Security

- Brute attack and /var/log/btmp
- Security against Brute attack

Brute attack and /var/log/btmp

`**`/var/log/btmp` is a system security log that records all failed login attempts to your Linux server.**
Every time someone tries to log in via SSH, the terminal, or local console and enters an incorrect password or username, the system logs it here. Because it captures malicious activity, it is a vital tool for checking if your server is under a brute-force attack.

Key Characteristics of `/var/log/btmp`

- **It is a binary file:** You cannot read it using normal text commands like `cat`, `nano`, or `less`. If you try, your screen will fill with unreadable scrambled code.

- **Strict Permissions:** Because it logs failed login details (which sometimes accidentally contain real passwords typed into the username field), only the `root` user can read it.

How to View It Safely
To read this binary file, Linux includes a specific tool called `last` with the `-f` flag. Run this command with `sudo`:
bash

```
sudo last -f /var/log/btmp
```

Use code with caution.

What the output looks like:
text

```
root     ssh:notty    192.168.1.50   Thu Aug 27 17:45 - 17:45  (00:00)
admin    ssh:notty    45.142.120.14  Thu Aug 27 17:42 - 17:42  (00:00)
invalid  ssh:notty    185.220.101.5  Thu Aug 27 17:40 - 17:40  (00:00)
```

Use code with caution.

- **Column 1:** The username attempted (e.g., `root`, `admin`, or completely fake usernames like `invalid`).
- **Column 3:** The **IP address** of the machine or bot trying to break into your server.

Common Issues: Why is it growing large?
If your server has a public IP address and port 22 (SSH) is open to the internet, automated botnets will target it 24/7 trying thousands of common passwords. This can cause `/var/log/btmp` to swell to hundreds of megabytes or gigabytes very quickly.
To check how many failed login attempts have hit your server, you can count the lines in the parsed file:
bash

```
sudo last -f /var/log/btmp | wc -l
```

Use code with caution.

Even with SSH moved to a custom port like `2010`, sophisticated automated bots will eventually scan your server, find the open port, and begin brute-forcing it.
You can use a few quick terminal commands to analyze your `/var/log/btmp` file and find out exactly who is attacking you and if **Fail2ban** is doing its job.

---

1. Find the Top Attacking IP Addresses
To see a ranked list of the specific IP addresses hitting your server with failed login attempts, parse the `btmp` file using this command:
bash

```
sudo last -f /var/log/btmp | awk '{print $3}' | sort | uniq -c | sort -nr | head -n 10
```

Use code with caution.

- **What this does:** It extracts the IP column, counts the duplicate entries, and displays the top 10 most aggressive IP addresses alongside the exact number of failed attempts they made.

2. Find the Most Targeted Usernames
Attackers usually guess common system accounts. Run this to see what usernames they are trying to use to break in:
bash

```
sudo last -f /var/log/btmp | awk '{print $1}' | sort | uniq -c | sort -nr | head -n 10
```

Use code with caution.

- **What to look for:** You will likely see `root`, `admin`, `user`, or `ubuntu`. If you see actual personal usernames that exist on your system, it means the attack is more targeted.

3. Verify if Fail2ban is Actually Blocking Them
Moving SSH to port `2010` means you need to ensure Fail2ban is actually listening to that custom port. Otherwise, it will ignore the attacks.
First, check the status of your SSH jail:
bash

```
sudo fail2ban-client status sshd
```

Use code with caution.

- Look at the **"Banned IP list"** at the bottom of the output. If it shows active IP addresses, Fail2ban is successfully protecting your new port.
- If the list is completely empty despite having thousands of entries in your `btmp` log, Fail2ban might still be looking for port 22.

4. Ensure Fail2ban Knows Your Custom Port
If Fail2ban isn't catching them, open your Fail2ban configuration file (usually `/etc/fail2ban/jail.local`):
ini

```
[sshd]
enabled = true
port    = 2010
logpath = %(sshd_log)s
backend = %(sshd_backend)s
```

Use code with caution.

Ensure `port = 2010` is explicitly set. If you make changes, restart the service:
bash

```
sudo systemctl restart fail2ban
```

Use code with caution.

Your terminal output provides highly valuable security data.
Yes, you are definitely being attacked by automated brute-force bots. The IP address `103.137.184.170` alone has hammered your custom SSH port over **216,000 times**.
The good news is that **Fail2ban is working**. It actively monitors your custom port (as shown by `Journal matches: _SYSTEMD_UNIT=sshd.service`), has banned 427 IPs total, and currently has 7 IPs blocked—including `211.253.8.115`, which is number 10 on your top attackers list.
However, because the attack numbers are so massive (hundreds of thousands of attempts), Fail2ban is letting too many retries through before trigger-banning, or your ban times are too short.

Recommended Hardening Steps
1. Make Fail2ban More Aggressive
Currently, bots are allowed to try thousands of times because your ban duration might be too short, or they reset after a few failures. Let's make it strict. Open your Fail2ban local configuration file:
bash

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

Use code with caution.

Find or create your `[sshd]` section and adjust these parameters:
ini

```
[sshd]
enabled = true
port    = 2010
maxretry = 3          ; Ban them after just 3 failed attempts
findtime = 1d         ; Look for 3 failures within a 24-hour window
bantime  = 1w         ; Ban them for a full week (or use -1 for permanent)
```

Use code with caution.

Save the file (`Ctrl+O`, `Enter`, `Ctrl+X`) and restart Fail2ban:
bash

```
sudo systemctl restart fail2ban
```

Use code with caution.

2. Manually Ban the #1 Threat Right Now
The IP `103.137.184.170` has broken the threshold completely. You can force Fail2ban to ban it instantly without waiting for the next trigger:

bash

```
sudo fail2ban-client set sshd banip 103.137.184.170
```

Use code with caution.

3. Disable Root Logins Over SSH
Your log shows **82,029** attempts trying to guess the password for the `root` user. If you haven't already, completely disable logging directly into the `root` account via SSH.
Open your SSH daemon configuration file:
bash

```
sudo nano /etc/ssh/sshd_config
```

Use code with caution.

Find the line `PermitRootLogin` and change it to `no`:
text

```
PermitRootLogin no
```

Use code with caution.

Save the file and restart your SSH service to apply:
bash

```
sudo systemctl restart ssh
```

Use code with caution.

(Note: Ensure you have your normal `meson` user password or SSH key working before doing this, so you don't lock yourself out).
4. Clean up `/var/log/btmp`
Now that you have analyzed the threat, you can safely clear out this binary log file to reclaim disk space, just like you did with the MariaDB log:
bash

```
sudo truncate -s 0 /var/log/btmp
```

Use code with caution.

To wrap up your server security checklist, let me know:

- Do you log in using a **password** or an **SSH Cryptographic Key**? (Switching to keys stops brute forces completely).
- Would you like help configuring **UFW (the firewall)** to restrict SSH port 2010 access to only your specific home/office IP?

AI responses may include mistakes. Learn more

Using an **SSH key** is the absolute best security choice you could make. Even with 216,000 brute-force password guesses, it is mathematically impossible for those bots to break in because your server completely rejects password authentication for your key-based account.
Since you are already using **UFW**, you can easily block these relentless bots at the firewall layer. This is much more efficient because it drops their network traffic completely before it can even touch or slow down your SSH service.
Here is how to finalize and lock down your setup based on your configuration:
1. Close the Root Exploit Completely
In Ubuntu, if `#PermitRootLogin prohibit-password` is commented out, the system defaults to allowing root login attempts under certain conditions. Since you are using SSH keys for your user account, you should explicitly deny the root user from ever logging in.
Open your configuration file:
bash

```
sudo nano /etc/ssh/sshd_config
```

Use code with caution.

Add or modify this exact line (make sure there is no `#` symbol in front of it):
text

```
PermitRootLogin no
```

Use code with caution.

Save the file (`Ctrl+O`, `Enter`, `Ctrl+X`) and restart SSH:
bash

```
sudo systemctl restart ssh
```

Use code with caution.

2. Block the Worst Attacker in UFW Immediately
Fail2ban works dynamically, but for an IP address as aggressive as `103.137.184.170`, you should block it permanently at the firewall level so it cannot access your Nextcloud instance or any other port:
bash

```
sudo ufw insert 1 deny from 103.137.184.170 to any
```

Use code with caution.

(Using `insert 1` ensures this rule sits at the very top of your firewall checklist, blocking the packet instantly).
3. Ultimate Security: Restrict Port 2010 to Your IP Only

If you always connect to your server from the same location (like your home, office, or a static VPN IP), you can tell UFW to block the *entire world* from port 2010 except for you.
First, delete your generic open SSH rule:
bash

```
sudo ufw delete allow 2010/tcp
```

Use code with caution.

Next, allow **only your specific IP address** to see port 2010:
bash

```
sudo ufw allow from YOUR_HOME_IP to any port 2010 proto tcp
```

Use code with caution.

(Replace `YOUR_HOME_IP` with your actual public IP address, which you can find by searching "what is my IP" on your computer).
Once you run this, the 216,000 automated bot attempts will instantly drop to zero because their network packets will be silently ignored by UFW.

Security against Brute attack

This volume of failed logins is **normal background noise** for any server with port 22 open to the internet — automated bots scan all IPv4 addresses and start hammering common usernames within minutes of your server going online. Your fail2ban and UFW are doing exactly what they're supposed to: stopping them from actually getting in.

That said, you can eliminate most of this traffic entirely. If you only have SSH keys enabled, the thousands of password guesses are harmless and simply never succeed.

Is this normal?

- **1564 failed logins / 192 bans / 7329 firewall blocks** is a typical-to-calm level for a public server. Many freshly-unprotected boxes see tens of thousands of attempts daily.
- The important test: do any of the failed logins *succeed*? If they don't (key-based auth only), this is nuisance traffic, not a breach.
- One thing worth checking: if most failures come from a **single IP**, fail2ban may be misconfigured or its filter may not match your auth log format — otherwise it should be banning that IP. Verify with `sudo fail2ban-client status sshd`.

What you can do to cut it down

The single most effective change is to make password login impossible, so the brute-force attempts can never succeed:

bash

- **Use SSH keys only** — this alone eliminates the overwhelming majority of brute-force risk.
- **Disable root login** (`PermitRootLogin no`) so attackers can't target root directly.
- **Rate-limit at the firewall**: `sudo ufw limit OpenSSH` (allows ~6 connections per 30s per IP, dropping brute-force spikes at the kernel level).
- **Use a non-standard port** (e.g., 2222) — reduces scanner traffic dramatically, though it's security-by-obscurity.

- **Restrict access** to known admin IPs or a VPN (Tailscale/WireGuard) if your IP is static — then port 22 isn't exposed at all.

Before you change anything

- Confirm key-based login **already works** from a fresh session before disabling passwords, or you'll lock yourself out.
- Keep one SSH session open, back up the config (`cp`, then `sshd -t` to validate syntax), then `sudo systemctl reload ssh` — not restart, so existing sessions survive.
- Check that your cloud provider has a web/serial console as a break-glass fallback.

Bottom line: there's nothing special or alarming in your numbers — it's the standard pre-scan traffic every internet-exposed server receives. If you want to reduce it (and the log noise), the fixes are key-only auth + disable root + firewall rate limiting. Once that's in place, the 1564 failures are wasted effort against a door that's already locked.

Want me to check whether any of the failed logins actually succeeded on a real account, or walk through a specific hardening step?

Used ssh keys only and disabled root login and I am using custom port 2010 for ssh. Is there anything if I changed ssh port for fail2ban ?

Yes — you should update fail2ban so its **ban action** targets your port 2010. The good news: fail2ban detects attacks by parsing your auth log, not by watching ports, so it still *sees* the failed logins on any port. The problem is only the ban itself — if fail2ban's action still blocks port 22 (or the `OpenSSH` app profile that points to 22), the banned IPs aren't actually being blocked from connecting to 2010.

What to change

On Ubuntu with UFW, the cleanest way is a dedicated UFW application profile for port 2010, then point the sshd jail at it.

1. Add an app profile for your port in `/etc/ufw/applications.d/openssh-server`:

```
text
```

2. Point the sshd jail at it in `/etc/fail2ban/jail.local`:

text

3. Reload and verify:

bash

The `port = 2010` line tells fail2ban the port to report/ban, and the `action = ufw[application="OpenSSH-2010"]` makes UFW insert a `deny from <ip> to any app OpenSSH-2010` rule that actually drops traffic to 2010.

Important caveat

There's a known failure mode: if you leave fail2ban's default `iptables`/`ufw` action pointing at 22 or forget to update it, `fail2ban-client status sshd` will show bans, but `ufw status` will have no matching deny rules and the IPs keep reaching 2010 — exactly the "fails to block ssh on custom port" problem.

Two checks to confirm it's working after changing the port:

- `sudo fail2ban-client status sshd` — shows the banned IP count.
- `sudo ufw status numbered` — verify a matching `DENY ... to 2010/tcp` (or `to ... app OpenSSH-2010`) rule exists for recently banned IPs.

If you'd rather not maintain a custom profile, the alternative is the simpler stock line `action = ufw` with `banaction = ufw` under `[DEFAULT]` — but that still binds to the `port = 2010` you set, so make sure that's updated.

One extra tip since you moved to 2010: add a second jail watching for activity on port 22 too — legacy scanners may still probe it, and banning them at 22 keeps noise out of your UFW logs.

[OpenSSH-2010] title=Secure shell server (custom) description=OpenSSH on custom port ports=2010/tcpJust append this?

Yes — just append that block to the end of `/etc/ufw/applications.d/openssh-server`. Adding a second `[OpenSSH-2010]` section alongside the existing `[OpenSSH]` (port 22) profile is fine; UFW supports multiple profiles in one file and each is referenced by its own bracketed name.

bash

Append at the bottom:

text

Two quick notes:

- Don't add a second `[OpenSSH]` with the same name, or UFW will complain about a duplicate app — a uniquely-named block is the correct approach.
- After saving, re-read the profile list and confirm it appears:

bash

Then you can use it in boththe firewall rule and the fail2ban action:

bash

If `ufw app list` doesn't show the new profile after saving, just re-run `sudo ufw app update OpenSSH-2010` (or reload UFW) to refresh its parsed application definitions.