Expert Cloud Consulting
Enhance Optimise & Scale
ASCP GPUonCLOUD Pvt Ltd

“Expert Cloud Consulting” -

Documentation for Automated IP Blocking on Apache Web Server Using Python and UFW [Title,18, Arial]

14.April.2025 [Subtitle,14, Arial]

version 1.0

—

Contributed by Yogiraj Deshpande [Normal text,14, Arial]
Approved by Akshay Shinde(In Review)
Expert Cloud Consulting
Office #811, Gera Imperium Rise,
Hinjewadi Phase-II Rd, Pune, India – 411057

“Expert Cloud Consulting”

Documentation for Automated IP Blocking on Apache Web Server Using Python and UFW[Title,18, Arial]

1.0: Document Overview:

This document outlines the procedure to set up an Apache web server on an EC2 instance, monitor Apache logs for failed login attempts, and automatically block IPs that try to access the server with unauthorized attempts (401 errors) more than 3 times. The script will run continuously and dynamically update firewall rules (UFW) to block such IPs.

2.0: Objective:

1. Launch an EC2 Linux instance and install Apache web server.
2. Configure user authentication (Apache Basic Auth) to enable login prompt.
3. Deploy a Python script to monitor Apache logs and automatically block IPs exceeding failed login attempts.
4. Integrate the script as a systemd service for automatic and persistent execution on startup.
5. Ensure firewall rules (UFW) are configured to allow/deny traffic appropriately.

3.0: Prerequisites:

1. **AWS EC2 Instance**:
    - A Linux-based EC2 instance (Ubuntu preferred) with a public IP.
2. **Install Apache Web Server:**
    - Install Apache on the EC2 instance to serve a basic web page.
3. **Install and Configure UFW (Uncomplicated Firewall):**
    - Ensure UFW is installed and configured to block IPs dynamically.
4. **Access Permissions:**
    - SSH access to the EC2 instance.
    - Sudo privileges to install packages and configure the firewall.
5. **Knowledge of:**
    - Basic Linux commands and networking.
    - UFW firewall management.
    - Python scripting.

4.0: Step-by-Step Process:

4.1: Set Up Apache Web Server:

Steps:

1. Launch an EC2 Instance with Ubuntu.

2. SSH into the instance:

```
ssh -i your-key.pem ubuntu@your-ec2-public-ip
```

3. Install Apache:

```
sudo apt update
sudo apt install apache2 -y
```

4. Enable and Start Apache Service

```
sudo systemctl enable apache2
sudo systemctl start apache2
```

5. Check Apache status:

```
sudo systemctl status apache2
```

6. Test Apache by visiting your server's public IP in a browser (http://your-ec2-public-ip). You should see the Apache default page.

4.2: Set Up UFW (Firewall):

1. Install UFW:

```
sudo apt install ufw
```

2. Allow necessary traffic (SSH, Apache):

```
sudo ufw allow OpenSSH
sudo ufw allow 'Apache Full'
```

3. Enable UFW:`

```
sudo ufw enable
```

4. Check UFW status:

```
sudo ufw status
```

Why UFW firewall rules?

Reason: Blocking IPs directly at the firewall level (rather than inside Apache) is much stronger — the malicious user can't even connect to the server anymore, saving system resources.

4.3: Configure Apache Basic Authentication (htpasswd)

```
sudo apt install apache2-utils -y
sudo htpasswd -c /etc/apache2/.htpasswd user1
```

- Edit Apache config:

```
sudo nano /etc/apache2/sites-available/000-default.conf
```

- Inside `<VirtualHost *:80>` add:

```
<Directory "/var/www/html">
    AuthType Basic
    AuthName "Restricted Content"
    AuthUserFile /etc/apache2/.htpasswd
    Require valid-user
</Directory>
```

- Restart Apache:

```
sudo systemctl restart apache2
```

Why Apache Basic Auth (htpasswd)?

Reason: Your Apache page was static → no login = no 401 errors.

We needed a login prompt (Basic Auth) to actually generate 401 Unauthorized logs, which our script can detect and respond to.(After applying this- See Sign-in Prompt Screenshot)

4.4: Install Python3 (if not already available)

```
sudo apt install python3 python3-pip -y
```

4.5: Create the Python Script to Block IPs:

1. Create the Python script (`block_bad_ips.py`):

```
nano block_bad_ips.py
```

2. Add the following script to `block_bad_ips.py`:

```
#!/usr/bin/env python3

import re
import subprocess
from collections import Counter
import time

LOG_FILE = "/var/log/apache2/access.log"
THRESHOLD = 3

# Regex pattern to extract IP and status code from log line
log_pattern = re.compile(r'(\d+\.\d+\.\d+\.\d+).+?"[A-Z]+ .*?"
(\d{3})')

# Dictionary to count failed attempts
failed_ips = Counter()

# Track blocked IPs so we don't block same IP repeatedly
blocked_ips = set()

def block_ip(ip):
    print(f"[INFO] Blocking IP {ip} on port 80 and 443 (HTTP/HTTPS)")
    subprocess.run(["sudo", "ufw", "insert", "1", "deny", "from", ip,
"to", "any", "port", "80"])
    subprocess.run(["sudo", "ufw", "insert", "1", "deny", "from", ip,
"to", "any", "port", "443"])
    blocked_ips.add(ip)
    print(f"[INFO] Blocked IP: {ip}")

def monitor_log():
    print("[INFO] Monitoring Apache access log for failed logins
(401)...")
    with open(LOG_FILE, "r") as file:
        # Move to end of file
        file.seek(0, 2)
        while True:
            line = file.readline()
            if not line:
                time.sleep(0.5)
                continue
```

```
            match = log_pattern.search(line)
            if match:
                ip = match.group(1)
                status = match.group(2)
                if status == '401':
                    failed_ips[ip] += 1
                    print(f"[INFO] {ip} has {failed_ips[ip]} failed
attempts")

                    if failed_ips[ip] > THRESHOLD and ip not in
blocked_ips:
                        block_ip(ip)

if __name__ == "__main__":
    monitor_log()
```

3. Save and exit the file.
4. Make the script executable:

```
chmod +x block_bad_ips.py
```

4.6: Create a systemd Service to Run the Script Automatically:

1. Create a systemd service file:

```
sudo nano /etc/systemd/system/auto-block.service
```

2. Add the following content to `auto-block.service`:

```
[Unit]
Description=Auto Block Bad IPs from Apache Logs
After=network.target

[Service]
ExecStart=/usr/bin/python3 /home/ubuntu/block_bad_ips.py
Restart=always
User=ubuntu

[Install]
WantedBy=multi-user.target
```

3. Reload systemd to recognize the new service:

```
sudo systemctl daemon-reload
```

4. Enable the service to start on boot:

```
sudo systemctl enable auto-block.service
```

5. Start the service:

```
sudo systemctl start auto-block.service
```

6. Check the status of the service:

```
sudo systemctl status auto-block.service
```

Why did we need systemd service?
Reason: You want the script to run automatically (not manually or with cron).
Systemd ensures:

- The script starts at boot automatically.
- The script restarts if it crashes.
- You can easily start/stop/status it with familiar commands.

5.0: How to Stop the Script:

1. Stop the systemd service:

```
sudo systemctl stop auto-block.service
```

2. Disable the service to prevent it from starting automatically on boot:

```
sudo systemctl disable auto-block.service
```

6.0: Final Verification:

- Access the website → Enter wrong password 4 times → Your IP should get blocked automatically.
- → Outcome Preview:

  Blocked IP Shown in UFW: See UFW Status Screenshot Here

  Blocked IP Cannot Access Site Screenshot**:** Blocked Browser Access Screenshot Here
- Confirm blocked IP via:

```
sudo ufw status numbered
```

7.0: Conclusion:

This setup provides an automated solution to detect and block IP addresses that attempt to access the Apache server with multiple failed login attempts (401 errors). The Python script runs in the background and automatically updates the firewall to block malicious IPs, preventing further unauthorized access.

This method ensures that your Apache web server is more secure by automatically mitigating brute-force attempts, with minimal manual intervention required.