How To: Use fail2ban to Protect SSH

I have a number of servers, including a few on the home office network, that accept SSH connections. Even though they are serving on different (non-standard) SSH ports, there are regular attempts made to break it via brute-force – I can see how some random IP addresses start trying to log in using different standard user names. It’s therefore never too late to use additional software for protecting SSH service, something like fail2ban.
What is fail2ban?

fail2ban is a tool that monitors OS logs, identifies failed connection and authentication (login) attempts and then temporarily bans these IP addresses using IPtables.

The idea is that any IP address that failed to login multiple times within a period of time must be blocked from further attempts to log in on a firewall level. This minimises risks because connections are simply blocked rather than allowed to try another username/password combination.

**INTERESTING:** fail2ban can do a lot more than just protect your SSH service – it has a growing library of contextual log files knowledge.

Install fail2ban in Ubuntu

Even on my Raspberry system I can just do this to install fail2ban:

```
$ sudo apt install fail2ban
```

**IMPORTANT:** double-check that you have **iptables** installed – think it would be installed as part of dependencies for fail2ban.

Once installed, this software needs to be activated – so you
need to start it using **systemctl** or **service command**.

**Configure fail2ban**

Before we can start, it makes sense to customise fail2ban to make sure it’s going to work properly.

Basic settings I focus on are:

- **SSH port** – by default fail2ban will keep blocking standard SSH port 22, which isn’t going to be all that helpful if your SSH service is listening on a different TCP port
- **Configure email** – fail2ban will notify you of new bans/unbans

So just edit the `/etc/fail2ban/jail.conf` file as root. I made the following changes:

```
# Some options used for actions
# Destination email address used solely for the interpolations in
# jail.{conf,local,d/*} configuration files.
destemail = greys@unixtutorial.org

# Sender email address used solely for some actions
sender = root@srv.unixtutorial.org

# E-mail action. Since 0.8.1 Fail2Ban uses sendmail MTA for the
# mailing. Change mta configuration parameter to mail if you want to
# revert to conventional 'mail'.
mta = sendmail
```

Email settings for fail2ban
Specifying custom port 202 for my SSH service

How to Use fail2ban

Start the service:

$ sudo systemctl start fail2ban

and check its log file:

2020-01-09 22:32:55,710 fail2ban.server [6038]: INFO
2020-01-09 22:32:55,712 fail2ban.server [6038]: INFO
Starting Fail2ban v0.10.2
2020-01-09 22:32:55,727 fail2ban.database [6038]: INFO
Connected to fail2ban persistent database '/var/lib/fail2ban/fail2ban.sqlite3'
2020-01-09 22:32:55,731 fail2ban.jail [6038]: INFO
Creating new jail 'sshd'
2020-01-09 22:32:55,779 fail2ban.jail [6038]: INFO
Jail 'sshd' uses pyinotify {}/

[sshd]

# To use more aggressive sshd modes set filter mode = normal
# normal (default), ddos, extra or aggressive
# See "tests/files/logs/sshd" or "filter.d/sshd"
port = 202
logpath = %(sshd_log)s
backend = %(sshd_backend)s
How To Inspect fail2ban Logs

As you can see from the output, the service created a “jail” for SSHd service and started looking at failed SSH login attempts. I started fail2ban at 22:32 last night, and at 2:46am got the first IP address blocked: it found 3 failed logins from 218.93.239.44 and banned it immediately.

You can also check iptables, they might have some IP addresses
blocked already:

```
root@srv:# iptables -nvL
Chain INPUT (policy ACCEPT 0 packets, 0 bytes)
         pkts bytes target     prot opt in     out     source                     destination
           266 17432 f2b-sshd tcp  --  *      *       0.0.0.0/0          0.0.0.0/0
                  multiport dports 202
Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
         pkts bytes target     prot opt in     out     source                     destination
Chain OUTPUT (policy ACCEPT 0 packets, 0 bytes)
         pkts bytes target     prot opt in     out     source                     destination
Chain f2b-sshd (1 references)
         pkts bytes target     prot opt in     out     source                     destination
           0    0 REJECT     all  --  *      *       218.93.239.44          0.0.0.0/0
                  reject-with icmp-port-unreachable
           266 17432 RETURN     all  --  *      *       0.0.0.0/0          0.0.0.0/0
```

That’s it for one day. Hope you’ve learned something new today!

See Also

- SSH reference
- SSH port
- Testing different config for SSH
- SSH port forwarding
Multiple OpenVPN Clients Sharing the Same Certificate

Traditionally I’ve been configuring OpenVPN in a scenario where each client would have a unique certificate. This requires a bit more time initially but is well worth it in terms of security.

Sometimes this is not needed though, so you can probably get away with multiple clients sharing the same certificate. This way you build a client key once and then propagate it across all the clients that you plan on connecting to your VPN server.

While setting this up just now, I noticed a curious thing: by default all clients with the same cert would end up having the same internal IP address. This behaviour is unlikely to be what you wanted though.

So in order to allow multiple OpenVPN clients share the same client certificate but enjoy a unique internal IP address (so that clients could connect to each other, for instance), add the following line to your OpenVPN server:

duplicate-cn

That’s it! Restart your openvpn service and enjoy.