# Intrusion Detection with fail2ban

Chris Binnie

By

For its size, fail2ban, a utility that scans logfiles and bans suspicious IP addresses, punches well above its weight.

I dare say that only a few sys admins haven't heard of *fail2ban* – maybe those starting out or those who have focused on different areas. In my experience, it's quite rare that really small utilities can affect the way you run your servers to the extent that fail2ban has. It certainly explains its popularity.

*fail2ban* is a feather-weight set of scripts that can easily integrate with popular firewalls and, amongst many other things, catch any failed logins for services that you're running and then ban the IP address after a certain number of failed attempts. Admittedly that sounds like quite simple functionality, but when you get down to the innards of the software, it's a truly powerful tool.

I had been using fail2ban on SSH login failures, probably it's most common usage, before I became increasingly annoyed with web server logs filling up with nefarious probes attempting to compromise PHP with remote exploits (and a myriad of other HTTP attacks). It got to the point at which a large proportion of the Apache logs were failed attempts to find hidden directories or non-existent Joomla installations among the legitimate hits on the websites.

I also ran a few mail servers that allowed mail relaying via SASL password authentication, which (and there are other ways of running the authentication side) had system user accounts with PAM checking for correct passwords. I had set the SASL user accounts so that a shell login

couldn't be used to access the server, but I was still more than aware than having a piece of software so readily open to abuse by brute force was far from ideal. So, fail2ban stepped forward yet again; I could simply ban any IP that entered the wrong password three times for as long as I wanted.

From the scenarios above, I hope you will agree that fail2ban can be applied in all sorts of ways. To give you a head start in this article, I'll offer some examples, ranging from those straight out of the documentation to those that were hard won. (Those of you who speak *regular expressions*, or *regex*, as your second language would have found them easy, I'm sure, but I prefer a cogent language that doesn't involve an aching head coupled with eye strain!)

## It Must Be Magic

Rather than the smoke and mirrors that some pieces of software employ to keep their workings secret, fail2ban is transparent in the work it does behind the scenes.

In simple terms fail2ban keeps a close eye on your logfiles, and when a specific pattern is matched within those logs, it triggers an action that you've predefined within its "action" config files. The distinction between its filters and actions in the directory structure is clear once you've installed it.

## Script Kiddies and Port 22

As I mentioned, the most popular usage of fail2ban is stopping probes that try every username ever conceived by mankind on your poor SSH server on port 22. Not only is it a logging irritation, but it is highly insecure, allowing people limitless tries at guessing your username and password combinations. Now is not the time to detail how to secure your SSH server, but for goodness sake, move it to a port other than port 22, and unless you've got a really, really good reason, then permit access by IP address

with TCP Wrappers. Either way, you should consider banning failed logins for a short time in case a poor configuration mistake gives miscreants access inadvertently.

By my reckoning, for user-accessible services on a server, any user who gets a password wrong more than three times is going to need a new password (from a Support Department most likely), so I'd prefer not to allow nefarious types to keep trying repeatedly until they give up. Bear in mind that some scripts attack a port several times a second, so in a minute, they could have just about exhausted all popular logins beginning with the first letter of the alphabet!

Figure 1 shows an SSH logfile example of failed logins. All you have to do (and please bear in mind my comments about eye-strain and regex earlier) is create a regular expression to catch those failed login attempts so you can trigger a response when they're spotted in the logs.

Thankfully, a few useful examples are already available post-installation, with a couple of handy tags to save you from figuring out other ways of spotting patterns – one such tag being <*HOST*>. If you're sitting comfortably, the pattern below matches an SSH login attempt failure which, on a Debian box, lives in the */etc/fail2ban/filter.d/sshd.conf* file. Don't worry this first example isn't too trying. The two salient lines in the config file are as follows:

```
failregex = ^%(__prefix_line)sFailed (?:password|publickey) for .* fro
ignoreregex =
```

By comparing that regex with the logfile in Figure 1, I hope you get the gist of how it pieces together. As I said, don't be too perturbed by the regex complexity; thankfully, several solid examples are included at installation time.

Fortunately, fail2ban is flexible – not just in the services it can monitor but

also across different operating systems, all of which have many versions themselves. So, understanding your potential regex pain, the developers, even in the standard SSH config file, have included a number of *failregex* examples, which should match your system's logfile format. The examples shown below are originally commented out in the config file, ready for you to choose one entry to copy and paste in place of the uncommented line:

```
failregex = ^%(__prefix_line)s(?:error: PAM: )?Authentication failure
            ^%(__prefix_line)s(?:error: PAM: )?User not known to the u
            ^%(__prefix_line)sFailed (?:password|publickey) for .* fro
            ^%(__prefix_line)sROOT LOGIN REFUSED.* FROM <HOST>\s*$
            ^%(__prefix_line)s[iI](?:llegal|nvalid) user .* from <HOST
            ^%(__prefix_line)sUser .+ from <HOST> not allowed because
            ^%(__prefix_line)sauthentication failure; logname=\S* uid=
            ^%(__prefix_line)srefused connect from \S+ \(<HOST>\)\s*$
            ^%(__prefix_line)sAddress <HOST> .* POSSIBLE BREAK-IN ATTE
            ^%(__prefix_line)sUser .+ from <HOST> not allowed because
```

I still need to explain the *ignoreregex* line in the first example. I hope it's clear, though, what the setting in that line, again written in hieroglyphics (sorry regex), would produce: an IP address or hostname match that should NOT trigger an action if it's spotted within that *failregex* pattern in the logfiles. I'll give you some *ignoreregex* examples a little later.

**SASL**

I briefly touched on SASL and mail server password authentication earlier, as well as my concerns about opening up a username and password combination from */etc/shadow* to the Internet as a serious hole for abuse. To allay my fears, a simple fail2ban instance is now configured to catch bad logins in a highly efficient manner.

Here is the *failregex* line for SASL login attempts from the config file */etc/fail2ban/filter.d/sasl.conf*:

```
failregex = (?i): warning: [-._\w]+\[<HOST>\]: SASL (?:LOGIN|PLAIN|(?:
```

A quick hunt through my */var/log/mail.log* files reveals:

```
Dec 20 22:59:16 Ganymede postfix/smtpd[26681]: warning: a-not-so-innoc
```

If you squint long enough at the *failregex* and the mail server log format (Postfix, in this case), you can see that they should probably tie in with each other. After some testing, you should be confident that they do.

## Lay of the Land

Now I'll look a bit closer at how the config files are laid out in fail2ban's directory structure. The above configuration files live in the *filter.d/* directory. A quick *ls* shows a lot of out-of-the-box fail2ban filter examples. A quick warning: if they don't fit your operating system perfectly, you might be using the search engines for a while, unless you have a diploma in regex. For many scenarios, though, the main site http://www.fail2ban.org has how-tos, as well as the bundled examples:

```
apache-404.conf       apache-nohome.conf     courierlogin.conf   pam-ge
apache-auth.conf      apache-noscript.conf   couriersmtp.conf    gssftpd
apache-badbots.conf   apache-overflows.conf  cyrus-imap.conf     lighttp
apache-misc.conf      common.conf            exim.conf           named-r
```

As you can see, a multitude of examples are ready to go, including a few Apache config files.

The main configuration file is called *jail.conf*, and it's commented with lots of helpful instructions. The *ignoreip* setting can take CIDR IP address formats or just single IP addresses, so the Admin doesn't get locked out accidentally; also, that's really handy for testing. Additionally, a default setting that applies to all jails (a brief set of rules about a particular service)

that don't have the *ignoreip* setting explicitly configured is:

```
bantime  = 1
maxretry = 3
```

Here, it's referring to how many opportunities a visitor has to trigger the filter; in this case, it's set really tight and is probably impractical for most services for which users occasionally make mistakes.

The next setting is which back-end daemon to use for polling the logfiles. In Debian's case, I set it to *gamin,* which is great at checking files like mailboxes and logfiles really frequently without putting system load up.

Finally, the *actions.d* directory contains what actions to take and with which firewall. In the *jail.conf* file, other than the level of action required, to all intents and purposes, the remainder of the file is filled with a short config for each service (a jail), switched on or off with *true* or *false*. Here's the entry for SASL:

```
[sasl]

enabled  = true
port        = smtp
filter       = sasl
logpath   = /var/log/mail.log
```

Aside from inheriting the *bantime* and *maxretry* default setting from above, all you have to think about now is the action setting in *jail.conf*, which is relatively straightforward. Also, you can calm fail2ban's reaction to a trigger; however, I prefer all guns blazing, with all the offending log entries and a whois lookup of the banned IP address mailed to me:

```
# ban the IP & send an e-mail with whois report and include relevant l
action_mwl = %(banaction)s[name=%(__name__)s, port="%(port)s", protoco
             %(mta)s-whois-lines[name=%(__name__)s, dest="%(destemail)
```

## Lights, Camera, Action

Now that we know what to send and when to send it, we simply need to know how to ban the offending IP address. I'll use the most popular iptables, the kernel's Netfilter firewall. The file */etc/fail2ban/action.d/iptables-multiport.conf* has these key config lines present:

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

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

To anyone familiar with iptables, the commonplace switches *-I* for *INSERT* and *-D* for *DELETE* for the ban and unban settings should be pretty clear. These are editable, as you'd expect if you want to log further to manipulate iptables in some other way.

## Apache

Several configurations are already in place for Apache, in-hand with the mail POP3/IMAP and FTP services, but I needed a mishmash of the examples, and admittedly it took me ages to get the regex correct.

I had a very specific requirement that I knew no pages were missing on the websites on the web server (which might generate HTTP status response 404s), but I also needed to ignore explicitly some files and directories.

Any other traffic to the web server that I did not specifically allow could therefore be banned if it attempted to access files that didn't exist or probe other parts of the server looking for PHP exploits, for example. Inside my *filter.d/* directory, I created a filter called *apache-misc*. I've shortened the example a little so it's readable, but the first line of the *failregex* after the *[A-Z]* part effectively drops any request to Apache that doesn't include a

forward slash. You might be surprised that pretty much everything legitimate should.

After the slash and inside the brackets are entries for what requests I want banned. The next line down (which is still part of the same *failregex*) beginning with <*HOST*> looks for matches, including the ever-common 404 errors. Finally the *ignoreregex* works beautifully, giving some leeway when it's needed.

```
[Definition]

failregex = <HOST>.*"[A-Z]* /(cms|user|muieblackcat|db|cpcommerce|wp-l
            <HOST>.*\" (502|500|417|416|415|414|413|412|404|405|403|40

ignoreregex = .*\"GET \/(press|mailto|domestic|word).*
```

The result of applying this filter to Apache is, whereas formerly I might have had a hundred attempts per IP probing the server, now I only have three entries in the logs (which all happened in the first second before *gamin* read the logfile and added the iptables rule). I'm delighted with the lack of unwanted logfile noise, and my web servers are safer! If the config is too strict for you, try and remove the 404 entry from the second *failregex* line and up the *bantime* and *maxretry* numbers to see how you get on.

I've barely scratched the surface of this topic here, but I hope it will inspire you to look further into how fail2ban can help secure your servers. After you've installed it, have a look at how many IP addresses are "already banned" in the fail2ban logfile, */var/log/fail2ban.log*, and see how many attempts you've successfully blocked in a day. With its level of reporting and the inherently efficient and controllable configuration, to my mind, it's a knockout piece of software that should help out any sys admin greatly.

## Related content

- Charly's Column

  **Users log on to services such as SSH, ftp, SASL, POP3, IMAP, Apache htaccess, and many more using their names and passwords. These popular access mechanisms are a potential target for brute-force attacks. An attentive bouncer will keep dictionary attacks at bay.**

  more »
- Sshutout and Fail2ban

  **Services that require a username and password for login are potential targets for dictionary attacks. Sshutout and Fail2ban introduce time penalties for invalid attempts.**

  more »
- Magic URL

  **With a highly secure Linux server, you don't need a fixed IP address to connect over a cellphone network from anywhere on the planet if you have a Magic URL.**

  more »
- Admin Workshop: Logrotate

  **Every multi-purpose Linux system produces an enormous amount of log data. To prevent your hard disk from overflowing, a rotating helper application archives logs and gets rid of obsolete data.**

  more »
- Logstash

  **When something goes wrong on a system, the logfile is the first place to look for troubleshooting clues. Logstash, a log**

**server with built-in analysis tools, consolidates logs from many servers and even makes the data searchable.**

more »

comments powered by Disqus