**CSCI401-Capstone Course**

**Professor Meryem Abouali**

**Cybersecurity Project Final Report**

**DDoS Attack Mitigation: Protected vs. Unprotected Servers**

**1. Objective**

The primary goal of this project was to compare effectiveness of Linux kernel-level mitigations strategies against SYN Flood attacks, a form of DoS attack (Denial of Service).

The experiment compared two Ubuntu 22.04 servers: one using the default defenses and configurations (Unprotected) and one reinforced with defenses aimed to mitigate and defend against SYN attacks (Protected).

The expected outcome would be the unprotected server to be severely debilitated while under stress, while the server with the hardened parameters would be able to continue through all the attacks under stress as if nothing is going on.

**2. Introduction**

**2.1 Overview of the Experiment**

The mechanism focused on in this project is the TCP SYN flood DoS attack, which takes advantage of the three way handshake. While default firewall parameters do help, adding further securities, such as internal hardening, provides necessary safeguards in defense.

**2.2 Problem Identification (Nature of Attack)**

The TCP Syn flood attack targets the servers memory and cpu resources. When a SYN packet is sent, the server holds that connection in memory waiting for the handshake to be completed, or in other words, the ACK portion of the handshake. The attack works by flooding the server with thousands of SYN packets, which end up filling the backlog. When the backlog is full, the server starts dropping connections, even if they are legitimate connections.

**2.3 Technical Approach**

The approach involved a comparative analysis using a controlled virtualized environment.

* **Mathematical Model:** SYN Cookies allow the server to reconstruct the connection from the ACK packets using cryptographic hash.
* **Rationale:** This allows the server to remain responsive despite the thousands of SYN-ACK packets it is receiving at once.

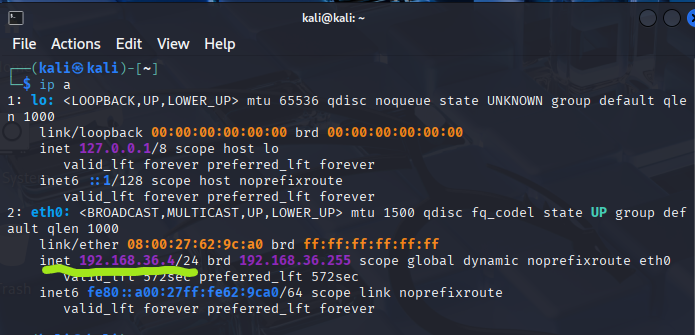
**3. Experiment**

**3.1 Network Topology, Configuration, and System Setup**

The experiment used a Host-Only network adapter in VirtualBox.

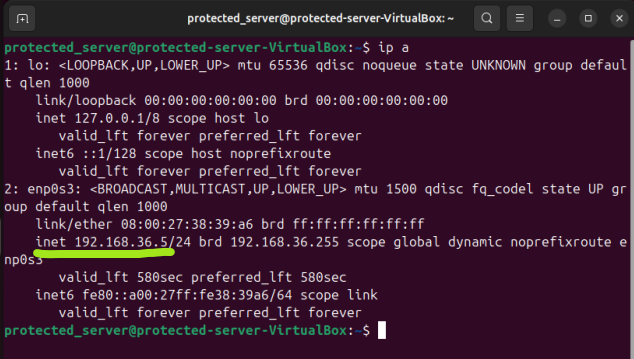
* **Attacker:** Kali Linux (IP: 192.168.36.4)

**Kali IP Address**

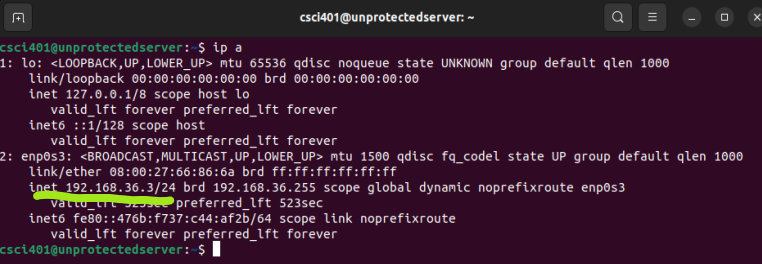


* **Protected Server:** Ubuntu 22.04 (192.168.36.5)

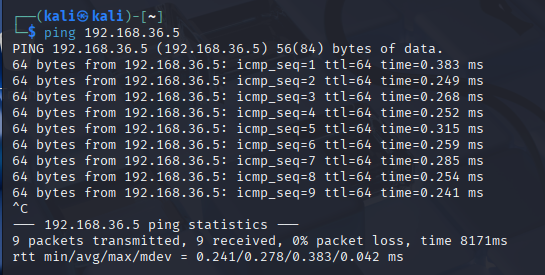
**Protected Server IP Address**

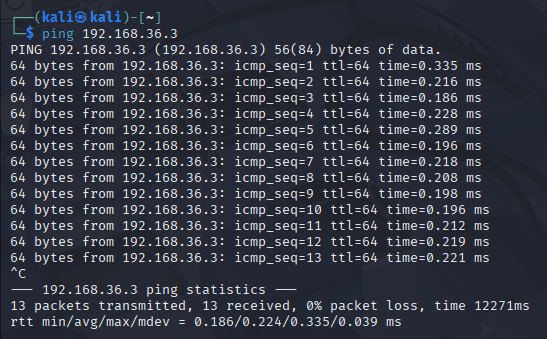


* **Unprotected Server:** Ubuntu 22.04 (192.168.36.3)

**Unprotected Server IP Address**

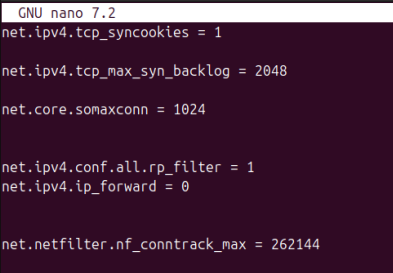
**Kali Pinging Protected Server**



**Kali Pinging Unprotected Server**

\***NOTE: The ping command (for both servers) confirms that the attacker is able to reach both servers.**

**Protected Server (sudo nano) Hardened File**

****

**syncookies=1 confirms that the defenses are on.**

**syn\_backlog=2048 allocates 2048 slots of half open connections for people trying to connect.**

**rp\_filter=1**

**conntrack\_max =262144 shows how many connections the firewall can remember at once. The larger the number, the harder it is for the attacker to crash the firewall. In DDoS attacks, the firewall usually crashed before the server.**

**Unprotected Server Defenses**

****

**syncookies=0 disables the syn-flood defenses (was originally set to 1). If it were set to 1, it would have been active.**

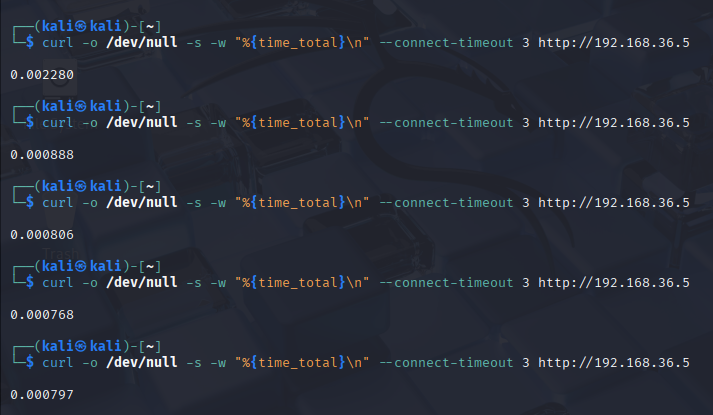
**syn\_backlog=1 allocates 1 slot(s) of half open connections for people trying to connect. In this case, there is only one connection open to a single person.**

**somaxconn=1**

**3.2 Experiment Steps**

**Step 1: Ensuring both servers have the same baseline measurements.**

* Using the **curl** command on both servers, idle times were measured.

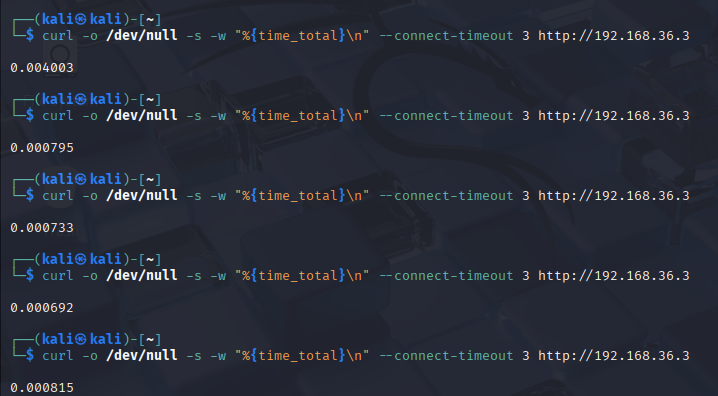
****

**Curl command line during idle time.**

**-o /dev/null doesn’t use the HTML**

**-w “%{time\_total}\n prints out the time in seconds**

**–connect-timeout times out the connection after 3 seconds pass**

****

**\*NOTE: Both screenshots showcase how long the request takes during idle time. Meaning they aren’t being subjected to an attack. Notice that both times are very similar. Meaning at start, they both have the same features.**

**Step 2: Attack Commencement**

* **Hping3** tool was used to launch the SYN flood attack, targeting port 80.
* **Command: sudo hping3 -S –flood -p 80 <target\_ip>**

****

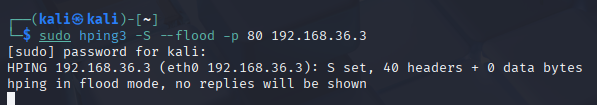
**This is the general command line for the hping3 command line.**

**-S is the SYN Flag in the TCP packet.**

**—flood lets hping know to send packets as fast as possible with replies.**

**-p 80 means to target port 80.**

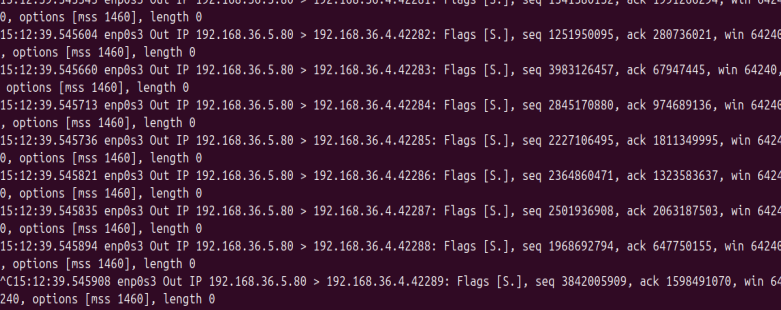
**192.168.36.5 is the target IP (Protected Server)**

****

**The same hping3 command, done on the Unprotected Server**

**Step 3: Verifying the attack was taking place**

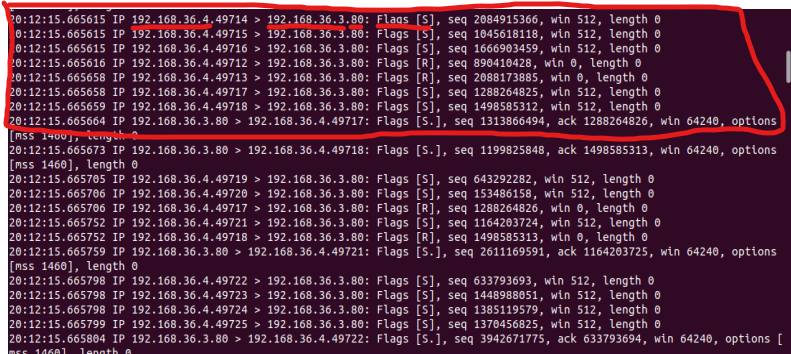
* **Tcpdump** was used to monitor/verify the incoming packet traffic.

****

**This is the tcpdump command on the protected server while the syn flood attack is going on.**

**When you look closely, you are able to see the attacker IP (.4) along with which port is being attacked (80)**

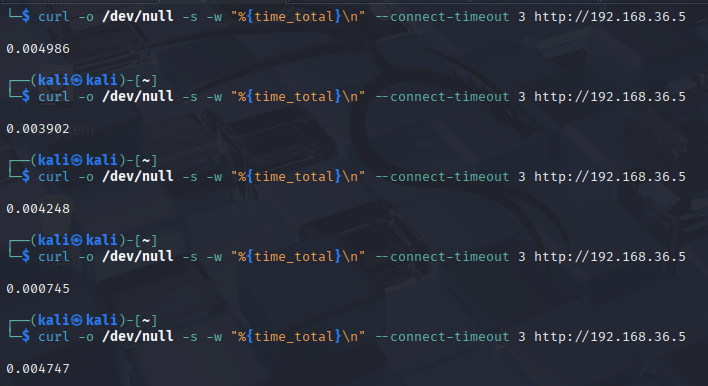
**Flags [S.] meaning it is a SYN-ACK handshake method attack**

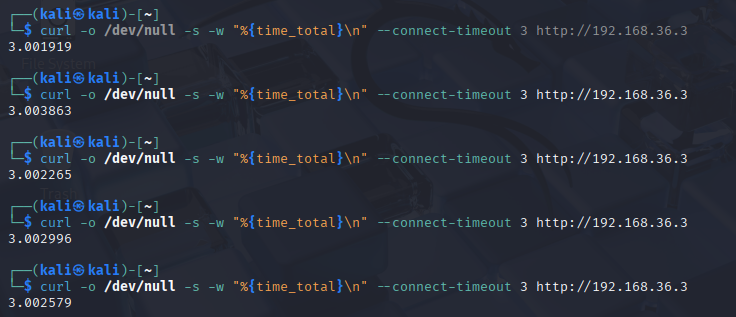
****

**Showcasing the attacker IP and the victim IP, along with the port being targeted.**

**Step 4: Measurements while servers are under stress**

* **curl** command was used to measure response times while the servers were under attack.

****

****

**\*NOTE: Both screenshots showcase the request runtime during being subjected to a SYN attack.**

**Notice that in this case, the Unprotected Server was overwhelmed from the attack and basically timed out, reaching the 3 second mark for each of the 5 requests.**

**The Protected Server on the other hand had a few seconds added to the total time, but otherwise was perfectly able to handle the excessive packets.**

**3.3 Results and Analysis**

The results showcased a massive difference in performance between both servers.

* **Baseline:** Both servers responded within 1 - 4 ms when running in idle.
* **Under Stress:** The unprotected server timed out, which indicated failure, ~+3s . The protected server showed the same response time as if it was under idle time, ~4 ms.

**Raw Data**

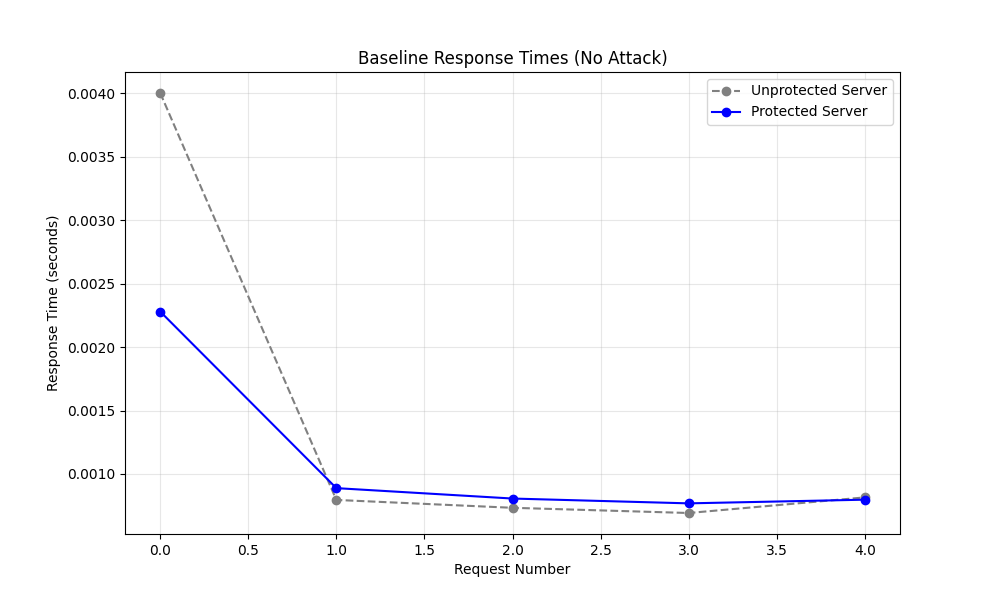
****

**The raw data used for the graphs. These were taken using the curl command line. Each time is shown in seconds.**

**If you notice on the bottom portion for the Unprotected Server during the interval, you could see that there were some difficulties during the time averages. During a SYN Flood attack on a server with no defenses against it, the request time should have gone up. But from the first 3 attempts, you could see that they were still in the milliseconds. Clearly, that shouldn’t have been the case. An unprotected server should have been able to go toe to toe with a server that had defenses against such attacks.**

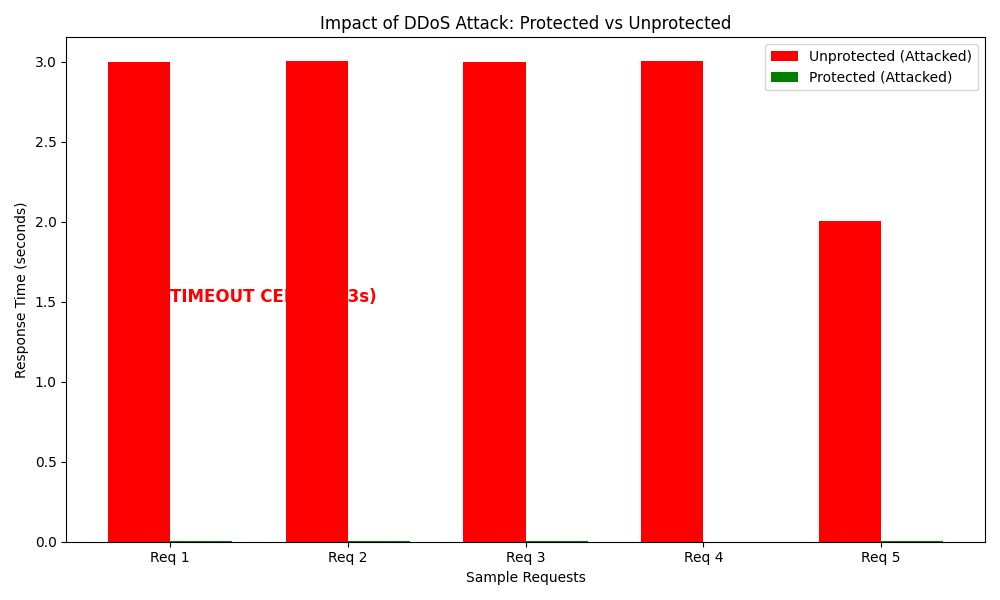
**One of the things about Ubuntu 22.04, they have defenses automatically installed, so they had to be disabled. Syncookies had to be changed from the 1 value, to the 0 value.**

**Graph 1**

****

**Showcasing that both servers are identical to each other under normal levels. Both servers are under no stress at this point, and any degradation after this point has to do with the SYN Flood attack later on.**

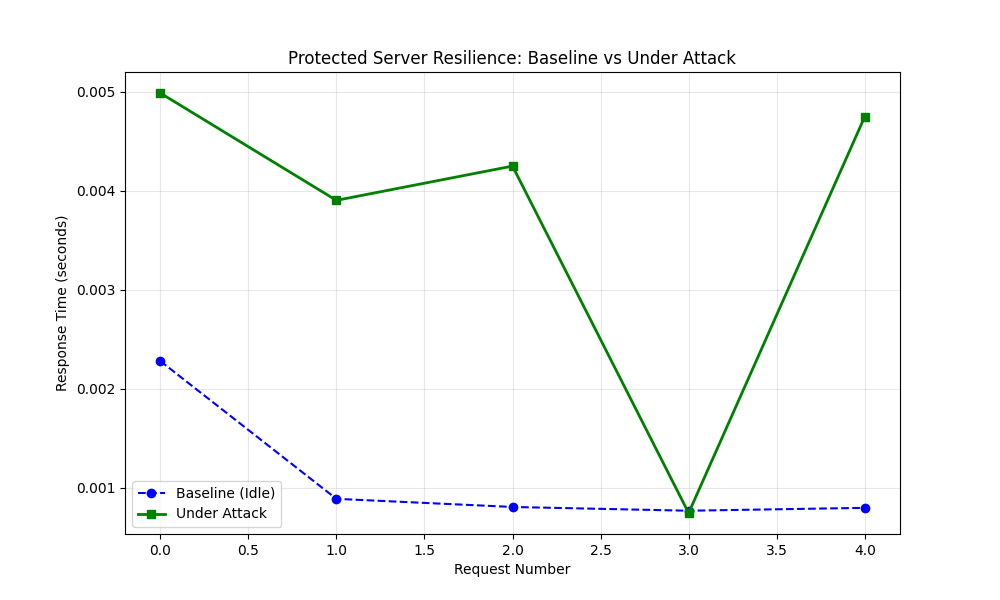
**Graph 2**

****

**This graph showcases both servers under SYN Flood attack. The red bar shows how long it takes for the unprotected server to respond to the request. Which in this case, takes roughly 3 seconds. Which is a huge drop in responsiveness difference from the one without any attacks going on. Going from roughly 1 - 4 milliseconds, to 3 seconds under stress.**

**In this graph, the protected server is not even visible, since even during stress, the server was still able to respond instantly. Showcasing how effective the SYN Flood protections were.**

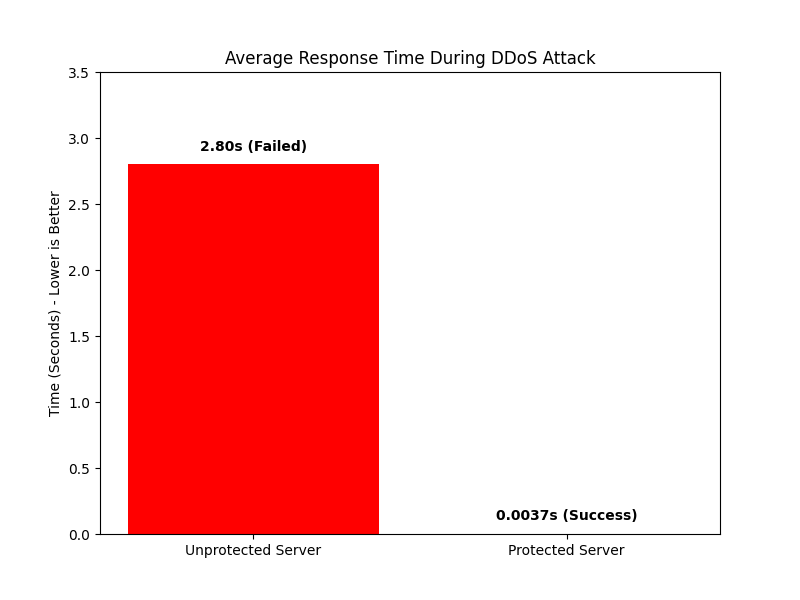
**Graph 3**

****

**This is the comparison of the Protected Server, both under attack and when it was running idly.**

**Even under stress, the responsiveness was a few millisecond difference.**

**Graph 4**

****

**The average response time after taking the 5 response times when both servers were under attack.**

**The unprotected server failed to pass the tests since it took nearly 3 seconds to respond to any of the requests. This could be considered a *timeout* on the requests.**

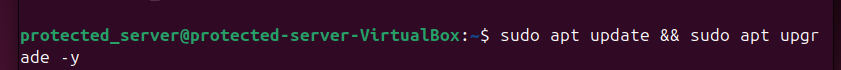
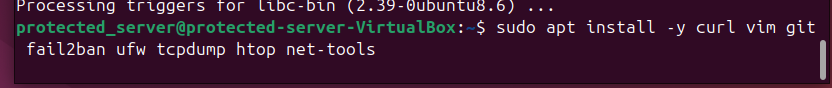
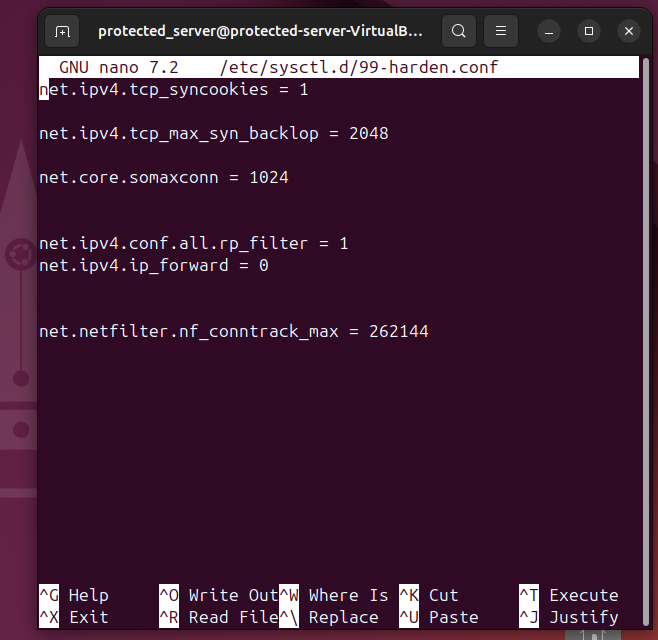
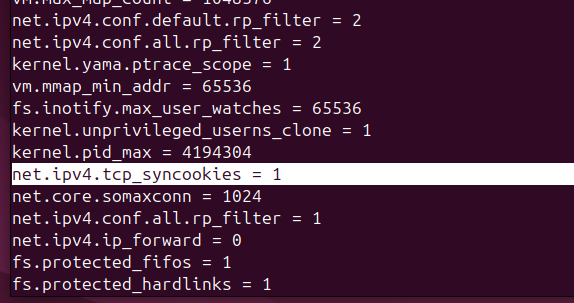
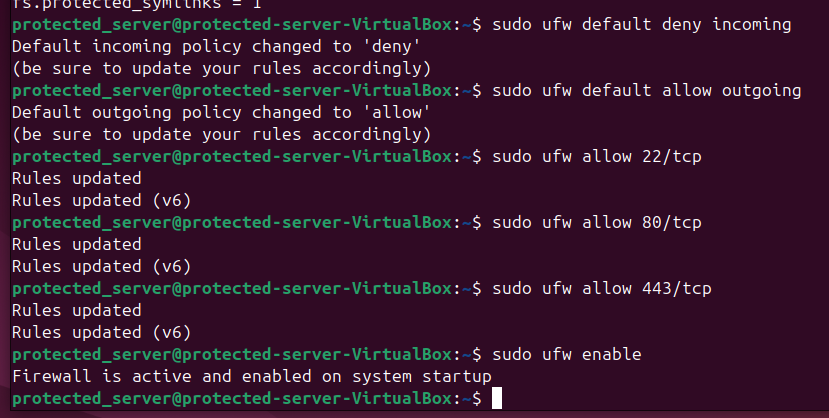
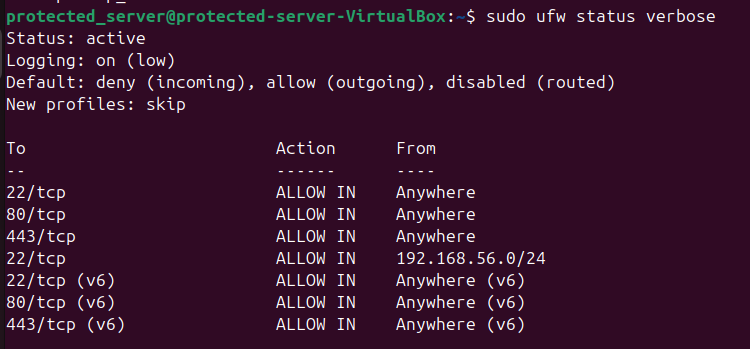
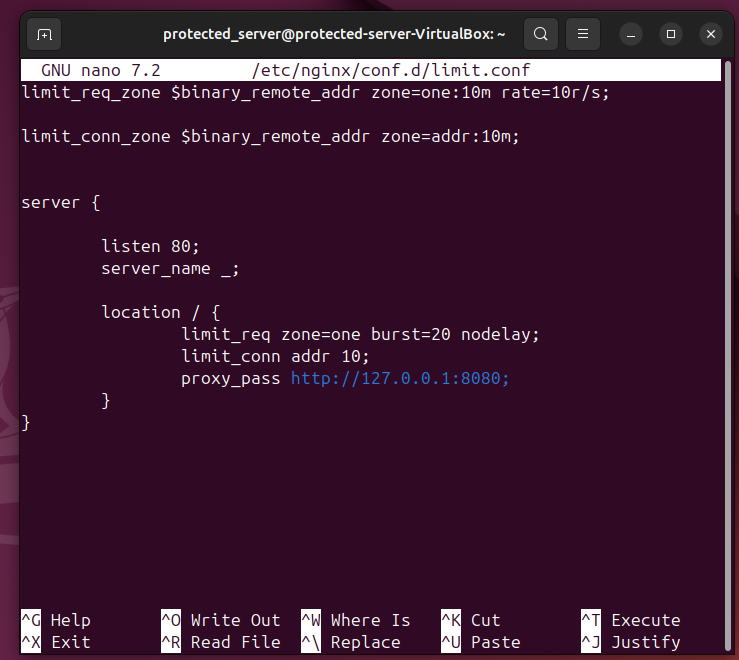
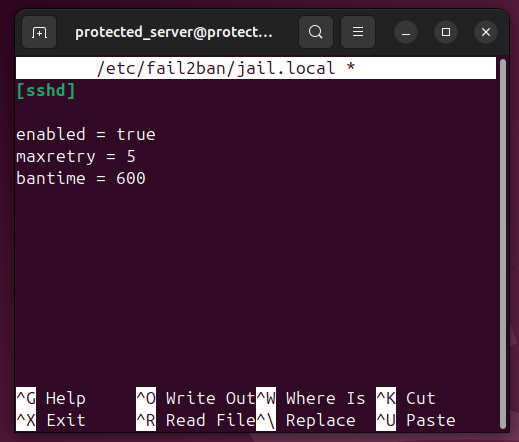
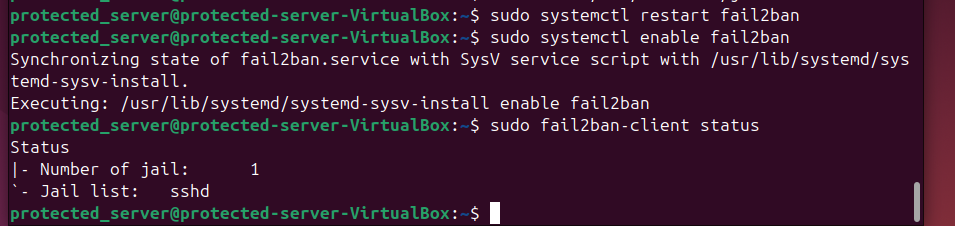
**This yet again showcases that the protections placed on the other server was able to protect it from the thousands of SYN Flood requests being sent to it.**

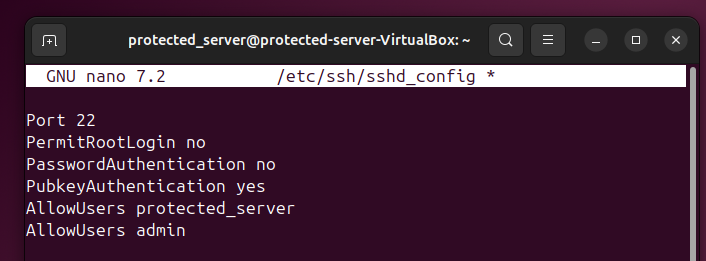
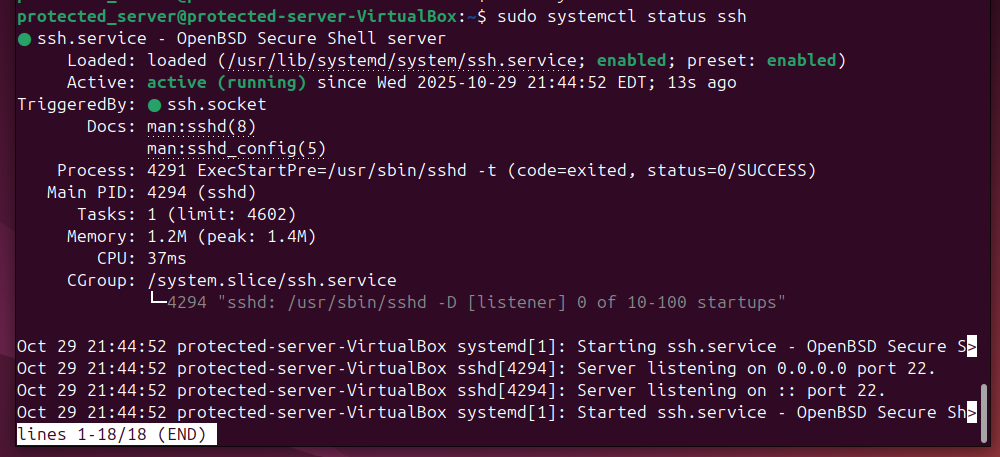
**4. Protocols and Algorithms for Prevention, Detection, and Countermeasures**

**4.1 Prevention Mechanisms**

* **Protocols:** Several network and host level protocols were enabled to prevent TCP SYN Flood attacks:
  + *TCP SYN Cookies*
    - Enabled via *net.ipv4.tcp\_syncookies = 1.*
    - This protocol modifies how the Linux kernel handles incoming TCP handshakes by avoiding allocation of connection state until the final ACK is received.
    - This prevents backlog exhaustion caused by spoofed or incomplete SYN packets
  + *Backlog and Queue Tuning*
    - *net.ipv4.tcp\_max\_syn\_backlog = 2048*
    - *net.core.somaxconn = 1024*
    - These protocol parameters increase the number of half open connections the server can temporarily withstand, slowing exhaustion during a flood.
  + *Conntrack Table Expansion*
    - This prevents the Netfilter connection-tracking system from overflowing when large quantities of incomplete TCP connections are created. (*net.netfilter.nf\_conntrack\_max = 262144)*
  + *Firewall Protocol Rules (UFW)*
    - Default deny inbound traffic
    - Only ports 22,, 80, and 443 allowed
  + *Nginx Reverse Proxy Rate-Limiting Protocols*
    - By using *limit\_req\_zone* and *limit\_conn\_zone* configurations, Nginx is able to enforce IP based rate limits and connection caps to protect upstream applications from high request bursts
* **Algorithms:**
  + *SYN Cookie Algorithm*
    - When a SYN arrives, the server encodes critical connection information into the SYN aCK sequence number instead of allocating memory
    - Only when the client responds with a valid ACK is the connection state created.
    - This ensures that incomplete SYNs don't cause resource exhaustion
    - The effectiveness of this is extremely high, since this algorithm remains the industry standard for mitigating SYN floods
  + *Fail2Ban Pattern Matching Algorithm*
    - Fail2Ban scans */var/log/auth.log* and */var/log/syslog* for repeated failed connections or abnormal patterns (like repeated SYN attempts).
    - It triggers automated firewall bans using IP tables or UFW
    - We used Fail2Ban because many SYN flood tools send repeated attempts from a small set of IP’s
    - This is effective for non-spoofed attacks and for detecting abusive SSH behaviors.
  + *Nginx Rate Limiting Algorithm*
    - A token bucket algorithm is used to regulate requests *(rate=10r/s, burst=20).)*
    - Connection limiting restricts each IP to a fixed number of open connections
    - Nginx ensures that the application layer remains responsive, even if the network stack is hit with excessive traffic
* **Implementation Details:** All prevention mechanisms were integrated at the OS level using:
  + */etc/sysctl.d/99-harden.conf* for kernel tuning (SYN cookies, backlog, conntrack).
  + UFW firewall rules applied through CLI configuration.
  + Nginx rate limiting configuration added to */etc/nginx/conf.d/limit.conf.*
  + Fail2Ban installed and configured in /*etc/fail2ban/jail.local.*
* **Challenges & Resolutions**
  + **Challenge**: Some sysctl parameters did not apply until reboot.
    - ***Resolution***: Used *sudo sysctl --system* to force immediate kernel reload.
  + **Challenge**: Nginx initially returned 502 errors due to missing backend service.
    - ***Resolution***: Set up a simple local service at *127.0.0.1:8080* to satisfy proxy requirements.
  + **Challenge**: Firewall bans interfered with internal testing.
    - ***Resolution***: Whitelisted team network range *192.168.56.0/24.*

**4.2 Detection Mechanisms**

* **Monitoring Tools:** The following tools were used to observe system state during the attack:
  + *Tcpdump:*
    - Captured live packets to .*pcap* files for later analysis
    - Helped visualize large quantities of incoming SYN packets and half open connections
  + *Curl*:
    - Used to measure server response times before and during the attack.
    - Helps in detecting performance degradation, timeout, or increased latency.
    - Useful in identifying when a DoS condition is impacting service responsiveness.
  + *Ping:*
    - A simple tool used to verify basic network reachability.
    - Confirmed that the attacker can reach both servers.
    - Helped in detecting connectivity issues that resulted from network overload or attack traffic.
* **Detection Algorithms:**
  + *SYN Cookies:*
    - Cryptographic technique that embeds connection information into the SYN ACK response.
    - Prevents the server from storing half open connections, in result- makes the backlog harder to exhaust.
    - Detects malicious traffic when ACK replies do not match valid cookie values.
    - Highly effective at identifying and mitigating SYN flood attacks.
  + *Backlog Monitoring (syn\_backlog logic):*
    - Tracks the number of half open TCP connections waiting in the backlog queue.
    - Sudden or sustained spikes indicate a potential SYN flood.
    - Helps trigger protective behaviors like enabling SYN cookies.
    - Effective as an early detection signal of abnormal connection rates.
  + *Reverse Path Filtering (rp\_filter):*
    - Validates that incoming packets arrived from an expected source IP address.
    - Detects spoofed IP packets that are commonly used in DoS attacks.
    - Helps reduce the effectiveness of attacks that rely on forged source addresses
    - Highly effective in identifying suspicious or illegitimate traffic.
  + *Connection Tracking (conntrack\_max behavior):*
    - Tracks all active and recent network connections maintained by the firewall.
    - Detects anomalies when connection counts rise abnormally fast or approach max capacity.
    - Helps identify volumetric DoS attempts before the server becomes unresponsive
    - Useful in spotting patterns associated with automated attack tools.
* **Implementation Process:**
* *System Setup and Tool Installation*
  + First step was preparing the server with the necessary monitoring and security utilities
    - Purpose: to ensure the system is fully patched, (by installing monitoring tools like tcpdump, curl, and security frameworks like UFW, and fail2ban)
    - 
    - 
* *Kernel Level Detection and Hardening Configuration*
  + Detection algorithms (SYN cookies, backlog checks, reverse-path filtering, conntrack monitoring) were implemented via a hardened sysctl file.
    - 
  + These settings improve detection accuracy by:
    - *SYN cookie*s: prevents backlog exhaustion and detects spoofed ACKs.
    - *Backlog tuning*: reveals abnormal spikes in half-open connections.
    - *rp\_filter*: drops spoofed packets used in SYN floods.
    - *Conntrack limits:* helps detect rapid connection spikes that indicate DoS attempts.
    - 
  + Applied it using: sudo sysctl - -system
* *Firewall Configuration (UFW):* 
  + UFW was configured to restrict network exposure and provide basic packet filtering.
    - 
    - This improves detection by:
      * Reducing attack surface.
      * Allowing easier identification of suspicious traffic.
      * Providing logs of connection attempts.
    - 
    - Status checked by: ***Sudo ufw status verbose***
* *Nginx Rate Limiting and Application Level Detection*
  + Nginx was used to throttle excessive requests, adding application-layer detection against burst attacks.
    - ******
    - How this improves detection:
      * Nginx logs show IPs exceeding request thresholds.
      * Rate limiting exposes abnormal surges in traffic typical of SYN floods or HTTP floods.
* *Fail2Ban Setup (Automatic Detection & Response)*
  + Fail2Ban was implemented to scan logs for abusive behavior and ban IPs based on detection rules.
    - 
    - Detection improvements:
      * Automatically identifies login brute force attempts.
      * Reduces false positives by allowing multiple retry attempts before banning.
      * Uses log based detection, which supplements network-level monitoring.
  + ******
  + Verification that it is active using: *sudo fail2ban-client status*

* *Additional SSH Hardening*
  + SSH protection was increased by disabling root login:
    - 
    - 

**4.3 Countermeasures**

* **Immediate Actions:**
  + **Stateless Handshake:** 
    - The kernel automatically dropped the connection from memory. If the attacker did not respond with the correct ACK packet, then the connection would be discarded.
* **Long-Term Solutions:**
  + **Fail2Ban:**
    - Having Fail2Ban update firewall rules and ban any IPs that constantly fail the handshake
    - Having Fail2Ban configured to monitor activities/logs.

**5. Conclusion**

This project demonstrates how by implementing kernel hardening, the Protected Server was able to maintain near instantaneous response times under the same conditions as the Unprotected Server, which in that case, it timed out. Which shows that default Linux Server configurations are very susceptible to SYN Flood attacks. The results validate that SYN cookies, in particular, are a critical mitigation tool. By eliminating the need to store half open connections in memory until the handshake is fully completed, the server avoided backlog exhaustion and remained responsive, even under sustained attack traffic. When combined with firewall rules, rate limiting, and automated banning, the system benefited from a layered defense strategy that addresses both network level and application level threats. Overall, this experiment confirmed that relying on default Linux configurations is insufficient for environments exposed to untrusted networks. Properly tuned kernel parameters can dramatically improve resilience against denial of service attacks, even without specialized hardware or external DDoS mitigation services.

**6. Future Recommendations**

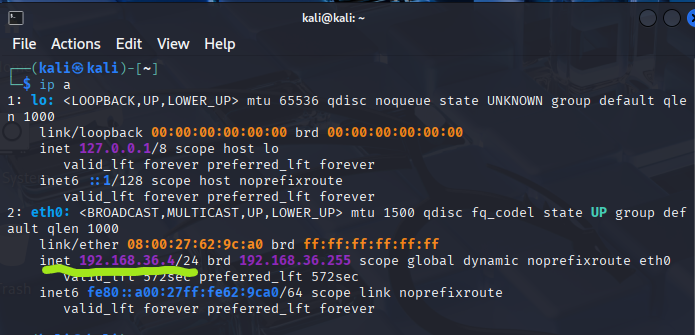
While the experiment was successful and demonstrated the effectiveness of SYN flood mitigation techniques, several improvements and extensions could be explored in future work to expand its test range. To test the extensiveness of the backlog and the hardened parameters, there should be multiple DDoS attacks (multiple VMs) attacking at the same time. Real world DDoS attacks are typically launched from botnets consisting of hundreds or thousands of compromised devices so simulating multiple attacking virtual machines would provide a more accurate assessment of how backlog tuning and SYN cookies behave under higher traffic volumes and more diverse source IPs.

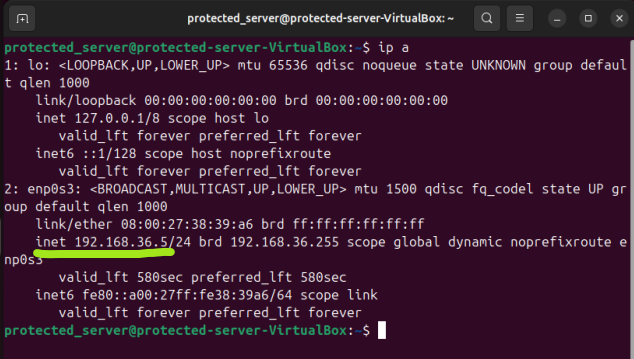
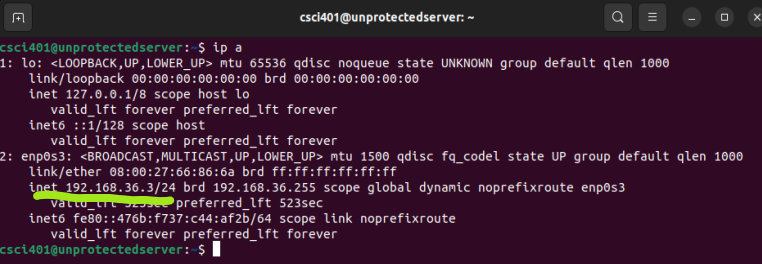
Along with TCP SYN floods, additional attack factors should be tested. These could include UDP floods, ICMP floods, and HTTP GET/POST floods. Evaluating how the same defense configurations perform against different attack types would offer a more comprehensive view of the servers overall DoS resilience

**7. References**

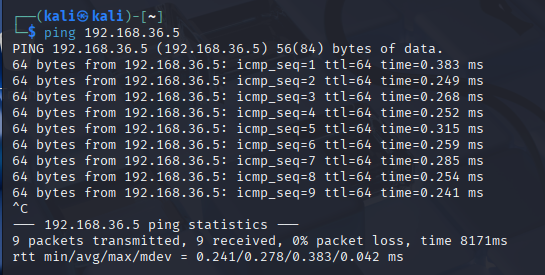
* Amazon Web Services. (n.d.). AWS best practices for DDoS resiliency.<https://docs.aws.amazon.com/whitepapers/latest/aws-best-practices-ddos-resiliency/aws-best-practices-ddos-resiliency.html>
* A comprehensive survey on DDoS attacks detection & mitigation in SDN-IoT network. (2024). Internet of Things and Cyber-Physical Systems, 7, Article 100125.<https://www.sciencedirect.com/science/article/pii/S2772671124001256>
* Belenky, A., & Ansari, N. (2006). RFC 4732: Internet denial-of-service considerations. Internet Engineering Task Force.<https://dl.acm.org/doi/pdf/10.17487/RFC4732>
* Cloudflare. (2025). DDoS threat report: 2025 Q2.<https://blog.cloudflare.com/ddos-threat-report-for-2025-q2/>
* Cloudflare. (n.d.). What is a DDoS attack?<https://www.cloudflare.com/learning/ddos/what-is-a-ddos-attack/>
* Cyberwar.nl. (n.d.). Hping3 cheat sheet v1.0.<https://cyberwar.nl/d/cheatsheets/hping3_cheatsheet_v1.0-ENG.pdf>
* DigitalOcean. (n.d.). How Fail2Ban works to protect services on a Linux server.<https://www.digitalocean.com/community/tutorials/how-fail2ban-works-to-protect-services-on-a-linux-server>
* DigitalOcean. (n.d.). UFW essentials: Common firewall rules and commands.<https://www.digitalocean.com/community/tutorials/ufw-essentials-common-firewall-rules-and-commands>
* Double Octopus. (n.d.). Stateless authentication.<https://doubleoctopus.com/security-wiki/network-architecture/stateless-authentication/>
* GeeksforGeeks. (n.d.). curl command in Linux with examples.<https://www.geeksforgeeks.org/linux-unix/curl-command-in-linux-with-examples/>
* Kali Linux. (n.d.). hping3.<https://www.kali.org/tools/hping3/>
* OpenSource.com. (2018). Introduction to tcpdump.<https://opensource.com/article/18/10/introduction-tcpdump>
* **Basic SetUp**

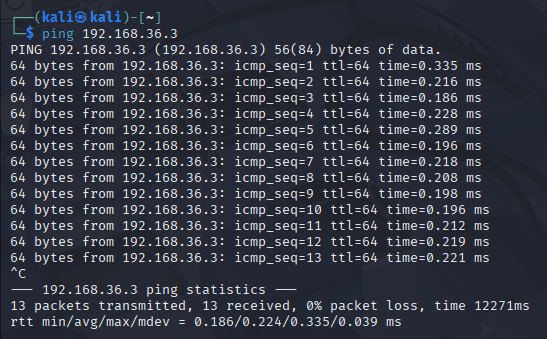
**Kali IP Address**



**Protected Server IP Address****Unprotected Server IP Address**

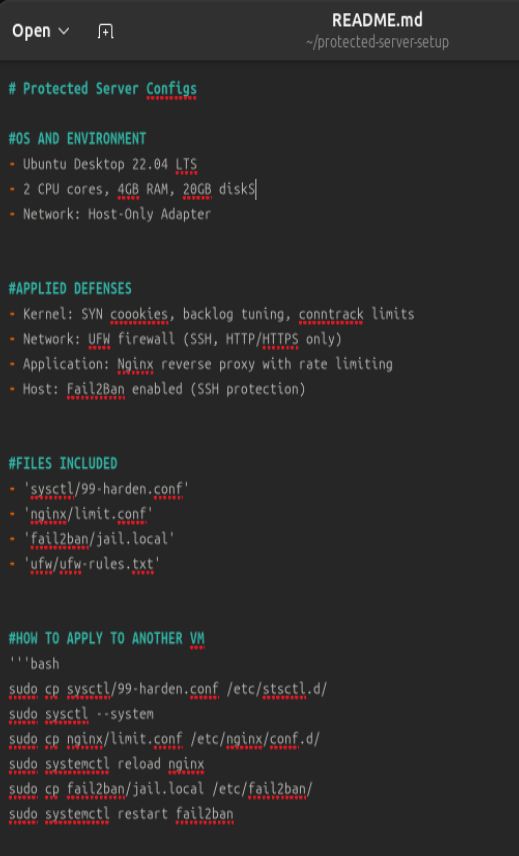
**Kali Pinging Protected Server**



**Kali Pinging Unprotected Server**

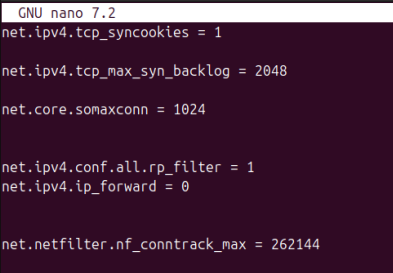
* **Protected Server Setup**

**Protected Server Protections (Hardened)**



**README file showing all the protections applied to the server.**

**Protected Server (sudo nano) Hardened File**

****

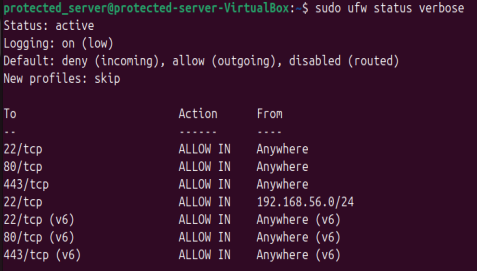
**syncookies=1 confirms that the defenses are on.**

**syn\_backlog=2048 allocates 2048 slots of half open connections for people trying to connect.**

**rp\_filter=1**

**conntrack\_max =262144 shows how many connections the firewall can remember at once. The larger the number, the harder it is for the attacker to crash the firewall. In DDoS attacks, the firewall usually crashed before the server.**

**UFW Setting**

****

**To make sure that the FireWall wasn’t stopping the attacks from getting through. As seen, on port 80, it is allowing all the packets to go through.**

**\*NOTE: nginx was installed on the side. (no screenshots of the install)**

* **Unprotected Server Setup**

**Unprotected Server Defenses**

****

**syncookies=0 disables the syn-flood defenses (was originally set to 1). If it were set to 1, it would have been active.**

**syn\_backlog=1 allocates 1 slot(s) of half open connections for people trying to connect. In this case, there is only one connection open to a single person.**

**somaxconn=1**

**\*NOTE: nginx was installed on the side. (no screenshots of the install)**

* **Attack Phase**

****

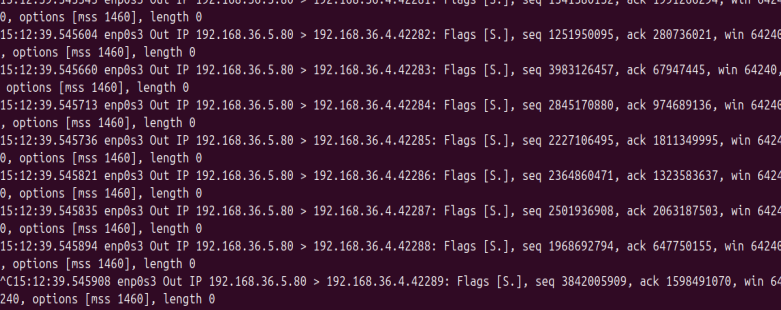
**This is the general command line for the hping3 command line.**

**-S is the SYN Flag in the TCP packet.**

**—flood lets hping know to send packets as fast as possible with replies.**

**-p 80 means to target port 80.**

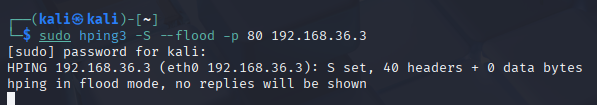
**192.168.36.5 is the target IP (Protected Server)**

****

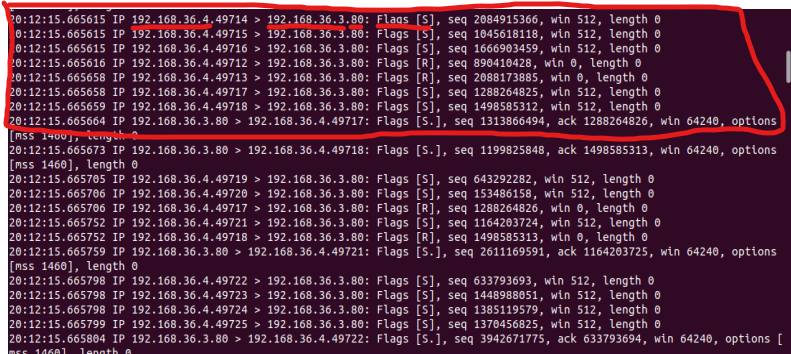
**This is the tcpdump command on the protected server while the syn flood attack is going on.**

**When you look closely, you are able to see the attacker IP (.4) along with which port is being attacked (80)**

**Flags [S.] meaning it is a SYN-ACK handshake method attack**

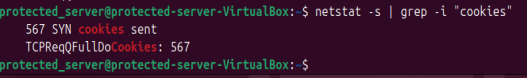
****

**The same hping3 command, done on the Unprotected Server**

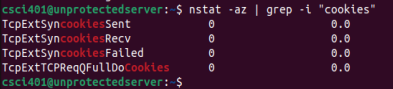
****

**Showcasing the attacker IP and the victim IP, along with the port being targeted.**

* **After-Action Reports**

****

The 567 cookies sent shows that the defenses activated for a short while while the SYN Flood attack was going on. It sent 567 SYN cookies to try and catch up with the attack. The sent cookies stopped at 567 since the server was quick and efficient enough to withstand the traffic without constantly needing cookies.

****

**These are the cookies that the unprotected server sent.**

**As you can see, there were none sent. Since the defenses were turned off, even when the server was under stress, it never sent cookies, which led to it being overwhelmed and crashing or it being timed out.**

**That will be showcased later on.**

* **Results**

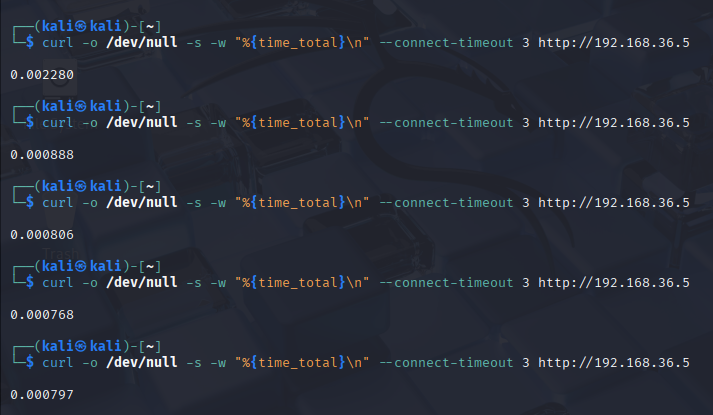
**Raw Data**

****

**The raw data used for the graphs. These were taken using the curl command line. Each time is shown in seconds.**

**If you notice on the bottom portion for the Unprotected Server during the interval, you could see that there were some difficulties during the time averages. During a SYN Flood attack on a server with no defenses against it, the request time should have gone up. But from the first 3 attempts, you could see that they were still in the milliseconds. Clearly, that shouldn’t have been the case. An unprotected server should have been able to go toe to toe with a server that had defenses against such attacks.**

**One of the things about Ubuntu 22.04, they have defenses automatically installed, so they had to be disabled. Syncookies had to be changed from the 1 value, to the 0 value.**

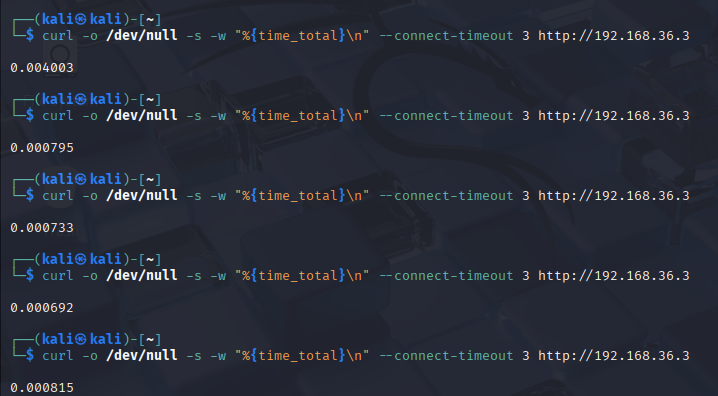
****

**Curl command line during idle time.**

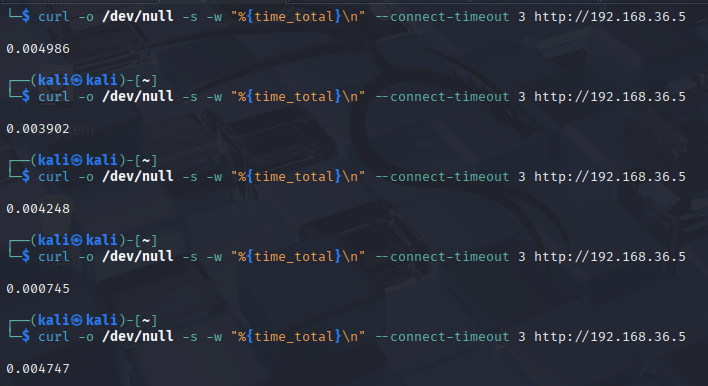
**-o /dev/null doesn’t use the HTML**

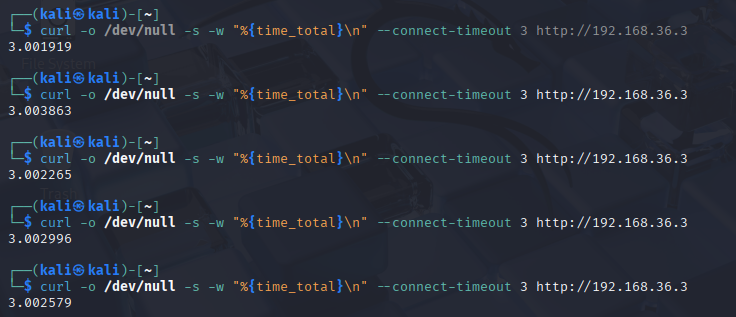
**-w “%{time\_total}\n prints out the time in seconds**

**–connect-timeout times out the connection after 3 seconds pass**

****

**\*NOTE: Both screenshots showcase how long the request takes during idle time. Meaning they aren’t being subjected to an attack. Notice that both times are very similar. Meaning at start, they both have the same features.**

****

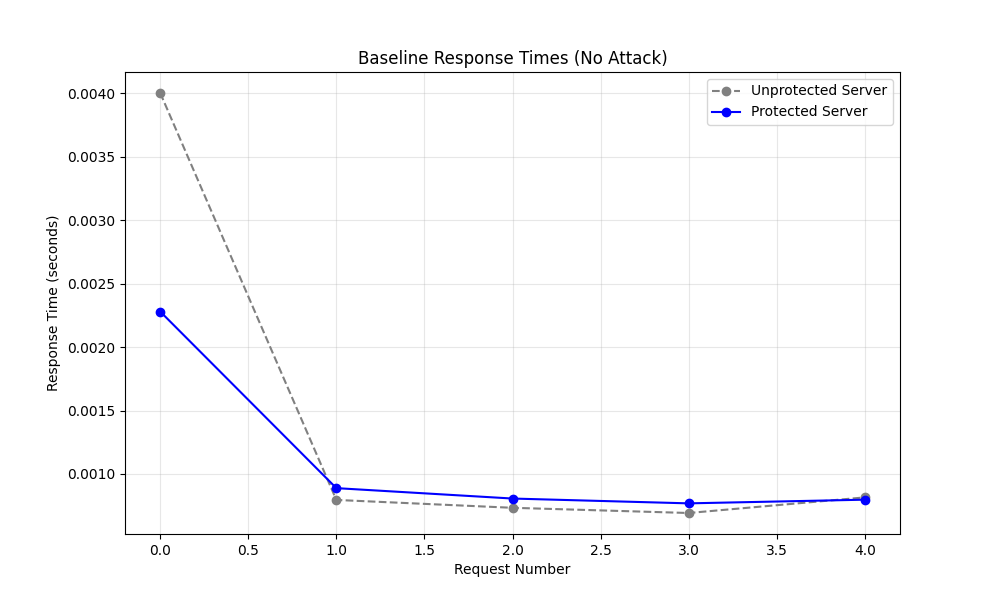
****

**\*NOTE: Both screenshots showcase the request runtime during being subjected to a SYN attack.**

**Notice that in this case, the Unprotected Server was overwhelmed from the attack and basically timed out, reaching the 3 second mark for each of the 5 requests.**

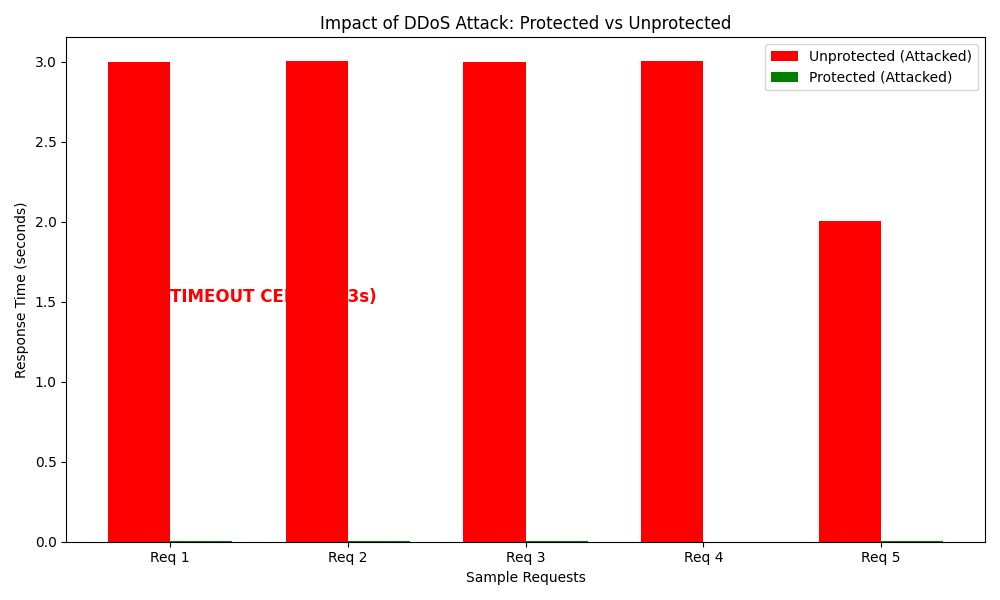
**The Protected Server on the other hand had a few seconds added to the total time, but otherwise was perfectly able to handle the excessive packets.**

**Graph 1**

****

**Showcasing that both servers are identical to each other under normal levels. Both servers are under no stress at this point, and any degradation after this point has to do with the SYN Flood attack later on.**

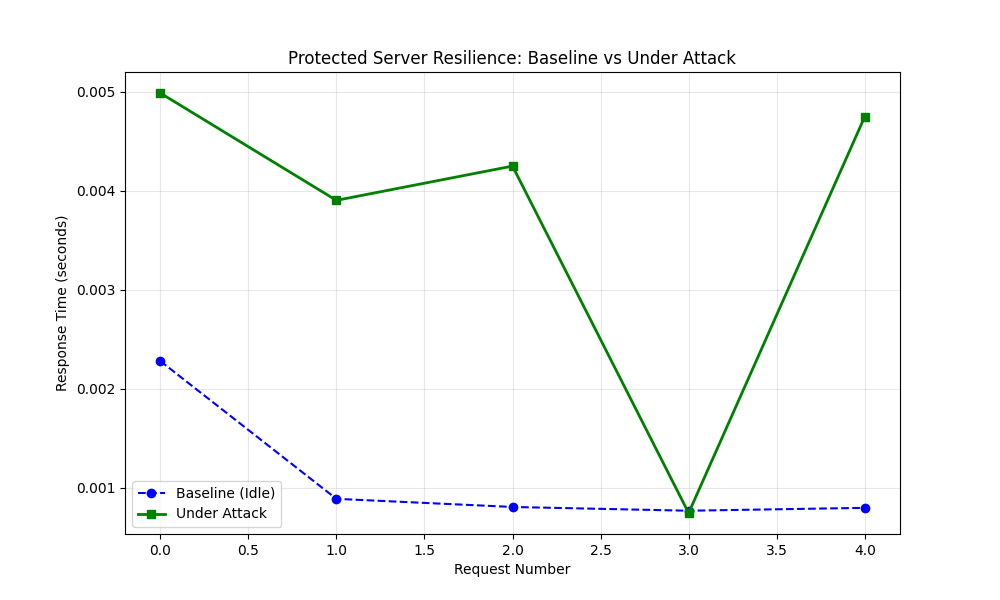
**Graph 2**

****

**This graph showcases both servers under SYN Flood attack. The red bar shows how long it takes for the unprotected server to respond to the request. Which in this case, takes roughly 3 seconds. Which is a huge drop in responsiveness difference from the one without any attacks going on. Going from roughly 1 - 4 milliseconds, to 3 seconds under stress.**

**In this graph, the protected server is not even visible, since even during stress, the server was still able to respond instantly. Showcasing how effective the SYN Flood protections were.**

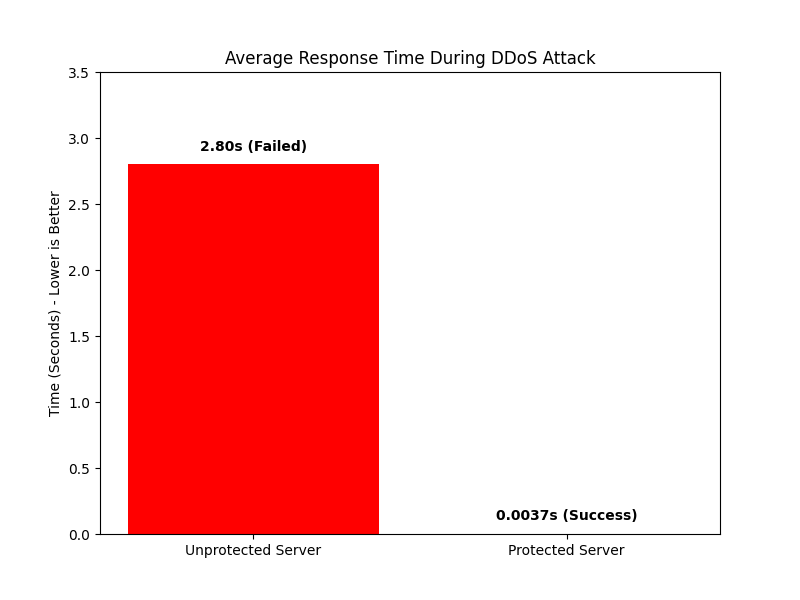
**Graph 3**

****

**This is the comparison of the Protected Server, both under attack and when it was running idly.**

**Even under stress, the responsiveness was a few millisecond difference.**

**Graph 4**

****

**The average response time after taking the 5 response times when both servers were under attack.**

**The unprotected server failed to pass the tests since it took nearly 3 seconds to respond to any of the requests. This could be considered a *timeout* on the requests.**

**This yet again showcases that the protections placed on the other server was able to protect it from the thousands of SYN Flood requests being sent to it.**