

Out-of-Band Defense: Securing VPNs from Password-Spray Attacks with Cloud Automation

Author: [Ryan Wehe, ryan@ryanwehe.com](mailto:ryan@ryanwehe.com)

Advisor: *Clay Risenhoover*

Accepted: *March 29th, 2025*

Abstract

This research examines an out-of-band solution to detect and block password-spray attacks on Remote Access VPN services, addressing vulnerabilities like Cisco's CVE-2024-20481 amid rising threats post-COVID-19. A virtual Cisco Adaptive Security Appliance (ASA_v) was deployed in AWS, using a Python script to monitor logs and block malicious IPs via AWS Network Access Control Lists (NACLs). While effective against controlled brute-force tests, the approach faltered against distributed real-world attacks, prompting a shift to Cisco Security Cloud Control (SCC) and an extended detection window. Limitations persisted with low-volume, multi-source attacks, underscoring the need for scalable cloud defenses, pattern correlation, and threat intelligence integration. Offering an MIT-licensed vendor-agnostic tool, this research enhances VPN security and guides future strategies against sophisticated cyber threats.

1. Introduction

A virtual private network (VPN) is a technology that enables secure remote access over an intermediary network like the Internet. By using encryption and authentication, VPNs ensure the integrity and privacy of communications within a private network, allowing for remote access and control with enhanced security (Kinsey & Stewart, 2020). Over the last several years, particularly during the COVID-19 pandemic, VPN usage surged as businesses needed a way to allow employees to continue their work remotely. This is supported in a study funded by Zscaler, which found, “71% of companies reported they were forced to increase VPN capacity... a third of [companies] increased VPN capacity by over 50%” (Schulze, 2021).

The swift adoption of VPNs by enterprises to handle the surge in employee VPN traffic created new challenges for IT administrators, who were responsible for maintaining the security and availability of the infrastructure. In a 2020 cybersecurity advisory, the Cybersecurity & Infrastructure Security Agency (CISA) and the United Kingdom’s National Cyber Security Centre (NCSC) issued a warning based on their findings of malicious cyber actors scanning and exploiting known vulnerabilities in VPNs and other remote work tools. This specifically targeted IT systems that enterprises had rapidly deployed in response to the COVID-19 pandemic (Cybersecurity and Infrastructure Security Agency, 2020).

1.1. Problem and Background

In April 2024, Cisco Talos published an article regarding a large-scale, brute-force attack campaign against targets, including VPN services, observed since at least March 18, 2024. Talos stated that a successful attack could lead to “unauthorized network access, account lockouts, or denial-of-service conditions” (Cisco Talos, 2024). Many VPN vendors were listed as attack targets, including Cisco, Checkpoint, Fortinet, SonicWall, etc. A brute-force attack “primarily consists in an attacker configuring predetermined values, making requests to a server using those values, and then analyzing the response” (OWASP® Foundation, n.d.). In a worst-case scenario, a malicious actor with many computing resources could perform a brute-force attack and discover valid

credentials to gain network access. Even if the actor doesn't successfully get valid credentials, a large-scale attack could exhaust the resources on the VPN infrastructure, causing it to be unavailable, known as denial of service (DoS). Because so many organizations rely on these VPNs, a brute-force or password spray attack can severely disrupt business operations, highlighting the need to protect this infrastructure.

Cisco is tracking this DoS flaw as CVE-2024-20481 and published a security advisory in late October 2024 with guidance for customers of their VPN servers, Cisco Adaptive Security Appliance (ASA), and Cisco Firepower Threat Defense (FTD). The advisory provides indicators of compromise, or what to look for on the device, indicating that it is actively being exploited by a brute-force (password spray) attack. The three primary log messages are:

- %ASA-6-113005: AAA user authentication Rejected : reason = Unspecified : server = 10.1.2.3 : user = admin : user IP = 192.168.1.2
- %ASA-6-113015: AAA user authentication Rejected : reason = User was not found : local database : user = admin : user IP = 192.168.1.2
- %ASA-6-716039: Group <DfltGrpPolicy> User <admin> IP <192.168.1.2> Authentication: rejected, Session Type: WebVPN.

(Cisco Systems, Inc., 2024)

In September 2024, Cisco published a software update that its customers can install on VPN appliances, named Threat Detection Features (Cisco Systems, Inc., 2024). This software enhancement enables IT administrators to configure a native feature to help prevent DoS attacks by automatically blocking IP addresses exceeding configured thresholds (Cisco Systems, Inc., 2024).

The idea for this research paper stemmed from direct exposure working with VPN appliances that were actively being exploited by this attack. Prior to Cisco releasing the "Threat Detection" features for their VPN appliances, and an enterprise coordinating processes to update to this newer version, there were limited options that could be implemented to stop an active exploitation.

1.2. Research Question

Can an out-of-band solution be implemented to detect and actively block password-spray attacks aimed at Remote Access VPN services?

This paper will explore options an enterprise could implement to enhance protection against brute-force activities targeting VPNs. This is meant to be particularly valuable to an organization under active exploitation, which hasn't been able to update its VPN to the newer software version containing the vendor's new security features. Another goal of this research is to share the source code created that can be tweaked and customized to be adapted to other vendors and even different attacks discovered in the future that a vendor hasn't patched.

1.3. Proposed Solution

A virtual Cisco Adaptive Security Appliance (ASAv) with VPN capabilities will be deployed in Amazon Web Services (AWS) and will be reachable by anyone on the Internet. Logs from the ASAv will be sent and processed on another Linux-based machine to detect and block brute-force username/password attacks against the VPN infrastructure in near real-time. All code published accompanying this research paper will be MIT licensed, allowing enterprises to have free use to modify and distribute as needed in their organization (FOSSA Editorial Team, 2021).

To keep a VPN vendor-agnostic approach, this project will attempt to demonstrate multiple blocking methods, both from cloud components and on the VPN appliance itself.

AWS was chosen as the cloud provider for the research environment due to its widespread use and straightforward API for managing network components. However, concepts could extend to other cloud providers or AWS services. From a high level, logs will be actively monitored and checked against user-defined thresholds, and when exceeded, will block the offending IP address from reaching the VPN infrastructure.

2. Research Environment

VPN gateways from Cisco and other leading vendors can be deployed as physical appliances or virtual machines running the same software (Cisco Systems, Inc., 2023). Most components were deployed to AWS for this project using Terraform, an infrastructure-as-code tool (HashiCorp, Inc., 2024). It is beyond the scope of this paper to dive deep into Terraform. However, the Terraform code used to deploy the ASA v and AWS networking components will be described in the following sections, and the complete code will be available in the published code repository's "asa-tf" directory. The figure below is a visualization of the research environment used for testing.

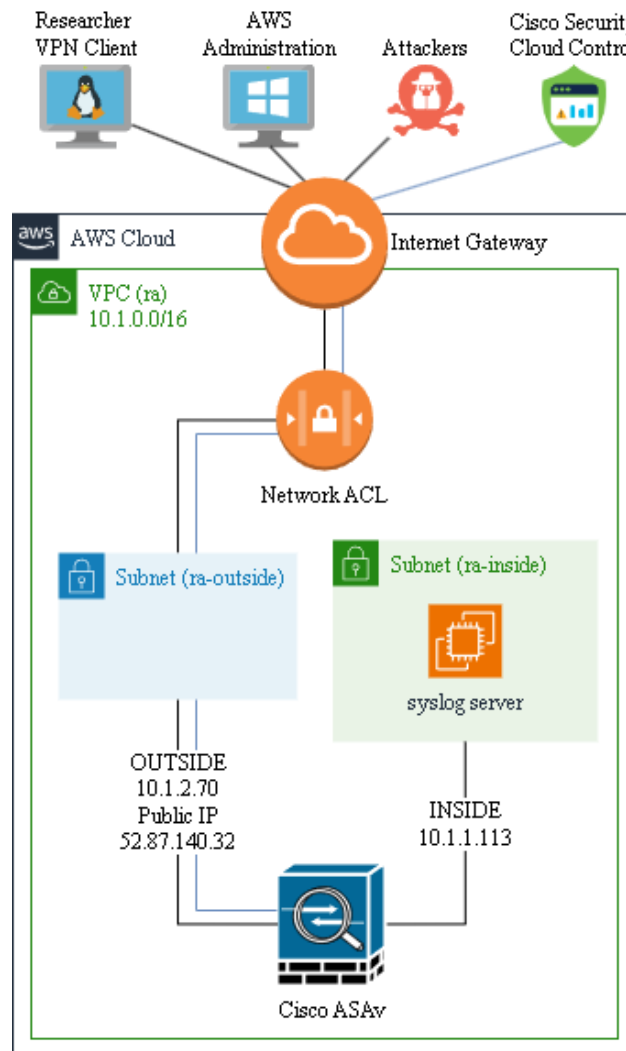


Figure 1: AWS Research Environment

2.1. AWS Networking Components

A dedicated AWS Virtual Private Cloud (VPC) is deployed and is the primary network boundary, using a CIDR block of 10.1.0.0/16. DNS support and hostnames are enabled to facilitate name resolution within the environment.

Two subnets are provisioned within the VPC for inside and outside connectivity. The inside subnet hosts the internal (trusted) interface of the ASA_v, and the outside subnet hosts the untrusted interface that connects to the public Internet. An Internet Gateway attaches to the VPC, enabling inbound and outbound traffic for the outside subnet. Inside and outside route tables are provisioned and associated with their respective subnets. The inside route table routes inside traffic to the ASA_v interface (for VPN-related traffic) or the Internet Gateway. The outside route table directs traffic in the outside subnet to the Internet Gateway. Two AWS security groups are created to control ingress and egress rules. The inside security group permits all inbound and outbound traffic within the trusted zone. The outside security group permits inbound access to VPN services (TCP/443, UDP/443) from all IP addresses. Also, it allows management access (SSH over TCP/22) from only the researcher's public IP address.

2.2. Cisco ASA_v

The Cisco ASA_v is sourced via an AWS Marketplace Amazon Machine Image (AMI). An AWS AMI is a pre-configured snapshot of an operating system stored in AWS that allows an AWS customer to deploy a template launching a virtual machine (Amazon Web Services, Inc., n.d.).

The ASA_v is deployed with three network interfaces. A management interface allows SSH or HTTPS-based management within the inside subnet. The inside interface connects to the trusted subnet. The outside interface binds to an Elastic IP for public reachability over the Internet.

A random administrator password is generated and provided to the administrator as an output of the Terraform command. Once the ASA_v instance is reachable, a pre-populated running configuration (asabootstrap.txt) is applied. This configuration names and configures the interfaces, enables SSH and HTTPS for management, enables the

VPN pool definitions, installs an SSL certificate, uploads a Cisco Secure Client image for VPN users, and configures logging to be sent to the Ubuntu VM described in the next section.

2.3. Log Server & Processor

An Ubuntu version 24.04 virtual machine is deployed in AWS to receive logs from the ASA, process the logs based on a configured threshold, and take a countermeasure to attempt to block the attacker via their public IP address. This VM will be referred to as the syslog server. Since this server does not need to be accessible from the public Internet, the AWS security group is relatively locked down, only allowing inbound network traffic from the researcher's public IP and traffic from the private address of the Cisco ASA. The process of receiving logs will be covered further in section 3.1 Logging.

2.4. VPN Client (and attacker) Configuration

To test the remote-access VPN, another Ubuntu 24.04 virtual machine is deployed locally, running in VMware Workstation. This client is the simulated "attacker," generating repeated authentication attempts against the ASA in AWS.

A commercially available and inexpensive VPN service, Private Internet Access (PIA), is installed on Ubuntu to dynamically change the source IP address for these simulated attacks, providing a more realistic test scenario. If the blocking mechanism works as intended, the attacks will cease. A different server in PIA's network can be selected to attempt the attack again.

Additionally installed on the Ubuntu VM is the OpenConnect VPN client, an open-source software package compatible with connecting to the ASA running in AWS (Orfanos, 2023). The OpenConnect client is chosen because connections can be easily automated in the command line interface (CLI) to repeatedly attempt authentications, just like an actual attacker. A sample OpenConnect command used to test authentication with a bogus username and password is as follows:

```
echo "fake_password" | sudo openconnect -u attacker --passwd-on-stdin --servercert <server certificate removed>  
https://vpn.aws.ryanwehe.me
```

The figure below shows a truncated output of the command above. It is evident that the ASAv received the authentication attempt, and “Login failed” indicates that the username and password combination was not successful.

```
POST https://vpn.aws.ryanwehe.me/  
Attempting to connect to server 52.87.140.32:443  
Connected to 52.87.140.32:443  
...  
GET https://vpn.aws.ryanwehe.me/+webvpn+/index.html  
SSL negotiation with vpn.aws.ryanwehe.me  
Connected to HTTPS on vpn.aws.ryanwehe.me with ciphersuite (TLS1.2)-(ECDHE-X25519)-(RSA-SHA256)-(AES-256-GCM)  
Got HTTP response: HTTP/1.1 200 OK  
...  
HTTP body chunked (-2)  
Please enter your username and password.  
POST https://vpn.aws.ryanwehe.me/+webvpn+/index.html  
Got HTTP response: HTTP/1.1 200 OK  
  
Login failed.
```

Figure 2: Failed VPN Login

The detection section of this paper describes the detection of failed logins in this simulated attack and real-world attacks that the ASAv experienced.

3. Detection and Prevention

As part of this research, a Python script was developed using Python 3.12.3 that monitors logs sent from the ASAv and attempts to act against a suspected attacker. Several user-configured variables need to be configured to set up a similar environment.

In the root directory of the accompanying code repository, files “.env.template” and “config.py.template” must be populated and renamed without “.template”. In the config.py file, the failure threshold value and time window (in seconds) must be

configured. These two values are read by the main Python script to determine when to act on an IP.

The .env file is populated with credentials for an AWS Identity and Access Management (IAM) user account, including an access key ID and a secret access key. As a security best practice, AWS permission policies should be attached to restrict this account to only have the minimum permissions necessary for the functionality, in this case, reading and modifying AWS network access control lists (NACLs). The policies applied to the IAM user are outlined in Appendix B.

3.1. Logging

For centralized log collection, the syslog server leverages rsyslog, an open-source software package installed by default on Ubuntu. The ASA v sends logs on the default port UDP/514. This research is only focused on the three logs mentioned in section 1.1 Problem and Background however, this approach can be customized to recognize interesting logs in various scenarios. The figure below shows a screenshot of the rsyslog configuration file created to match the interesting logs sent by the ASA v and drop everything else. The logs are written to /var/log/asa.log.

```
ubuntu@ip-10-1-2-21:~$ cat /etc/rsyslog.d/30-asa.conf
if $fromhost-ip == '10.1.2.70' then {
    if $rawmsg contains '%ASA-6-113015:' or \
       $rawmsg contains '%ASA-6-113005:' or \
       $rawmsg contains '%ASA-6-716039:' then {
        action(type="omfile" file="/var/log/asa.log")
    }
    stop
}
```

Figure 3: Rsyslog Configuration File

A screenshot of the “asa.log” file shows what can be expected in the log file as the ASA v receives and rejects invalid authentication attempts.

```
ubuntu@ip-10-1-2-21:~/logs$ head 2025-01-19.log
2025-01-19T05:22:15.431134+00:00 ip-10-1-2-70.ec2.internal %ASA-6-113015: AAA user authentication
Rejected : reason = User was not found : local database : user = 1022 : user IP = 77.111.247.44
2025-01-19T05:22:15.431134+00:00 ip-10-1-2-70.ec2.internal %ASA-6-716039: Group <DefaultWEBVPNGrou
p> User <1022> IP <77.111.247.44> Authentication: rejected, Session Type: WebVPN.
2025-01-19T05:22:20.745458+00:00 ip-10-1-2-70.ec2.internal %ASA-6-113015: AAA user authentication
Rejected : reason = User was not found : local database : user = 1022 : user IP = 77.111.247.44
2025-01-19T05:22:20.745458+00:00 ip-10-1-2-70.ec2.internal %ASA-6-716039: Group <DefaultWEBVPNGrou
p> User <1022> IP <77.111.247.44> Authentication: rejected, Session Type: WebVPN.
2025-01-19T05:22:24.306125+00:00 ip-10-1-2-70.ec2.internal %ASA-6-113015: AAA user authentication
Rejected : reason = User was not found : local database : user = 1022 : user IP = 77.111.247.44
2025-01-19T05:22:24.306125+00:00 ip-10-1-2-70.ec2.internal %ASA-6-716039: Group <DefaultWEBVPNGrou
p> User <1022> IP <77.111.247.44> Authentication: rejected, Session Type: WebVPN.
2025-01-19T05:22:28.650238+00:00 ip-10-1-2-70.ec2.internal %ASA-6-113015: AAA user authentication
Rejected : reason = User was not found : local database : user = 1022 : user IP = 77.111.247.44
2025-01-19T05:22:28.650238+00:00 ip-10-1-2-70.ec2.internal %ASA-6-716039: Group <DefaultWEBVPNGrou
p> User <1022> IP <77.111.247.44> Authentication: rejected, Session Type: WebVPN.
2025-01-19T05:22:39.064291+00:00 ip-10-1-2-70.ec2.internal %ASA-6-113015: AAA user authentication
Rejected : reason = User was not found : local database : user = 1022 : user IP = 77.111.247.44
2025-01-19T05:22:39.065010+00:00 ip-10-1-2-70.ec2.internal %ASA-6-716039: Group <DefaultWEBVPNGrou
p> User <1022> IP <77.111.247.44> Authentication: rejected, Session Type: WebVPN.
```

Figure 4: Syslog Server Rejected Authentications Log

The log above shows five rejected login attempts within a time period of less than 60 seconds. All of these attempts came from the same IP address (77.111.247.44) and attempted authentication with the same username, 1022. The login attempts shown in this figure were all simulated using the OpenConnect command.

3.2. Log Monitoring & Thresholds

When it is running, the Python script's main function monitors the asa.log file shown in Figure 4. Because there are only rejection logs, no further filtering is required in this script. Each line the syslog server receives is analyzed to parse the IP address from the log and then processed. The `handle_failed_login` function tracks failed login attempts for each IP address. It uses the current time to append a timestamp for each failed login to a dictionary called `failed_logins`, where the IP address serves as the key. It then filters out any login attempts older than the user's predefined time window (`config.TIME_WINDOW`), ensuring only recent failures are considered. The following screenshot shows the `handle_failed_login` and `main` functions described above.

```
def handle_failed_login(ip):
    """Track failed logins and decide whether to block."""
    now = time.time()
    failed_logins[ip].append(now)
    # Remove timestamps older than TIME_WINDOW
    failed_logins[ip] = [ts for ts in failed_logins[ip] if (now - ts) <= config.TIME_WINDOW]

    if len(failed_logins[ip]) >= config.FAILURE_THRESHOLD:
        print(f"[!] Blocking IP {ip} - too many failures in {config.TIME_WINDOW} seconds")
        block_ip_in_aws(ip)
        # Optionally clear the list to avoid repeated blocking
        failed_logins[ip] = []

def main():
    print(f"Monitoring log file: {config.LOGFILE}")
    print(f"Threshold: {config.FAILURE_THRESHOLD} failures in {config.TIME_WINDOW} seconds")

    for line in tail_f(config.LOGFILE):
        ip = parse_log_line(line)
        if ip:
            handle_failed_login(ip)
```

Figure 5: Handle Failed Login and Main Python Functions

3.3. Blocking Mechanism with AWS NACLs

Network Access Control Lists (NACLs) in AWS serve as stateless firewalls at the subnet boundary, regulating inbound and outbound traffic based on defined rules (Amazon Web Services, Inc., n.d.). This research leverages a NACL in the VPN appliance's public subnet to block malicious IP addresses before they reach the internal network. When the Python script running on the syslog server detects a source IP that triggers configured thresholds of failed authentication attempts in a specified time, it issues an API call to insert a deny rule in the NACL.

The `parse_log_line` function extracts source IP addresses from ASA log entries using a regular expression, while the `handle_failed_login` function tracks the number of recent login failures per IP address. If an IP's failures exceed the defined threshold, `block_ip_in_aws` programmatically creates a deny rule in the NACL, ensuring that subsequent connections from the offender never reach the Cisco ASA.

3.4. Initial Detection and Prevention Testing

To verify the detection and blocking mechanism, an initial test was run with a threshold of five failed authentication attempts within 60 seconds. The VPN client described in Section 2.4 repeatedly attempted authentication with invalid credentials, causing the Cisco ASA to log each failure. The Python script analyzed these logs in real time, tracking failures per IP address through the `handle_failed_login` function.

Once the threshold was reached, the script called `block_ip_in_aws` created a deny rule in the AWS Network Access Control List (NACL). Figure 6 is a screenshot showing the simultaneous console outputs of the attacker VM (top) and the syslog server running the Python detection and prevention code (bottom). The attacker VM first runs a `curl` command to verify the public IP address of the machine, followed by a `for` loop running the `OpenConnect` command 5 times simulating the repeated connection attempts. The bottom half of the screenshot taken on the syslog server indicates the AWS NACL rule was added after the threshold was hit.

The screenshot displays two terminal windows. The top window, titled 'rwehe@ubuntuvvm:~\$', shows the execution of a script that first checks the public IP using `curl https://icanhazip.com`, which returns `77.81.142.121`. It then enters a loop of 5 failed authentication attempts using `openconnect` with a fake password. The bottom window, titled 'ubuntu@ip-10-1-2-21:~/5901_code\$', shows the output of a Python script monitoring the log file `/var/log/asa.log`. The script reports 'Threshold: 5 failures in 60 seconds' and then '[!] Blocking IP 77.81.142.121 - too many failures in 60 seconds'. A subsequent line shows '[+] Successfully added deny rule for 77.81.142.121 with rule number 20'. A white callout box with the text 'Simulated attacker IP address matches blocked IP' has an arrow pointing from the IP address in the script's output to the IP address in the attacker's terminal output.

```

rwehe@ubuntuvvm:~$ curl https://icanhazip.com
77.81.142.121
rwehe@ubuntuvvm:~$ for i in {1..5}; do
    echo "fake_password" | sudo openconnect -u attacker \
        --passwd-on-stdin \
        --servercert https://vpn.aws.ryanwehe.me >/dev/null 2>&1
    sleep 2
done
rwehe@ubuntuvvm:~$

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL PORTS
○ (venv) ubuntu@ip-10-1-2-21:~/5901_code$
○ (venv) ubuntu@ip-10-1-2-21:~/5901_code$
○ (venv) ubuntu@ip-10-1-2-21:~/5901_code$ python3 main.py
Monitoring log file: /var/log/asa.log
Threshold: 5 failures in 60 seconds
[!] Blocking IP 77.81.142.121 - too many failures in 60 seconds
[+] Successfully added deny rule for 77.81.142.121 with rule number 20

```

Figure 6: Attack Successfully Blocked

Figure 7 provides a view of the AWS Console displaying the deny entry. Further attempts from the same IP address failed to reach the ASA_v, confirming that the threshold-based approach effectively blocked rapid brute-force attempts under controlled conditions.

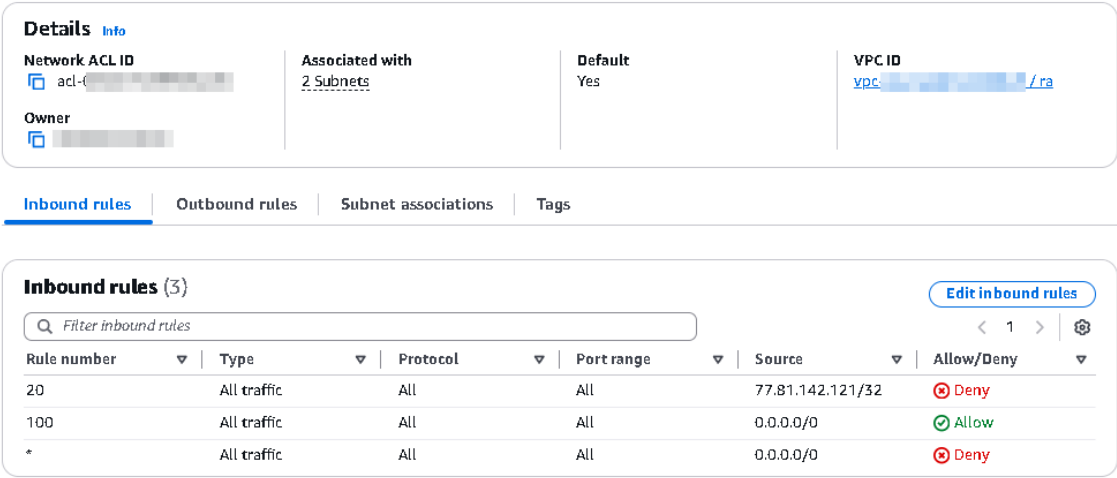


Figure 7: AWS VPC Dashboard Confirming NACL Rule

The next section examines why these settings proved insufficient when real attackers utilized more distributed methods.

3.5. Emergence of External Attacks

The Cisco ASA_v instance in AWS included a valid DNS A record (vpn.aws.ryanwehe.me) and a publicly reachable IP address, which provided the necessary conditions for external attackers to discover and target the service. During the first four days of operation, only locally generated testing attempts appeared in the logs, matching the activity described in earlier sections. After the fifth day, external entities outside of the control of this research environment began performing what appeared to be password spray attacks from a wide range of IP addresses. Table 1 documents the number of rejected login attempts and unique IP addresses recorded daily from N+0 to N+23, where N+0 marks the initial deployment date.

| Days Since Deployment (N+) | Rejected Login Attempts | Unique IP Addresses |
|----------------------------|-------------------------|---------------------|
| N+0-4 | 0 | N/A (attacker VM) |
| N+5 | 32 | 1 |

| | | |
|------|-------|-----|
| N+6 | 358 | 1 |
| N+7 | 236 | 4 |
| N+8 | 6515 | 129 |
| N+9 | 13984 | 158 |
| N+10 | 14898 | 158 |
| N+11 | 13944 | 157 |
| N+12 | 13621 | 157 |
| N+13 | 14160 | 157 |
| N+14 | 15725 | 162 |
| N+15 | 15505 | 156 |
| N+16 | 14586 | 156 |
| N+17 | 14502 | 160 |
| N+18 | 12883 | 154 |
| N+19 | 23457 | 171 |
| N+20 | 29193 | 169 |
| N+21 | 21490 | 169 |
| N+22 | 23006 | 166 |
| N+23 | 18999 | 648 |

Table 1: External Authentication Events Recorded

Appendix A provides details on the log-filtering process used to generate these metrics. The data indicates that external attackers did not appear immediately, but increased login attempts significantly after day five. The surge included password spray activities that originated from many distinct IP addresses, often evading the threshold-based blocking approach described in Section 3.4.

3.6. Limitations of the Initial Detection and Prevention Strategy

The initial strategy relied on two primary components: a threshold of five failed authentication attempts within 60 seconds and a mechanism to block offending IP addresses at the AWS Network Access Control List (NACL) layer. Although these measures effectively stopped basic brute-force attempts during controlled tests, further analysis revealed significant limitations when external attackers employed distributed password-spray techniques.

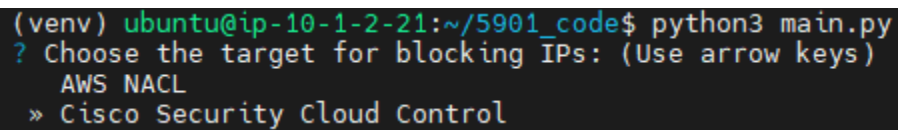
First, the threshold of five attempts per IP in 60 seconds failed to capture a large number of sources. Attackers who spread out login attempts across many unique IP addresses circumvented the script’s per-IP rate limit. As a result, only a small fraction of the malicious traffic ever met the threshold necessary to trigger an AWS NACL block.

The logs documented hundreds of different IPs, each producing relatively few attempts per minute, ultimately remaining undetected by the initial approach.

Second, AWS NACLs have a finite rule capacity. According to AWS documentation, rules per network ACL allow only 20 inbound rules by default, with the option to expand the quota up to 40 (Amazon Web Services, Inc., 2025). This rule limitation proved insufficient for handling the exponential growth in unique attacker IPs. By day eight of recording external attacks, dozens of malicious addresses appeared daily, which exhausted the available NACL entries. Once the NACL reached capacity, additional rules could not be inserted, rendering the IP-blocking approach unsustainable for large-scale threats.

3.7. Revised Blocking Options

To address the AWS NACL limitations, the Python script was updated to offer an additional blocking method, using Cisco Security Cloud Control (SCC). SCC provides a cloud-based solution for managing security policies in distributed environments. It centralizes configuration, automation, and real-time visibility for Cisco security products, including Cisco ASA (Cisco Systems, Inc., 2024). When the revised script launches, it prompts the administrator to choose either AWS NACL blocking or SCC-based blocking. If the SCC option is selected, the script adds offending IP addresses to a dedicated Network Group Object, named blocklist, which enforces traffic denial on the ASA. The screenshot below shows the revised script as it is launched.



```
(venv) ubuntu@ip-10-1-2-21:~/5901_code$ python3 main.py
? Choose the target for blocking IPs: (Use arrow keys)
  AWS NACL
  » Cisco Security Cloud Control
```

Figure 8: Revised Python Script Launch

Along with this new blocking option, the detection logic was updated to use a 1200-second (20-minute) time window, while retaining the threshold of five failures per IP. After running this script for eight days, the environment recorded 350 unique IP addresses that generated 119,764 failed connection attempts. During the same period, 70 IP addresses met or exceeded the threshold and were added to the blocklist. The truncated

screenshots below show the GUI of SCC as well as the configuration on the ASA where the IP address get added, once they meet the threshold.

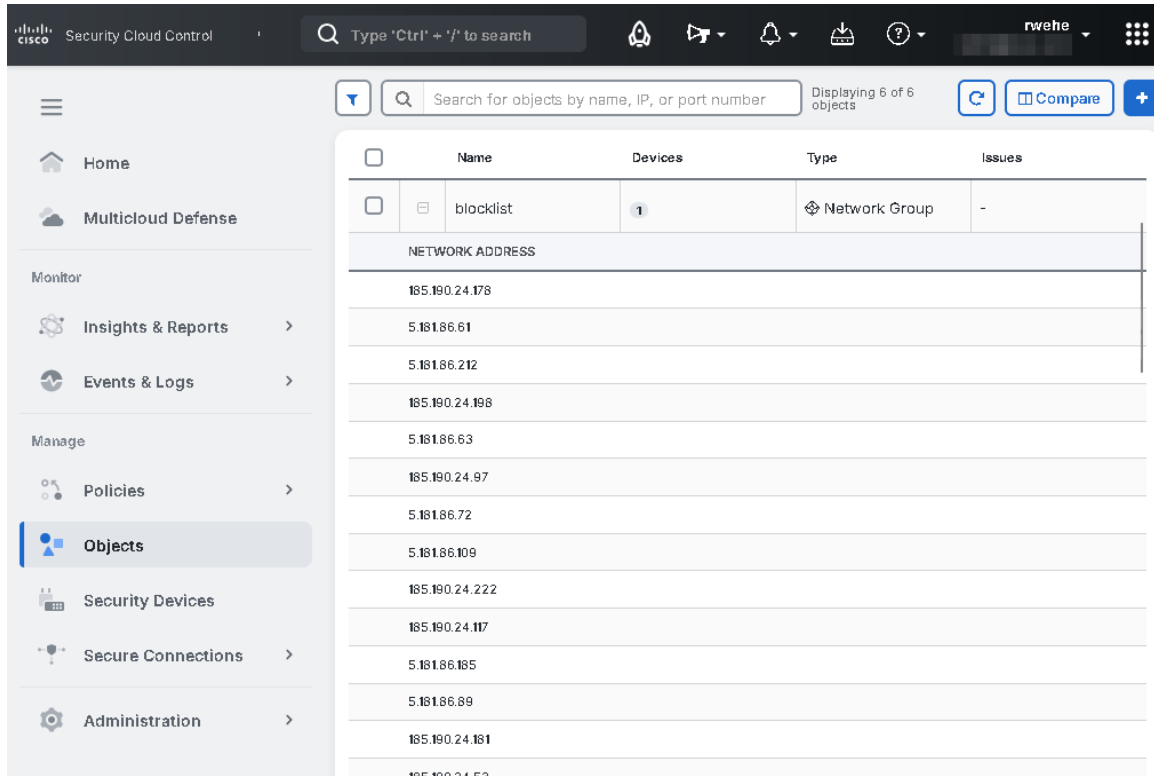


Figure 9: Cisco Security Cloud Control - Blocklist Network Group


```

ra-asa-1# sh run | beg blocklist
object-group network blocklist
description list of IP addresses to be blocked
network-object host 185.190.24.178
network-object host 5.181.86.61
network-object host 5.181.86.212
network-object host 185.190.24.198
network-object host 5.181.86.63
network-object host 185.190.24.97
network-object host 5.181.86.72
network-object host 5.181.86.109
network-object host 185.190.24.222
network-object host 185.190.24.117
network-object host 5.181.86.185
network-object host 5.181.86.89
network-object host 185.190.24.181
network-object host 185.190.24.53
network-object host 87.251.67.251
network-object host 193.37.69.141
network-object host 45.138.16.239
network-object host 185.220.101.63
network-object host 94.142.244.16
network-object host 192.42.116.217
network-object host 192.42.116.194

```

Figure 10: Blocklist on Cisco ASA (CLI)

While these adjustments captured more attackers than the original 60-second window, a substantial number of IPs remained below the threshold. This outcome highlights the evolving nature of password-spray campaigns; attackers often distribute login attempts so that each IP triggers only a few failures over a longer period.

4. Recommendations and Implications

Organizations facing large-scale password-spray attacks require flexible, scalable measures that adapt to rapidly shifting IP sources, multiple credential attempts, and evolving network protocols. The initial script-based blocking approach and subsequent revisions highlight the feasibility of automated responses and the inherent challenges distributed threats pose. The following subsections offer potential strategies for refining detection capabilities and strengthening the overall security posture of VPN infrastructures.

4.1. Leveraging More Scalable Cloud-Based Blocking

AWS Network Access Control Lists (NACLs) present a straightforward way to block traffic at the subnet boundary, but they accommodate only a limited number of deny rules. A more scalable approach involves migrating the script's blocking logic to solutions such as AWS Web Application Firewall (WAF) or AWS Shield. These services support extensive rule sets, accommodate high numbers of distinct IP addresses, and allow more granular traffic analysis. Rather than relying on each IP exceeding a static threshold, a WAF can implement rate-limiting rules and deep packet inspection, making it more difficult for attackers to evade detection by distributing attempts across multiple hosts.

4.2. Correlating Attempts Across Multiple Usernames

This research focuses on individual IP thresholds, counting the number of failed attempts before triggering a block. Modern password-spray attacks often rely on rotating IP addresses and usernames, resulting in a low volume of failures per IP–username pair. Tracking the total number of failures linked to a single IP or analyzing attempts across multiple usernames can help uncover patterns that remain undetected under a strictly per-IP model. An opportunity for future research could refine this code and implement logic to flag scenarios where one IP attempts multiple usernames over a short time frame or where multiple IPs attack the same account in a coordinated manner. Correlating this data would enable more sophisticated detection strategies with less of a possibility of compromising legitimate user activity.

4.3. Accounting for IPv6 Traffic

Another area for possible expansion of this research is to account for IPv6 traffic. This environment's existing detection and blocking strategies focus exclusively on IPv4, leaving IPv6-based connections relatively unchecked. Attackers can exploit this gap by shifting brute-force attempts to an IPv6 range that the script does not currently parse or block. Extending the regular expressions and ACL configurations to include IPv6 addresses would help ensure that future threats do not bypass the established safeguards.

With that said, if an organization doesn't have a requirement to use IPv6, it could be blocked altogether as a mitigating technique.

4.4. Integrating External Threat Intelligence

Password-spray campaigns often involve IP addresses already flagged maliciously by reputable threat intelligence feeds. Cisco Talos, for instance, cataloged thousands of addresses tied to large-scale brute-force attempts (Cisco Talos, 2024). Importing this Talos feed into the defined blocklist would proactively deny traffic from known malicious sources, even before an IP address accumulates enough failed attempts locally. This measure could significantly shorten the exposure window and reduce the risk of damage during an active attack. It requires regular updates to keep the blocklist current and an evaluation of false positives if legitimate IPs appear on these lists.

After continuously running the Python script blocking for 2 weeks, a comparison was made between the IP addresses that met the thresholds to be blocked against Talos' published list of IP addresses. Appendix C details the steps taken to compare and shows 21 IP addresses in common. This indicates that external attackers are still using some of the same IP addresses that they were nearly a year ago when Talos classified them as malicious. An organization seeking to prevent brute-force activity should proactively block IP addresses identified by Talos or other reputable threat intelligence firms.

5. Conclusion

This research paper has explored the feasibility of an out-of-band solution to detect and block password-spray attacks on Remote Access VPN services, targeting vulnerabilities like CVE-2024-20481. Deploying a virtual Cisco Adaptive Security Appliance (ASAv) in AWS, the research implemented a Python-based log monitoring system that initially blocked malicious IPs via AWS Network Access Control Lists (NACLs). Controlled tests successfully mitigated rapid brute-force attempts, but real-world distributed attacks from numerous IPs exposed limitations, including AWS NACL capacity constraints and undetected low-volume attempts. A revised Cisco Security

Cloud Control (SCC) approach and a 20-minute detection window improved blocking efficacy, yet many attackers evaded capture by spreading efforts across multiple sources.

The Python script and Terraform code developed for this research provide a foundation for organizations looking to implement a similar defensive strategy, with the flexibility to adapt based on their existing technology stack. Future research could focus on improving AWS scalability by integrating AWS WAF and leveraging threat intelligence sources like Cisco Talos. Further refinement of detection methods, such as correlating attack patterns across usernames and IPs, addressing IPv6 traffic, and incorporating machine learning, could enhance the accuracy of the detection logic. Expanding support beyond Cisco and AWS would make this approach more versatile, allowing organizations to integrate it with a broader range of security technologies and infrastructure.

References

- Amazon Web Services, Inc. (2025). *Amazon VPC quotas*. Retrieved from Amazon Virtual Private Cloud: <https://docs.aws.amazon.com/vpc/latest/userguide/amazon-vpc-limits.html#vpc-limits-nacls>
- Amazon Web Services, Inc. (n.d.). *Network ACL basics*. Retrieved from Amazon Virtual Private Cloud User Guide: <https://docs.aws.amazon.com/vpc/latest/userguide/nacl-basics.html>
- Amazon Web Services, Inc. (n.d.). *What is AWS Marketplace?* Retrieved from AWS Documentation: <https://docs.aws.amazon.com/marketplace/latest/userguide/what-is-marketplace.html>
- Cisco Systems, Inc. (2023, December 13). *Chapter: Deploy the ASA Virtual On the AWS Cloud*. Retrieved from Cisco Secure Firewall ASA Virtual Getting Started Guide, 9.20: https://www.cisco.com/c/en/us/td/docs/security/asa/asa920/asav/getting-started/asa-virtual-920-gsg/asav_aws.html
- Cisco Systems, Inc. (2024, October 23). *Cisco Adaptive Security Appliance and Firepower Threat Defense Software Remote Access VPN Brute Force Denial of Service Vulnerability*. Retrieved from Cisco Security Advisories: <https://sec.cloudapps.cisco.com/security/center/content/CiscoSecurityAdvisory/cisco-sa-asafthd-bf-dos-vDZhLqrW>
- Cisco Systems, Inc. (2024, November 19). *Cisco Security Cloud Control At-a-Glance*. Retrieved from <https://www.cisco.com/c/en/us/products/collateral/security/security-cloud-control-aag.html>

- Cisco Systems, Inc. (2024, October 25). *Configure Threat Detection for Remote Access VPN Services on Secure Firewall ASA*. Retrieved from Configuration Examples and TechNotes: <https://www.cisco.com/c/en/us/support/docs/security/secure-firewall-asa/222315-configure-threat-detection-services-for.html>
- Cisco Talos. (2024, April 16). *Large-scale brute-force activity targeting VPNs, SSH services with commonly used login credentials*. Retrieved from <https://blog.talosintelligence.com/large-scale-brute-force-activity-targeting-vpns-ssh-services-with-commonly-used-login-credentials/>
- Cybersecurity and Infrastructure Security Agency. (2020, April 8). *COVID-19 Exploited by Malicious Cyber Actors*. Retrieved from Cybersecurity Advisory: <https://www.cisa.gov/news-events/cybersecurity-advisories/aa20-099a>
- FOSSA Editorial Team. (2021, January 27). *Open Source Software Licenses 101: The MIT License*. Retrieved from Open Source License Compliance: <https://fossa.com/blog/open-source-licenses-101-mit-license/>
- HashiCorp, Inc. (2024, November 22). *What is Terraform?* Retrieved from Intro to Terraform: <https://developer.hashicorp.com/terraform/intro>
- Kinsey, D., & Stewart, J. M. (2020). *Network Security, Firewalls, and VPNs, 3rd Edition*. Jones & Bartlett Learning.
- Orfanos, D. P. (2023, May 1). *OpenConnect Project*. Retrieved from OpenConnect Readme: <https://gitlab.com/openconnect/openconnect/-/blob/master/README.md>

OWASP® Foundation. (n.d.). *Brute Force Attack*. Retrieved January 17, 2025, from
Community, Attacks: [https://owasp.org/www-
community/attacks/Brute_force_attack](https://owasp.org/www-community/attacks/Brute_force_attack)

Schulze, H. (2021). *2021 VPN Risk Report*. Cybersecurity Insiders. Retrieved from
[https://info.zscaler.com/rs/306-ZEJ-256/images/2021-VPN-Risk-Report-
Zscaler.pdf](https://info.zscaler.com/rs/306-ZEJ-256/images/2021-VPN-Risk-Report-Zscaler.pdf)

Appendix A

To perform an initial log analysis against the large “asa.log” file, the first step taken is to break the large log file into smaller chunks based on the day the logs occurred. Copy the log file into another directory so that ongoing logging and monitoring are not impacted.

```
cd ~/log_analysis
```

```
sudo cp /var/logs/asa.log ~/log_analysis/asa.log
```

Take ownership of the log file as the ubuntu user

```
sudo chown ubuntu:ubuntu asa.log
```

Break the logs out into individual files by date

```
awk -F'T' '{print >> substr($1, 1, 10) ".log"}' asa.log
```

Iterate over each file to count the total number of rejected authentications (only considering syslog ID 113015).

```
for file in ~/log_analysis/*.log; do
    count=$(grep -c "113015" "$file")
    echo "$file: $count"
done
```



```

ubuntu@ip-10-1-2-21:~/logs$ for file in ~/logs/*.log; do
    count=$(grep -c "113015" "$file")
    echo "$file: $count"
done
/home/ubuntu/logs/2025-01-17.log: 2
/home/ubuntu/logs/2025-01-19.log: 60
/home/ubuntu/logs/2025-01-22.log: 32
/home/ubuntu/logs/2025-01-23.log: 358
/home/ubuntu/logs/2025-01-24.log: 236
/home/ubuntu/logs/2025-01-25.log: 6515
/home/ubuntu/logs/2025-01-26.log: 13984
/home/ubuntu/logs/2025-01-27.log: 14898
/home/ubuntu/logs/2025-01-28.log: 13944
/home/ubuntu/logs/2025-01-29.log: 13621
/home/ubuntu/logs/2025-01-30.log: 14160
/home/ubuntu/logs/2025-01-31.log: 15725
/home/ubuntu/logs/2025-02-01.log: 15505
/home/ubuntu/logs/2025-02-02.log: 14586
/home/ubuntu/logs/2025-02-03.log: 14502
/home/ubuntu/logs/2025-02-04.log: 12883

```

Figure 11: Rejected IP Addresses by Total Count

Count unique IP addresses in each of the log files using cut, sort, uniq and grep.

```

for file in *.log; do
    echo -n "Unique IPs in $file: " && echo $(cat "$file" |
grep 1130 | cut -d "=" -f 4 | sort -n | uniq -c | sort -rn |
wc -l)
done

```

```

rwehe@ubuntuvm:~/logs/split$ for file in *.log; do     echo -n "Unique IPs in $file: "
&& echo $(cat "$file" | grep 1130 | cut -d "=" -f 4 | sort -n | uniq -c | sort -rn | wc
-l); done
Unique IPs in 2025-01-17.log: 1
Unique IPs in 2025-01-19.log: 4
Unique IPs in 2025-01-22.log: 1
Unique IPs in 2025-01-23.log: 1
Unique IPs in 2025-01-24.log: 4
Unique IPs in 2025-01-25.log: 129
Unique IPs in 2025-01-26.log: 158
Unique IPs in 2025-01-27.log: 158
Unique IPs in 2025-01-28.log: 157
Unique IPs in 2025-01-29.log: 157
Unique IPs in 2025-01-30.log: 157
Unique IPs in 2025-01-31.log: 162
Unique IPs in 2025-02-01.log: 156
Unique IPs in 2025-02-02.log: 156
Unique IPs in 2025-02-03.log: 160
Unique IPs in 2025-02-04.log: 154
Unique IPs in 2025-02-05.log: 171
Unique IPs in 2025-02-06.log: 169
Unique IPs in 2025-02-07.log: 169
Unique IPs in 2025-02-08.log: 166
Unique IPs in 2025-02-09.log: 648
Unique IPs in 2025-02-10.log: 170

```

Appendix B

As a security best practice, a new AWS IAM user credential was created with the sole purpose of reading and modifying AWS NACLs. The screenshot below shows the permissions given to this IAM user with these limited permissions.

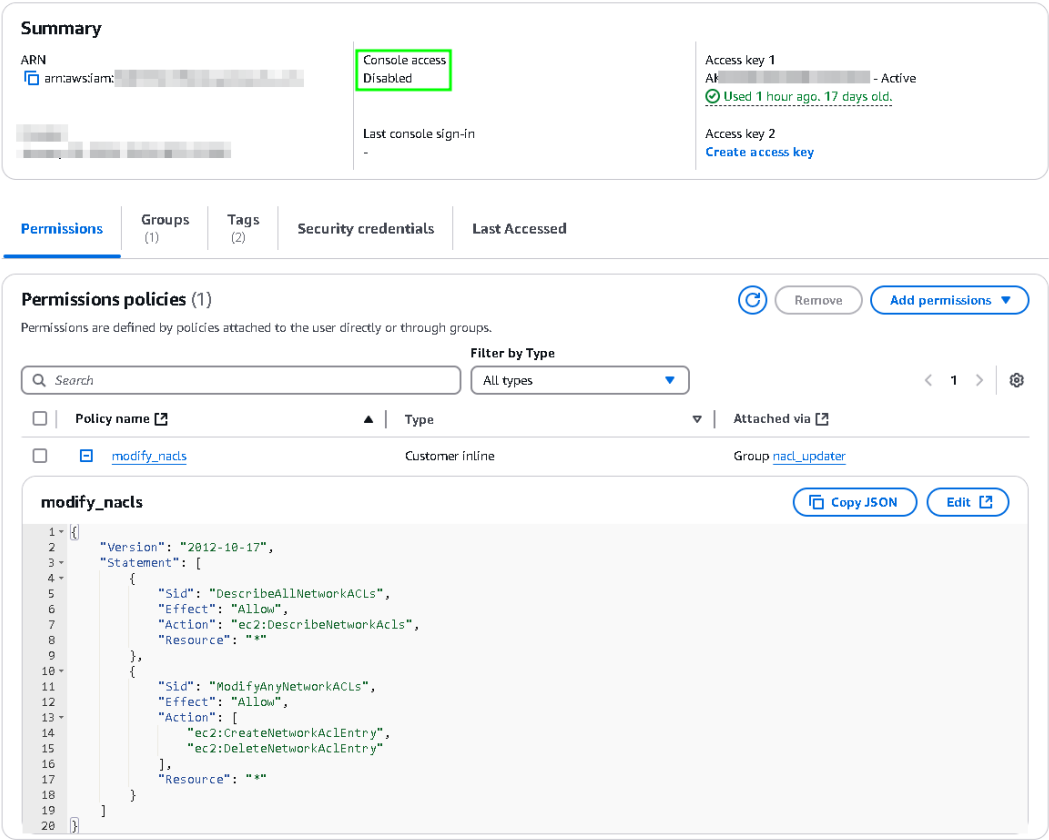


Figure 12: AWS IAM User Permissions

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "DescribeAllNetworkACLs",
      "Effect": "Allow",
      "Action": "ec2:DescribeNetworkACLs",
      "Resource": "*"
    },
    {
      "Sid": "ModifyAnyNetworkACLs",
      "Effect": "Allow",
      "Action": [
        "ec2:CreateNetworkAclEntry",
        "ec2>DeleteNetworkAclEntry"
      ],
      "Resource": "*"
    }
  ]
}

```

Appendix C

Steps to compare a list of IP addresses against the IP addresses that Talos provided as indicators of compromise (IOCs) linked to their 2024 blog post (Cisco Talos, 2024).

1. Place all IPs from the blocklist into a TXT file
2. Download Talos' TXT list from GitHub (<https://github.com/Cisco-Talos/IOCs/blob/main/2024/04/large-scale-brute-force-activity-targeting-vpns-ssh-services-with-commonly-used-login-credentials.txt>)
 - a. Remove "ip addresses:" header
 - b. Remove usernames (below line 3932)
 - c. Save the file as `talos_ips.txt`
 - d. Remove the `[.]` so IP addresses have a consistent formatting, and save in a new file with the command:

```
sed 's/\\[\\.\\.\\]/./g' talos_ips.txt > cleaned_ips.txt
```
3. Compare the two files to see if there are IP addresses that are found in both files:

```
comm -12 <(sort cleaned_ips.txt) <(sort blocklist.txt)
```

The screenshot below shows 21 IP addresses that are common between the two files.

```
rwehe@buntuvvm:~/ip_compare$ comm -12 <(sort cleaned_ips.txt) <(sort blocklist.txt)
109.70.100.70
185.100.87.174
185.220.100.248
185.220.100.251
185.220.100.252
185.220.100.253
185.220.100.254
185.220.101.108
185.220.101.33
185.220.101.35
185.220.101.36
185.220.101.41
185.246.188.74
192.42.116.173
192.42.116.192
192.42.116.194
192.42.116.209
192.42.116.210
192.42.116.211
192.42.116.216
94.142.244.16
rwehe@buntuvvm:~/ip_compare$ comm -12 <(sort cleaned_ips.txt) <(sort blocklist.txt) | wc -l
21
```

Figure 13: IP Address in Blocklist and Talos' IOC List

Appendix D

The code described in this project can be found on GitHub at:

https://github.com/rwehe/Dynamic_IP_Blocker.