comment...

This textbox defaults to using **Markdown** to format your answer.

You can type **!ref** in this text area to quickly search our full set of tutorials, documentation & marketplace offerings and insert the link!

Sign In or Sign Up to Comment

shannond • June 3, 2015

After opening

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

the config file has, among other things, the following:

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

I use a non-standard ssh port, which I've configured in `/etc/fail2ban/jail.local` - do I also need to change it in `iptables-multiport.conf` ?

Reply

liataian • August 21, 2014

This tutorial is amazing and easy to learn. Thank you for your tutorial. I learn a lot.

Show replies Reply





Popular Topics

Ubuntu

Linux Basics

JavaScript

Python

MySQL

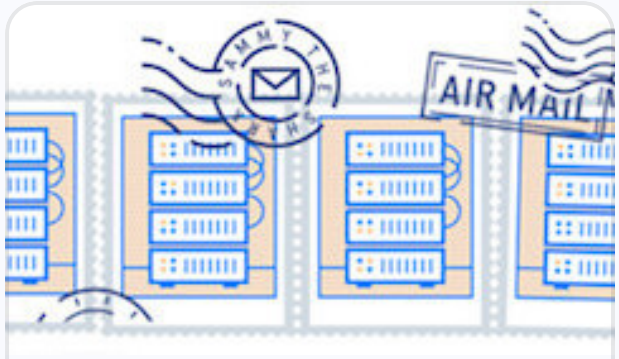
Docker

Kubernetes

All tutorials →

---

Talk to an expert →




## Get our biweekly newsletter

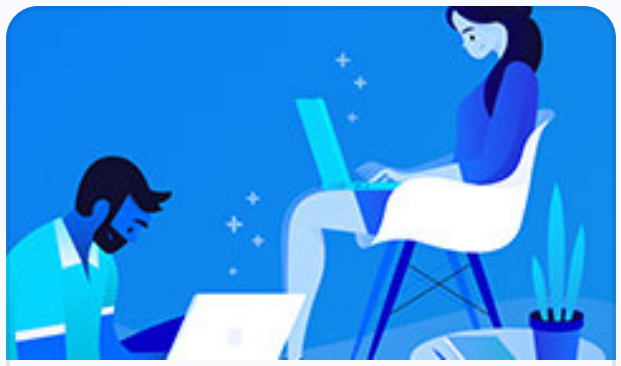
Sign up for Infrastructure as a Newsletter.

Sign up →


## Hollie's Hub for Good

Working on improving health and education, reducing inequality, and spurring economic growth? We'd like to help.

Learn more →



## Become a contributor

Get paid to write technical tutorials and select a tech-focused charity to receive a matching donation.

Learn more →

### Featured Tutorials

Kubernetes Course Learn Python 3 Machine Learning in Python Getting started with Go Intro to Kubernetes

### DigitalOcean Products

App Platform Virtual Machines Managed Databases Managed Kubernetes Block Storage Object Storage

Marketplace VPC Load Balancers