Seneca College

Marc Haye

SPR500NBB

Easton Soares – 108851213

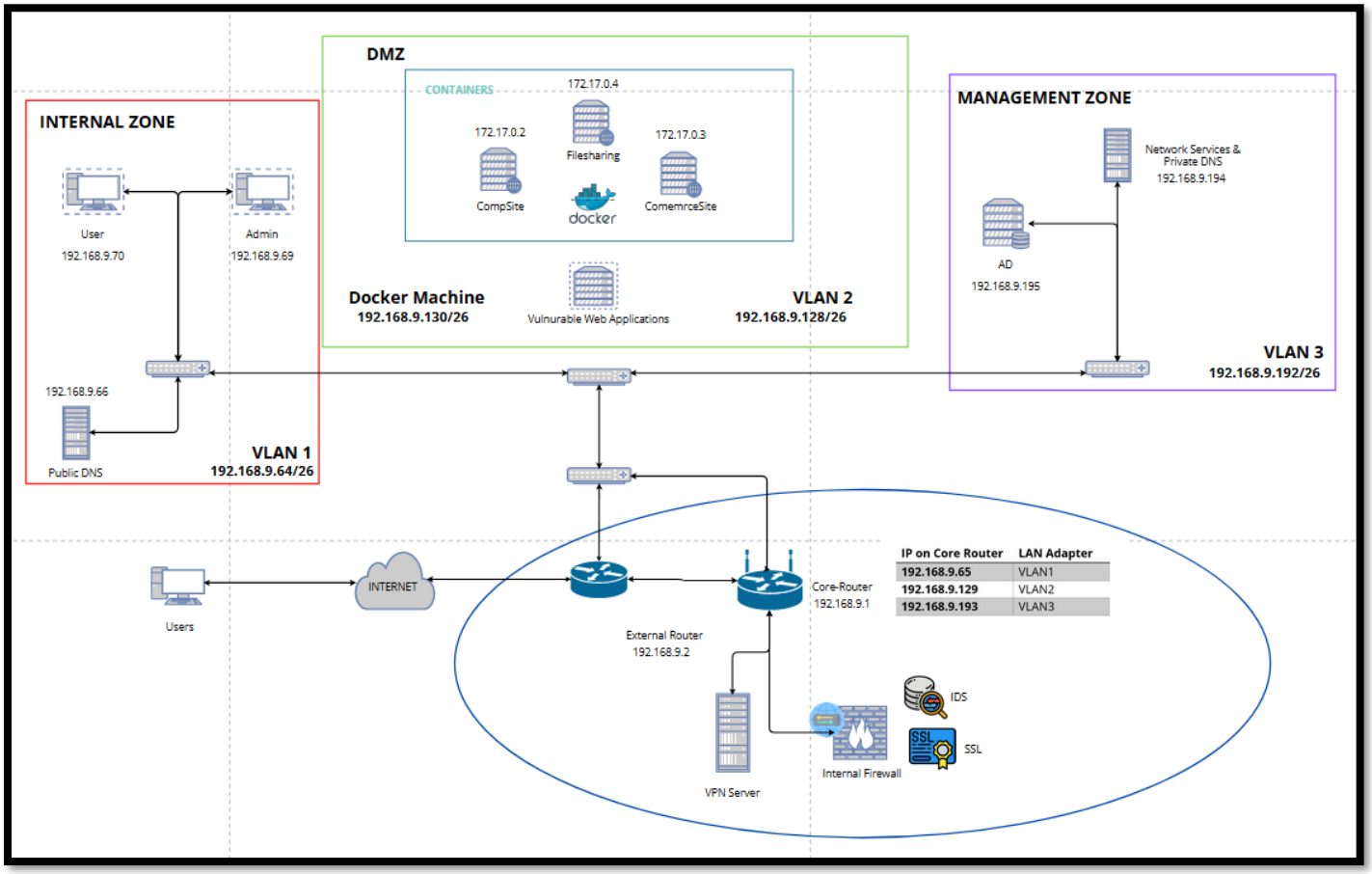
Jaskaran Sohal - 150343218

Group 8 – Professional final low-level design and functional testing report

Table of Contents

Table of Figures

Diagram of Final Implementation



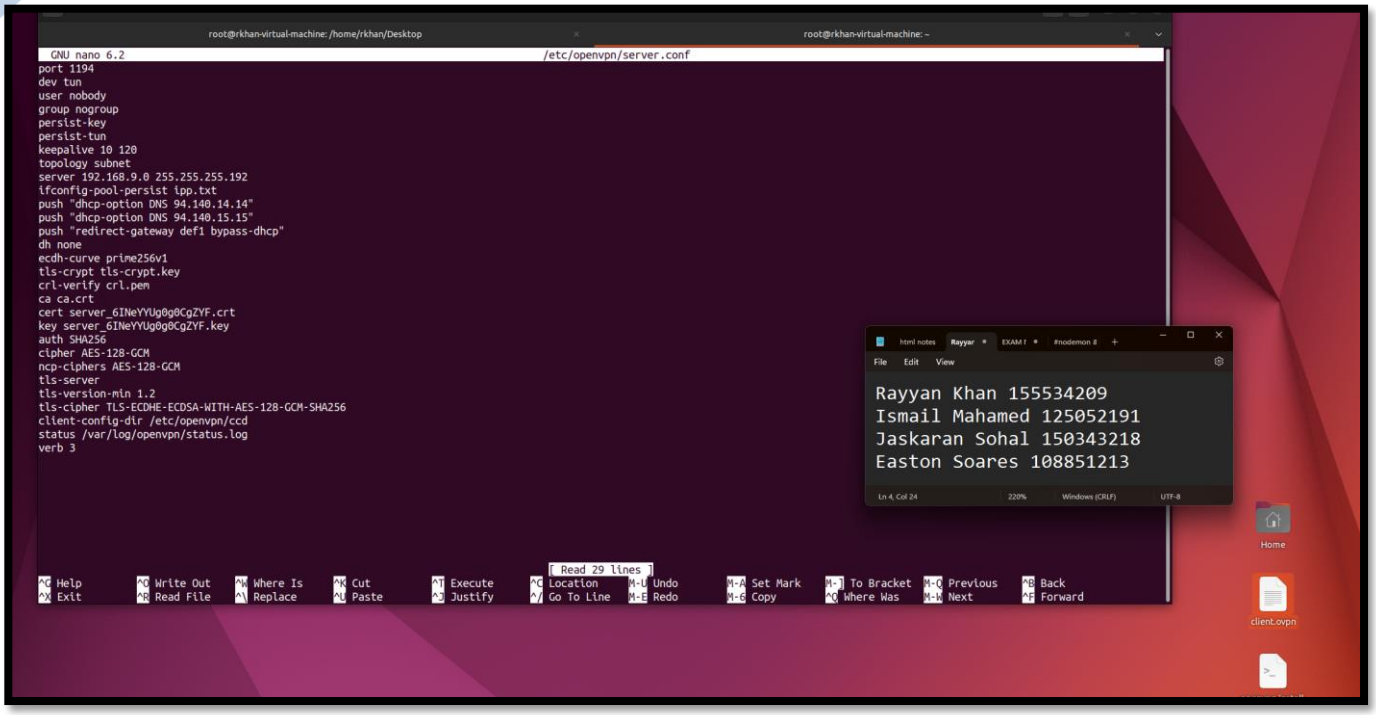


Figure 1: OpenVPN Server Configuration

We're defining the user as 'nobody' and the group as 'nogroup', which is a security measure to ensure that the VPN processes don't have too many permissions. The port is set to 1194, which is the default for OpenVPN. We're using UDP as the protocol because it's faster. The server IP is in the 10.8.0.0 network range, and we're setting up a tun device for routing the VPN traffic.

We've configured the server with a pool of IP addresses ranging from 10.8.0.2 to 10.8.0.253, so we can support many clients without address conflicts. Pushing the 'redirect-gateway' option means we're going to route all client traffic through the VPN, and we're also pushing DNS settings so clients will use our specified DNS servers. We've specified the keys and certificates paths for the ca.crt, server.crt, and server.key, ensuring our connection is secure. We also have a dh.pem file for the Diffie-Hellman key exchange, which is crucial for setting up a secure channel. Lastly, we've enabled logging by specifying the log file at /var/log/openvpn/status.log, which will help us keep track of the VPN's status and troubleshoot any issues that come up.

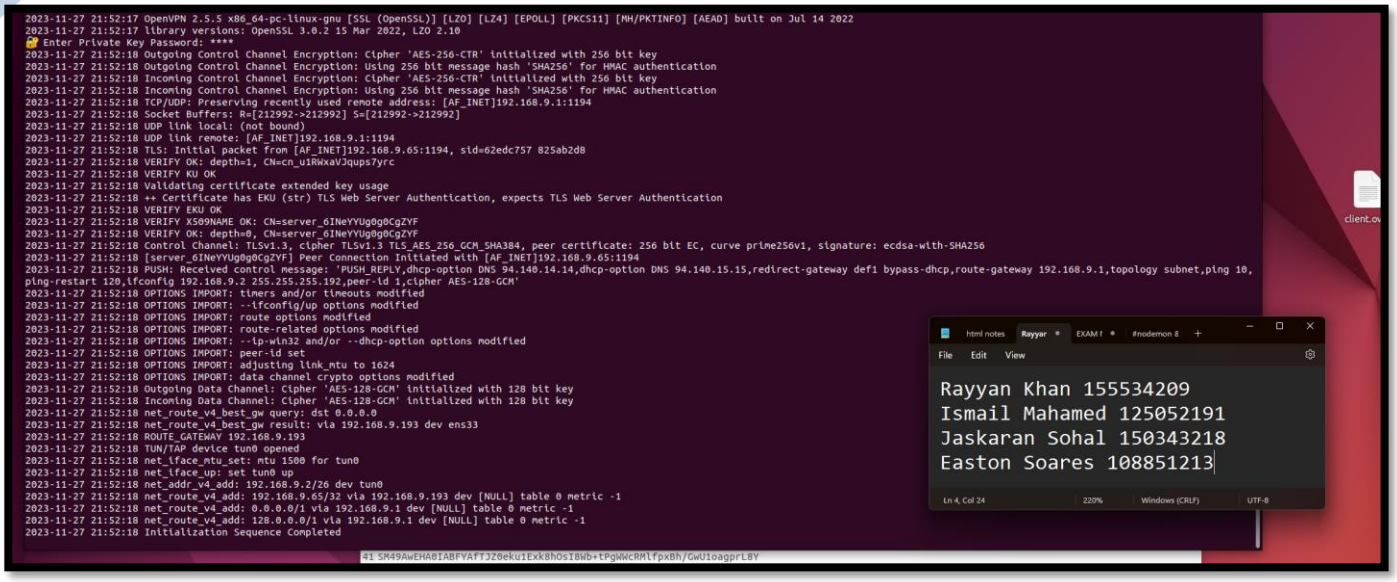


Figure 2: OpenVPN Client Access

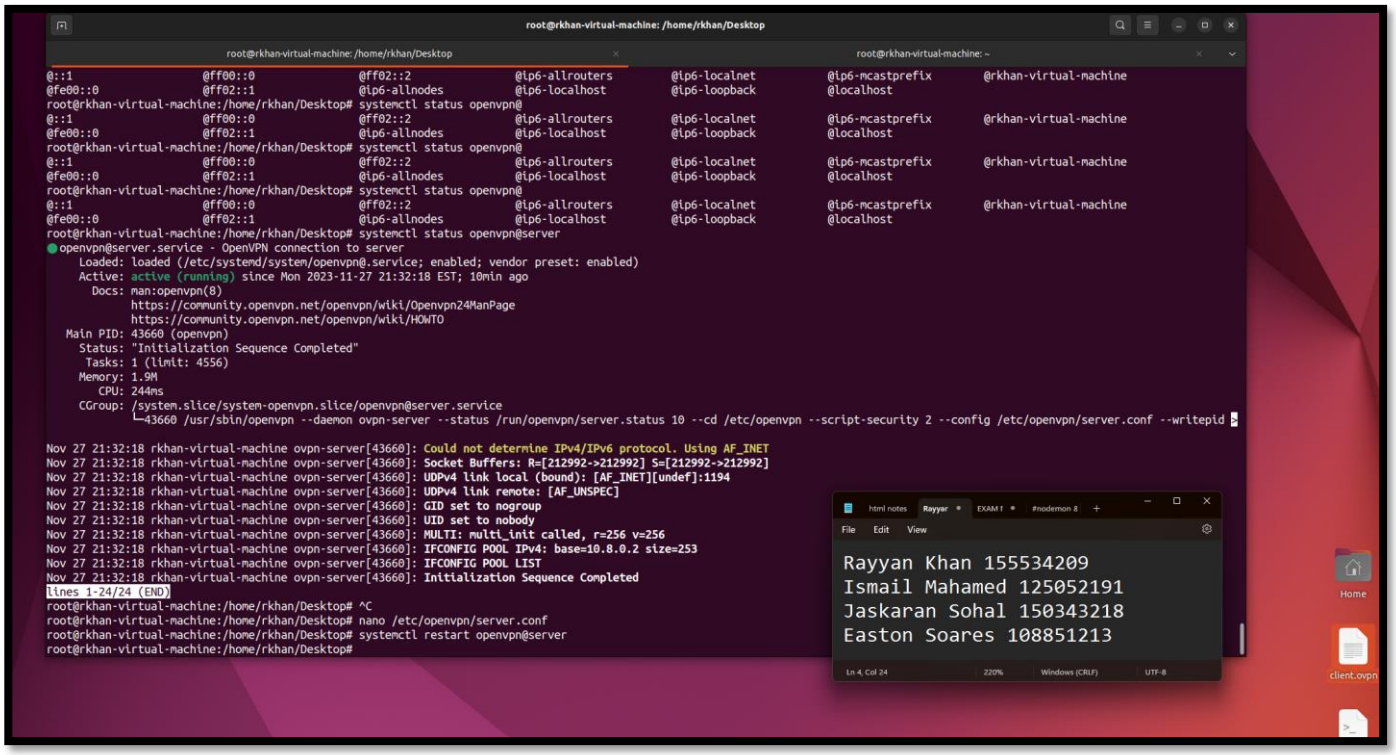


Figure 3: OpenVPN Server Status Running

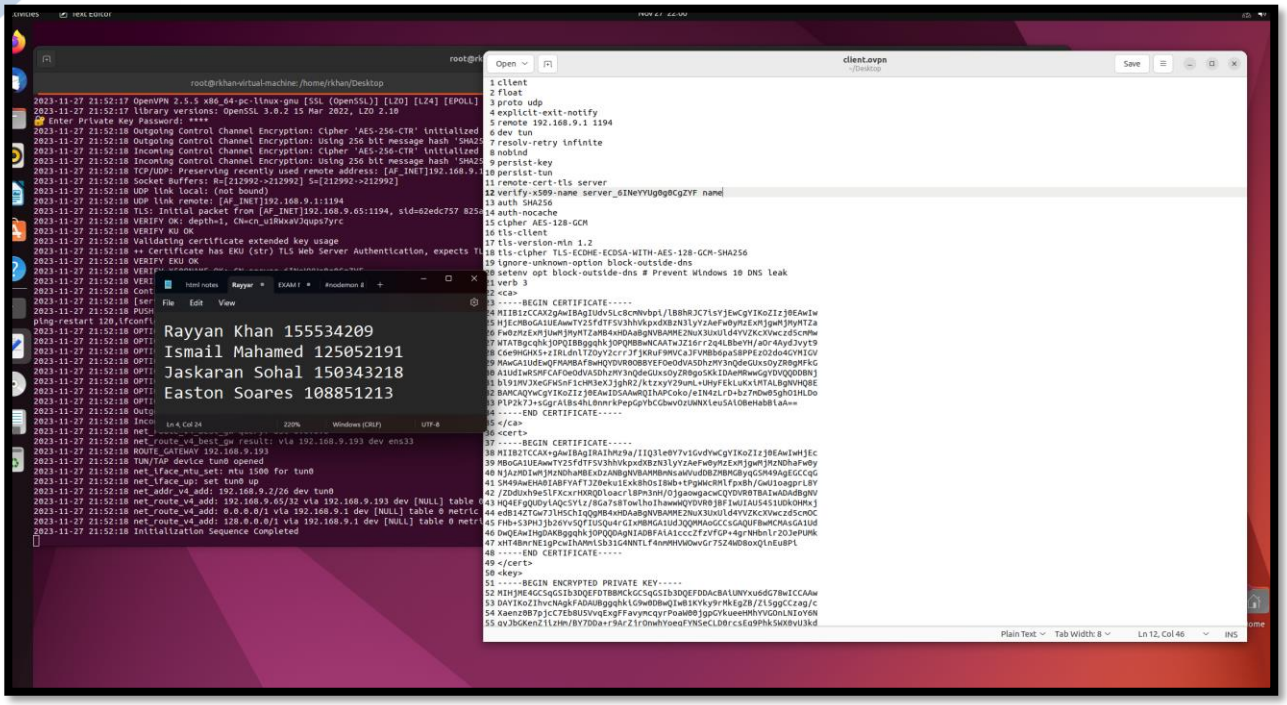


Figure 4: Client.OPVN

This is the configuration file that each of our clients will use to connect to our VPN server. Firstly, you'll notice the `client` directive at the top, which specifies that this configuration is for a client. We've set the connection protocol to UDP with the `proto udp` line because it's faster for our needs.

We've set our VPN server's address with the `remote` line – it's not shown here for security reasons, but that's where you'd input the server's IP address or domain name, followed by the port number, which is the standard `1194`. The `dev tun` line tells our client to use a tun device to create a secure point-to-point connection. We've specified `nobind` so that the client's port number will dynamically change, making it more flexible and easier to work behind different NATs without needing a static port.

For security, we've set `persist-key` and `persist-tun` to ensure that our key and tunnel stay active even if the connection drops momentarily. This helps us maintain a stable connection without reinitializing the connection entirely. The `verify-x509-name` line is a security measure to check the name on the server certificate, making sure we're connecting to the right server. The name after `server` should match the `Common Name` on the server's certificate. We're using strong encryption here, as you can see with `cipher AES-128-CBC` and `auth SHA256`. This means we're using AES 128-bit encryption for our data channel and SHA256 for our control channel, ensuring that our data is secure in transit.

The `tls-client` and `tls-version-min` directives are part of our transport layer security settings, which are important for preventing certain types of attacks and ensuring compatibility with our server's TLS version. Now, these `remote-cert-tls server` and `key-direction` lines are more about TLS authentication, ensuring we're connecting to a legitimate server and setting the direction for the TLS auth key. The `block-outside-dns` is a Windows-specific directive to prevent DNS leaks which could expose our real IP address when we're connected to the VPN. Finally, you'll see the `<ca>`, `<cert>`, and `<key>` tags where the actual certificate authority certificate, client certificate, and client private key are embedded directly into the file. This simplifies setup for our users because they won't need to manage multiple files. Before we distribute this configuration to our users, we need to make sure that all personal certificates and keys are correctly generated and inserted into this file for each user. This is essential for the security of our VPN connections.

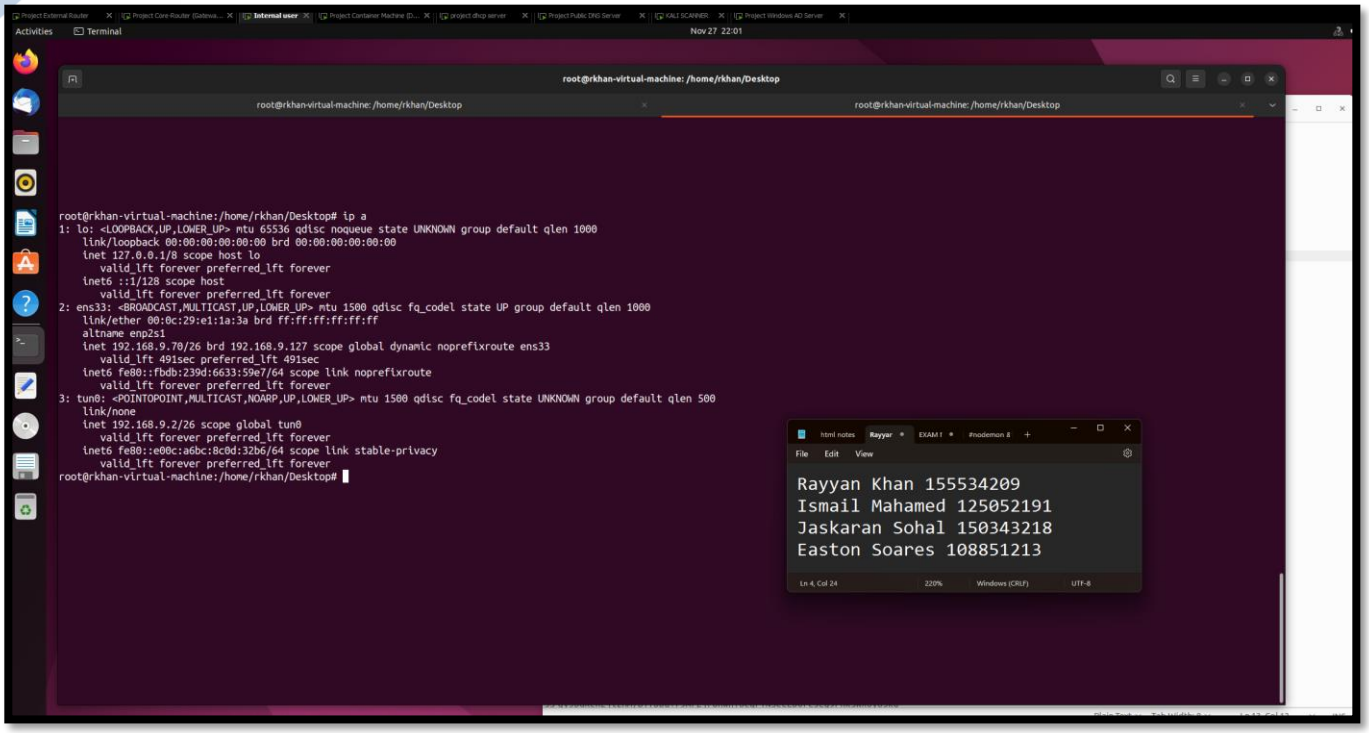


Figure 5: Tun0 OpenVPN Interface

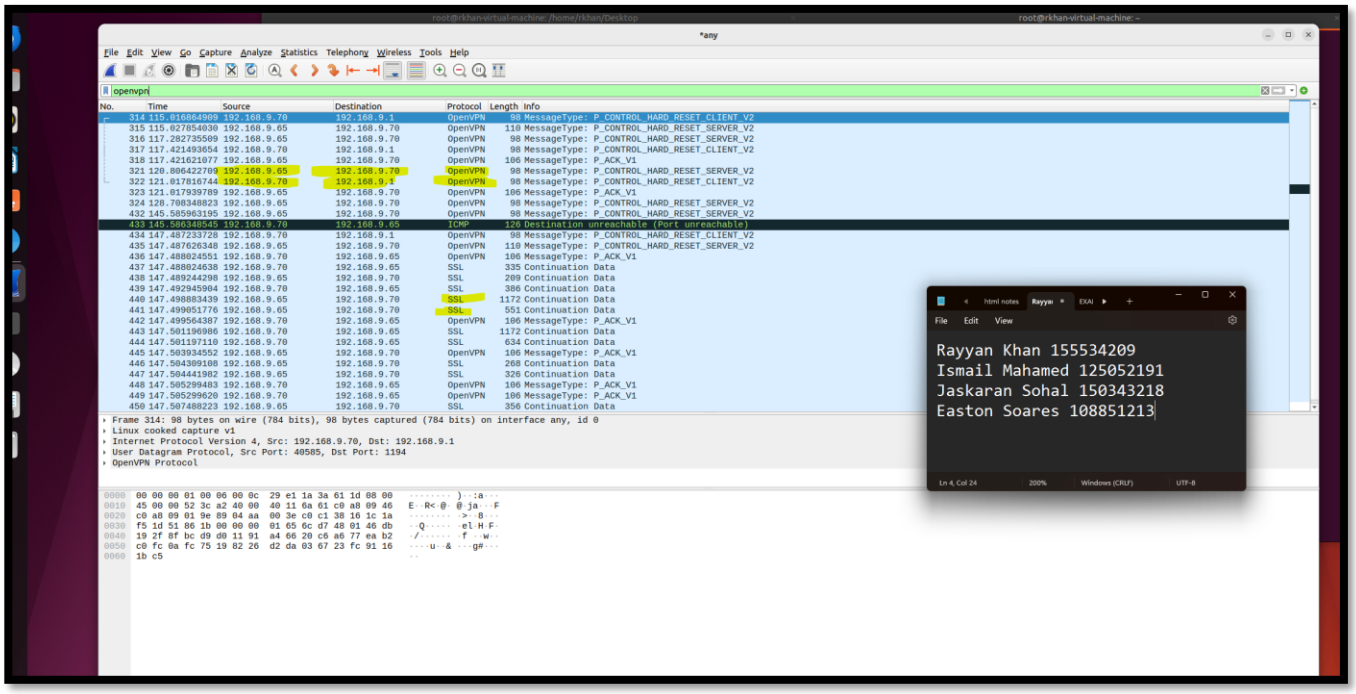


Figure 6: OpenVPN Wireshark

There are multiple packets using the OpenVPN protocol, indicating that OpenVPN is being used to create a secure connection. The packets are being transmitted over UDP, which is typical for OpenVPN due to its speed and efficiency. The `P_CONTROL_HARD_RESET_CLIENT_V2` and `P_CONTROL_HARD_RESET_SERVER_V2` messages are part of the OpenVPN handshake process. The `CLIENT_V2` and `SERVER_V2` messages are indicative of the initial negotiation between the client and server to establish a new VPN session. The destination port `1194` is the default port used by OpenVPN, which further confirms that this traffic is related to an OpenVPN service. Several packets are labeled as SSL, which likely

represent the TLS encrypted traffic that OpenVPN uses to secure the connection. This includes the actual data transmission as well as the exchange of cryptographic parameters. Packets labeled `P_ACK_V1` suggest that these are acknowledgment packets used in the transmission control process within OpenVPN. Even though UDP does not have built-in acknowledgment like TCP, OpenVPN implements its own mechanisms to ensure delivery. The timestamps show a continuous sequence of communication without significant delays, which indicates that the VPN connection is currently active and that there's a steady exchange of packets.

Snort IDS

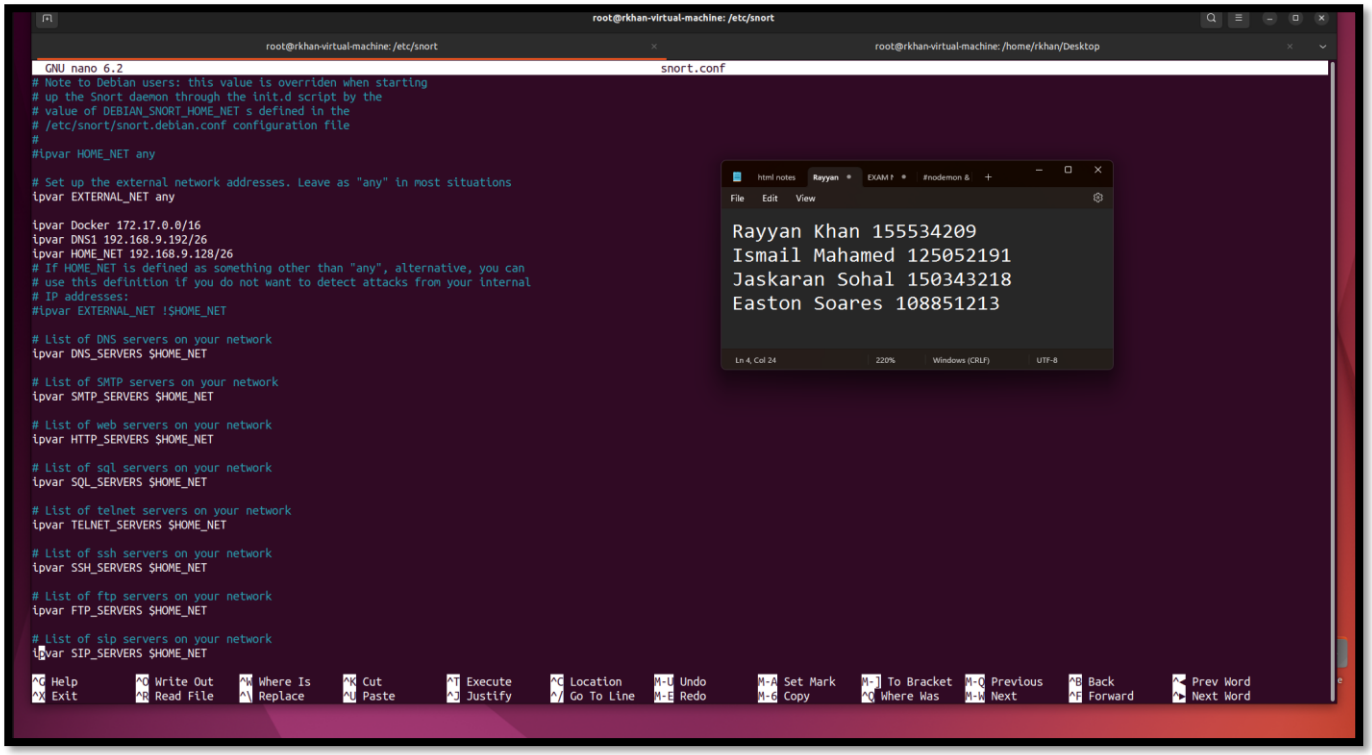


Figure 7: Snort Config File

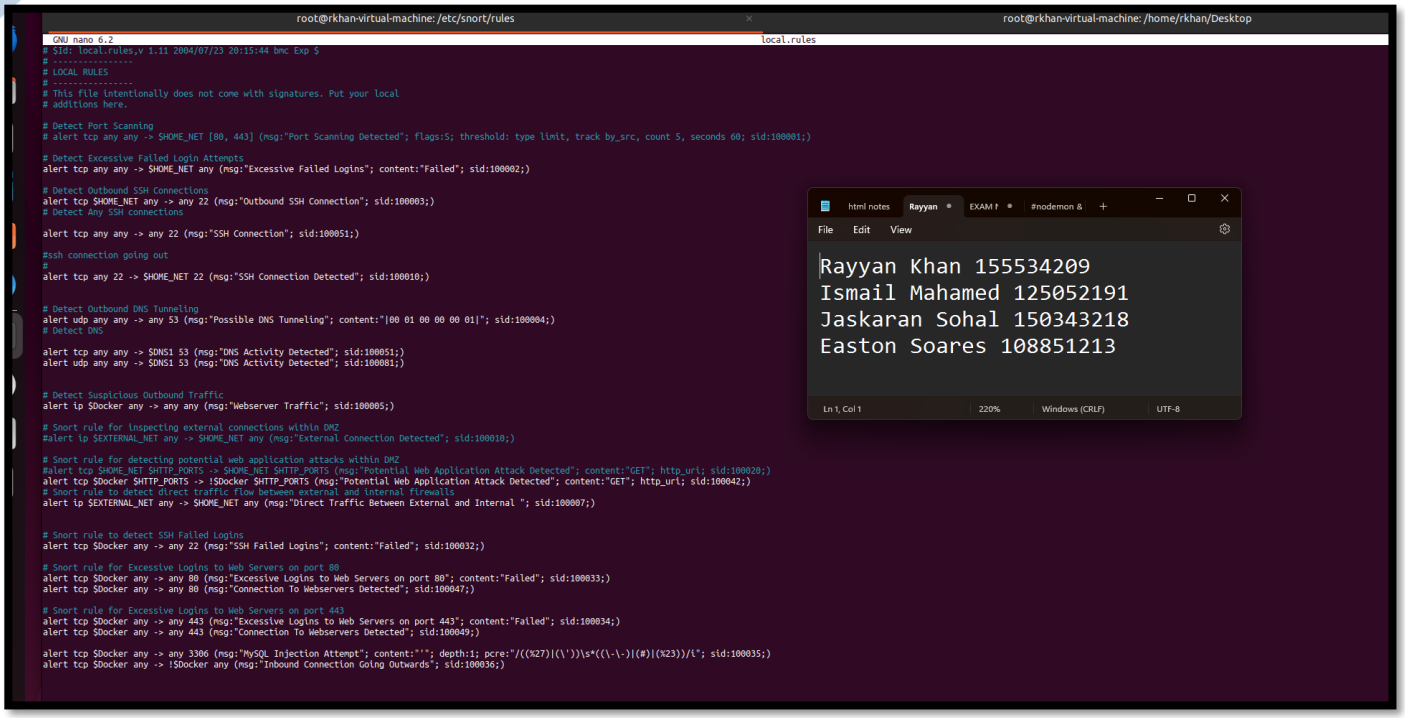


Figure 8: Local Rules

First off, we've defined a rule to detect port scanning activity. This rule flags any TCP traffic that hits ports 80 or 443 on our `$HOME_NET`, which are the common web ports. If there's a scan from a single source on these ports more than five times in 60 seconds, we'll get an alert. It helps us catch anyone who's scouting our network.

We've also set up a rule to catch excessive failed login attempts. If we see more than 5 failed login attempts on our `$HOME_NET` in 60 seconds, it will trigger an alert. This is a clear indicator of someone trying to brute force their way into our systems.

For SSH traffic, we've created rules to monitor both inbound and outbound SSH connections. Any SSH connection attempts to and from our `$HOME_NET` are logged. It’s crucial since SSH can be a vector for data exfiltration or unauthorized access.

Now, we've considered the possibility of DNS tunneling, which can be used to sneak data through DNS requests. Our rule looks for unusually large DNS packets, which could be a sign of this kind of activity. We're capturing both any and DNS53 traffic to be thorough.

We're also watching for suspicious outbound traffic that could indicate a compromised web server. This is tagged as 'Webserver Traffic' and will alert us to any unusual outgoing requests.

Inside our DMZ, we have rules to inspect external connections. We need to know if something is reaching out to our exposed services or if we're seeing potential web application attacks, like those signaled by certain HTTP GET requests. We've got a specific rule for Docker HTTP ports too since we're using containers, and we need to be aware of any potential web application attack attempts.

Lastly, we've got a rule for SQL injection attempts. This looks for the specific pattern of an SQL injection attack in the payload of packets going to our MySQL server on port 3306. We've written this rule to be quite precise, so it should minimize false positives.

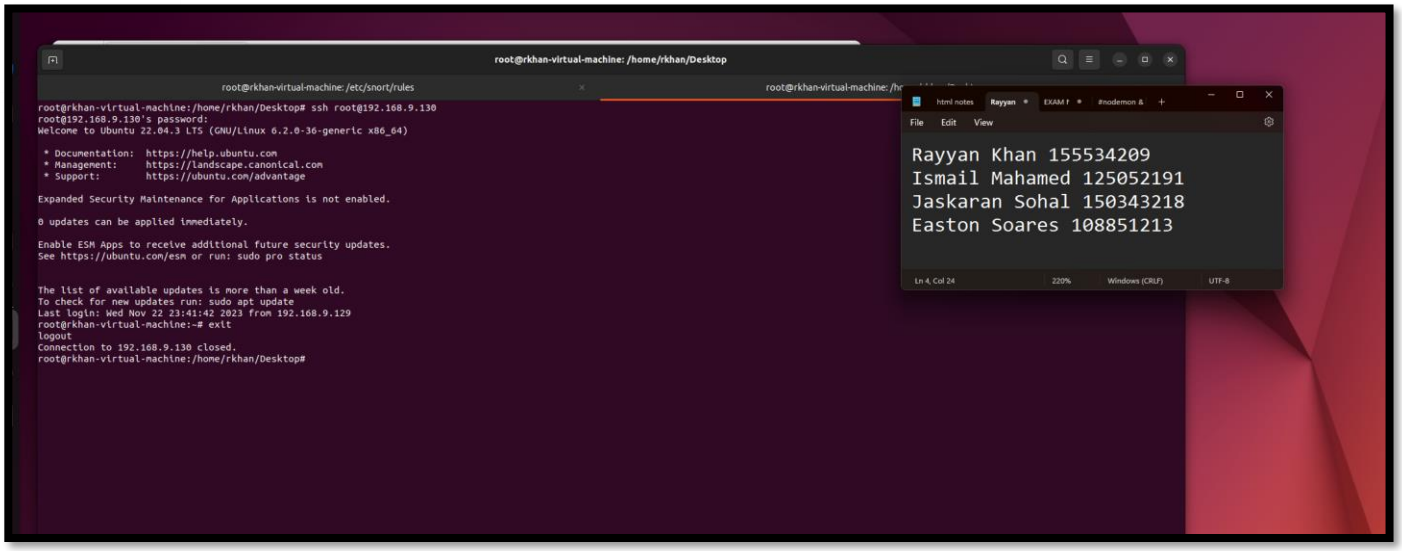


Figure 9: SSH Connection

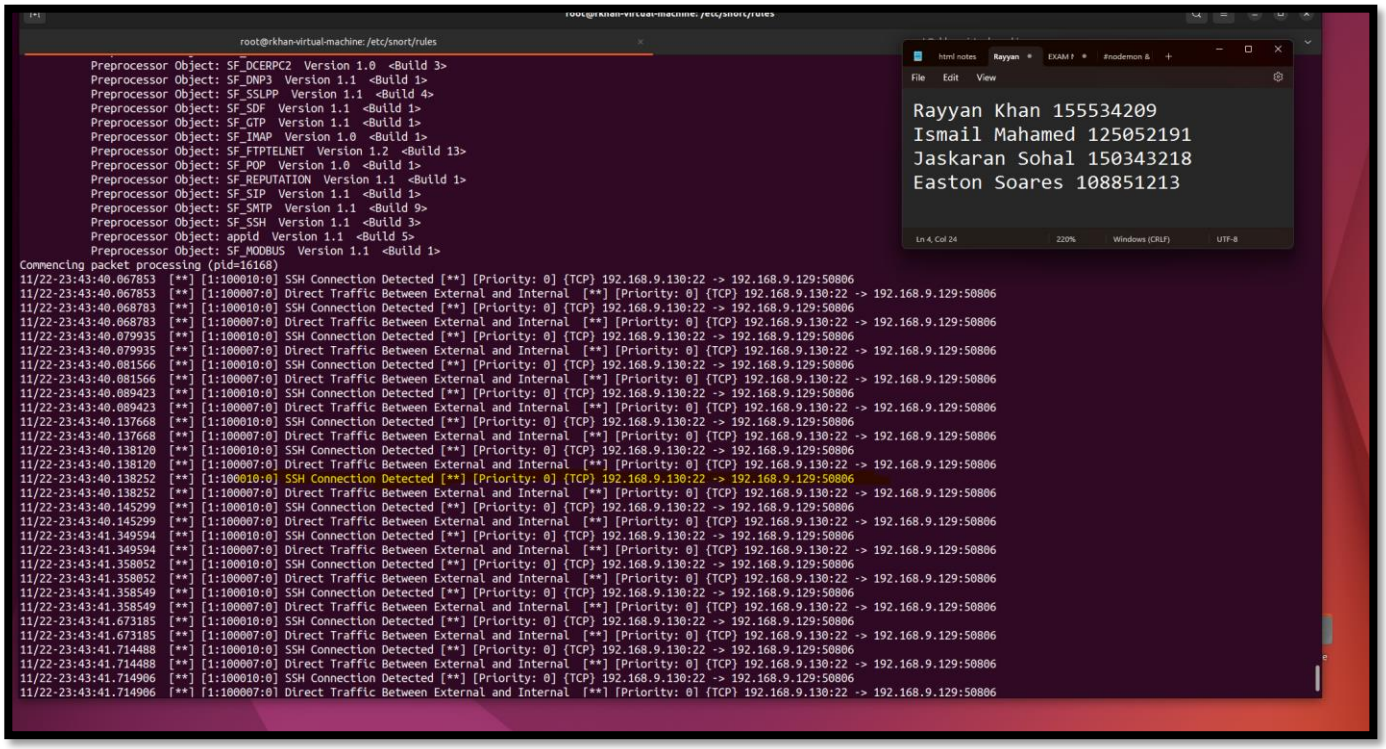


Figure 10: DNS Snort Logs

his screenshot from our Intrusion Detection System (IDS) using Snort is showing that our configuration is working as intended when we test it with an SSH connection.

We've set up our IDS to monitor and alert us whenever there is SSH traffic between external and internal IP addresses on our network. In this case, we can see a series of alerts triggered by SSH traffic between the external IP `192.168.9.130` and the internal IP `192.168.9.129` on the port `58006`. This is exactly what we expect our IDS to do when it detects SSH connections as per our rules.

Each alert is timestamped to the microsecond, ensuring that we have precise logging of when each event occurs. This level of detail is crucial for post-event analysis or forensics. The priority is set to 0, which in

Snort indicates the highest priority, showing that we consider SSH traffic monitoring to be of utmost importance.

The SID, or Snort ID `[1:100007:0]`, corresponds to the specific rule we wrote for detecting SSH connections. The '0' at the end of the SID indicates the revision number of the rule, which is useful for us if we need to update or troubleshoot the rule in the future.

The message "SSH Connection Detected External and Internal" is the message we defined in our Snort rule, making it clear and identifiable in the logs. This is the description that will help any team member understand what's happening without having to dig into the rule details.

Our IDS is not only capturing the events, but it's also logging the traffic in the background. If we need to, we can go back and analyze the full content of these connections to see if there was any actual breach or if it was just benign activity such as our own testing or an authorized user accessing the system.

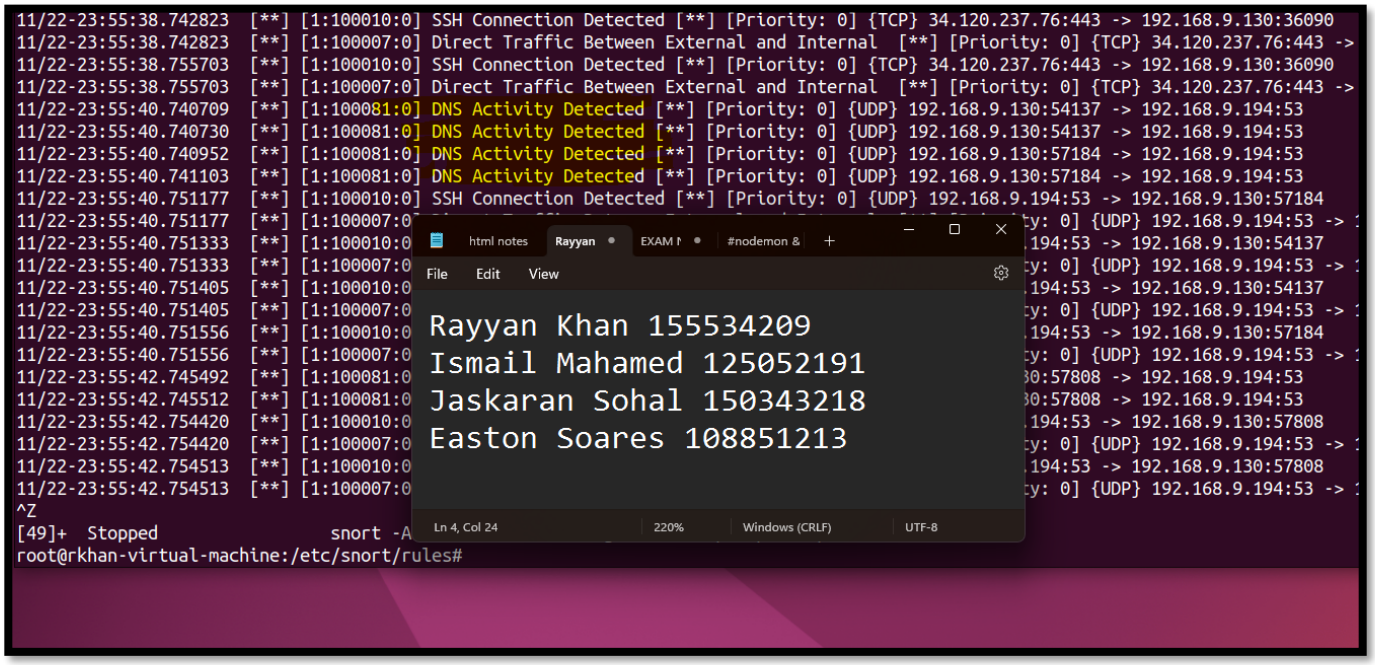


Figure 11: DNS Logs

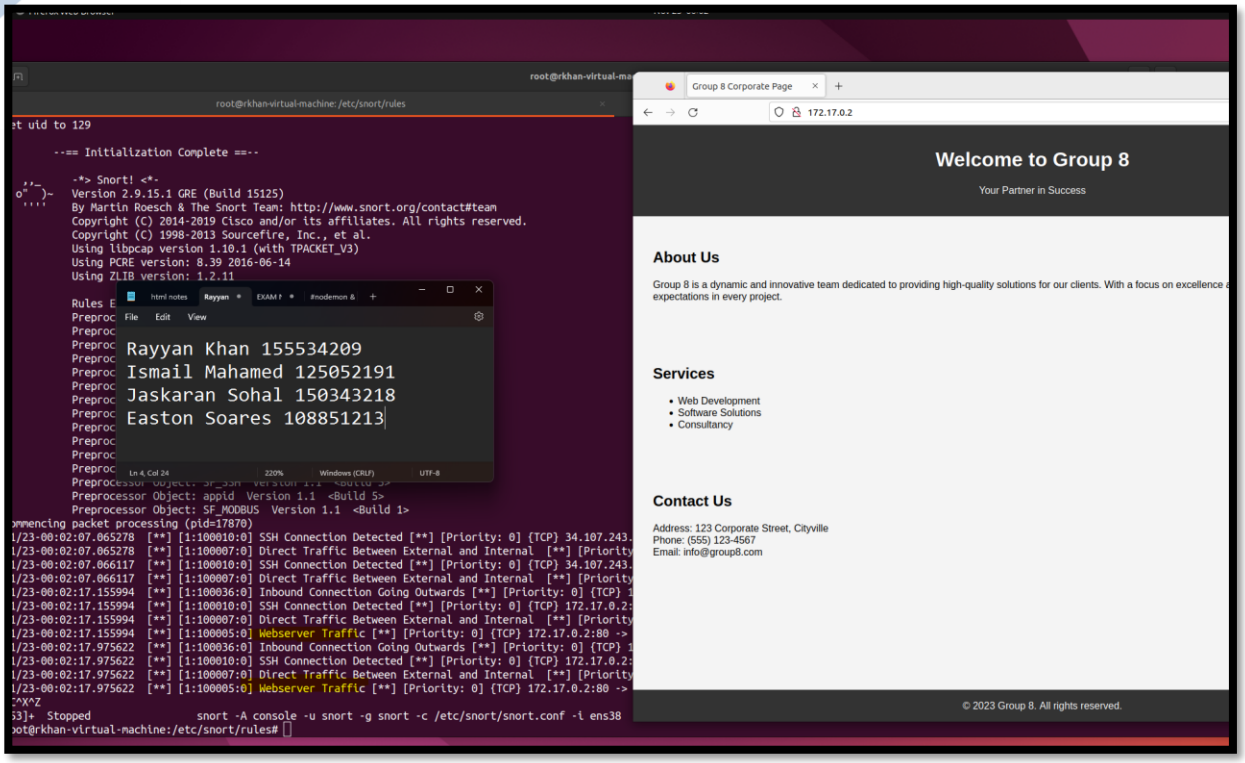


Figure 12: Webserver Logs

We also see a similar level of logging when we connect to our webserver or when the DNS is active.

Mod Security / HTTPS

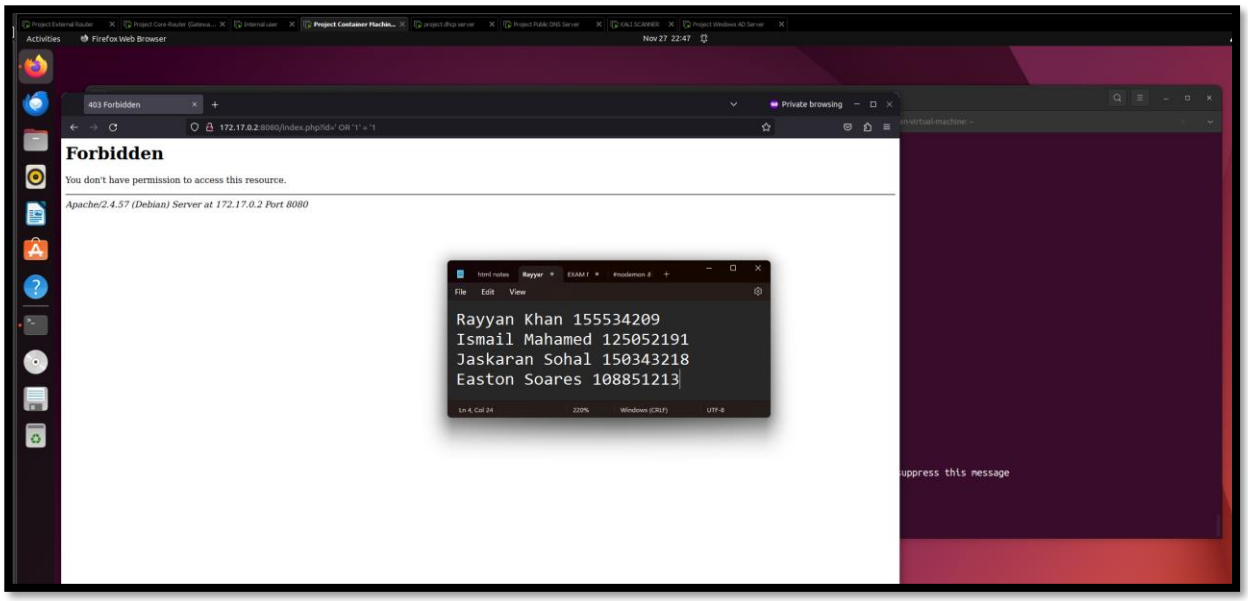


Figure 13: Modsecurity Blocking Website Access

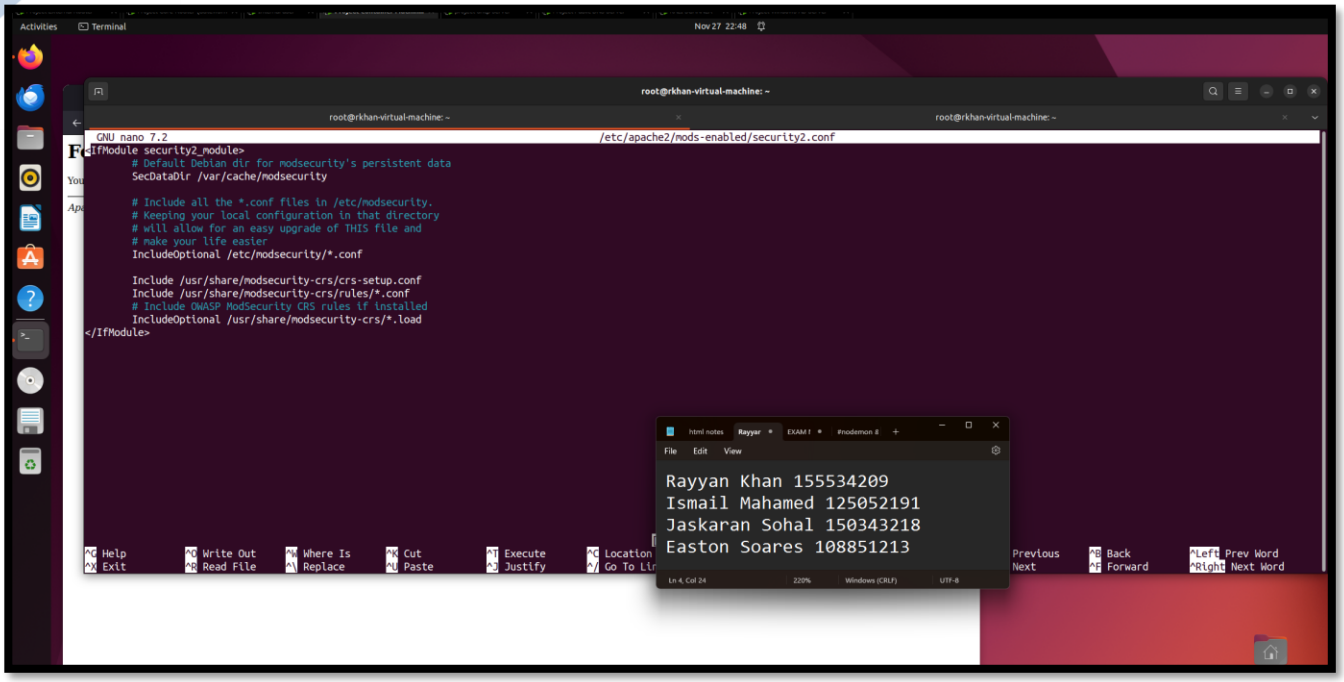


Figure 14: Modsecurity Configuration

Let's dive into the ModSecurity configuration we've put together for our web application firewall (WAF). It's designed to help us detect and prevent various types of attacks, such as SQL injection, session hijacking, and cross-site scripting (XSS), among others.

In the first screenshot, we've set up the base configuration. Here's what we've done:

- We've included the main ModSecurity configuration file, `modsecurity.conf`, which sets the basic operation rules for ModSecurity. This includes the default settings which we can tweak later if needed.
- We've then included the recommended default rule set by importing `owasp-modsecurity-crs.conf`. This file is part of the OWASP Core Rule Set (CRS), which provides us with a strong base of generic attack detection rules.
- The next line, `Include /usr/share/modsecurity-crs/rules/*.conf`, ensures that all the individual rule files within that directory are loaded. These are the granular rules that target specific attack vectors and vulnerabilities.

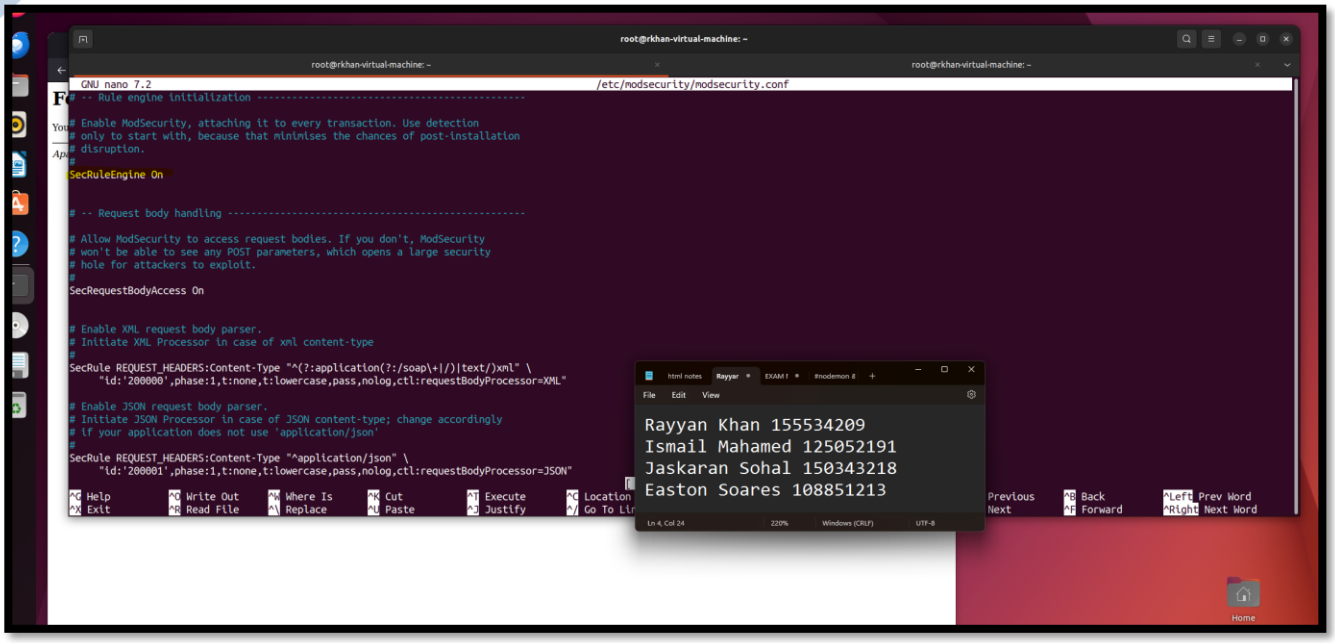


Figure 15: Configuration ModSecurity Cont.

In the second screenshot, we've added some more specific configurations:

- We've set `SecRuleEngine On`, which activates ModSecurity. If we set this to `DetectionOnly`, it would only log the attacks without actually blocking them.
- With `SecRequestBodyAccess On`, we're instructing ModSecurity to inspect the body of POST requests, which is essential for detecting attacks that leverage POST data.
- We've set `SecResponseBodyAccess On` to make sure we also inspect the data that's being sent back to users. This is especially useful to prevent data leakage through our responses.
- `SecResponseBodyMimeType` is set to include text/html, which means we're particularly looking at HTML content for any kind of malicious payload that might be reflected back to the user, a common vector for XSS attacks.
- The `SecResponseBodyLimit` and `SecResponseBodyLimitAction` directives control how much response data ModSecurity will hold in the buffer for inspection. We've set a limit that makes sense for our expected traffic to avoid performance issues.

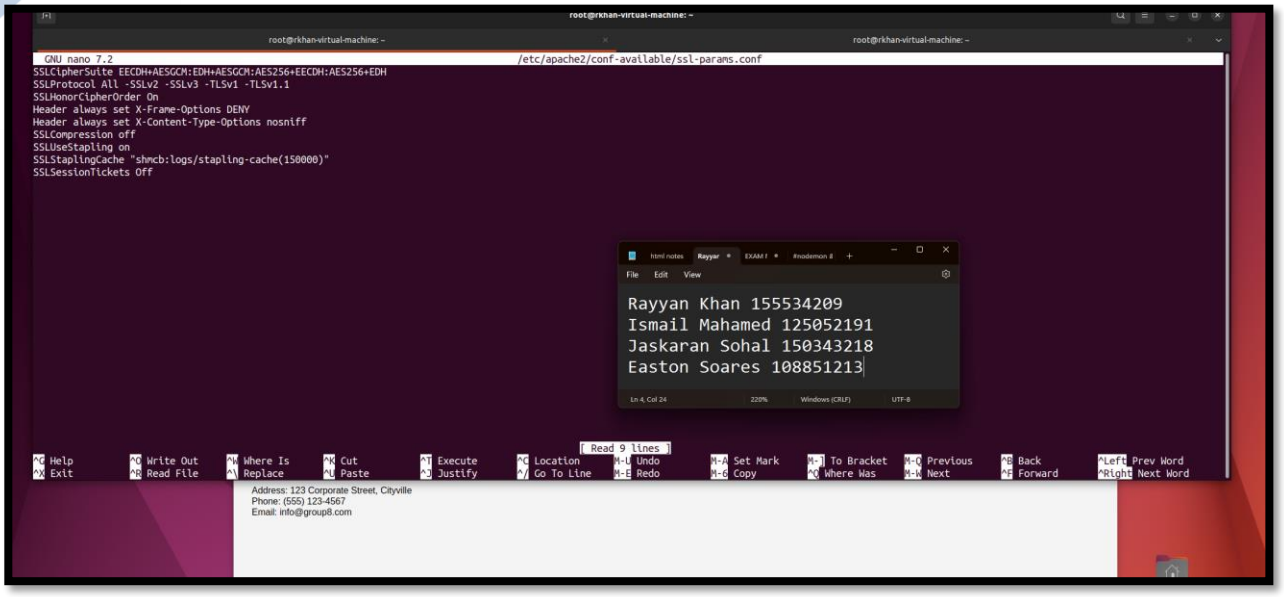


Figure 16: HTTPS/SSL Config File

In this configuration snippet for Apache's HTTP SSL module, we've taken steps to secure our web server communications. Here's the rundown of what we've implemented:

We've specified the `SSLEngine on` directive, which means we've activated SSL/TLS for this server block. This is essential for encrypting traffic between our web server and the clients. With `SSLCertificateFile` and `SSLCertificateKeyFile`, we've pointed to the location of our SSL certificate and private key. These files are critical for establishing a secure connection and verifying the server's identity to clients. The `SSLCertificateChainFile` directive has been set to point to our chain file. This file contains the intermediate certificate(s) that complete the trust chain between our server's SSL certificate and the root certificate on client machines. We've used the `SSLCipherSuite` directive to define which ciphers are permissible for use during SSL/TLS handshakes. We've chosen strong ciphers to ensure secure communication and to protect against vulnerabilities associated with weaker ciphers.

By setting up these directives, we've ensured that any data transmitted between our web server and the clients is encrypted, which is crucial for protecting sensitive information against eavesdropping and man-in-the-middle attacks.

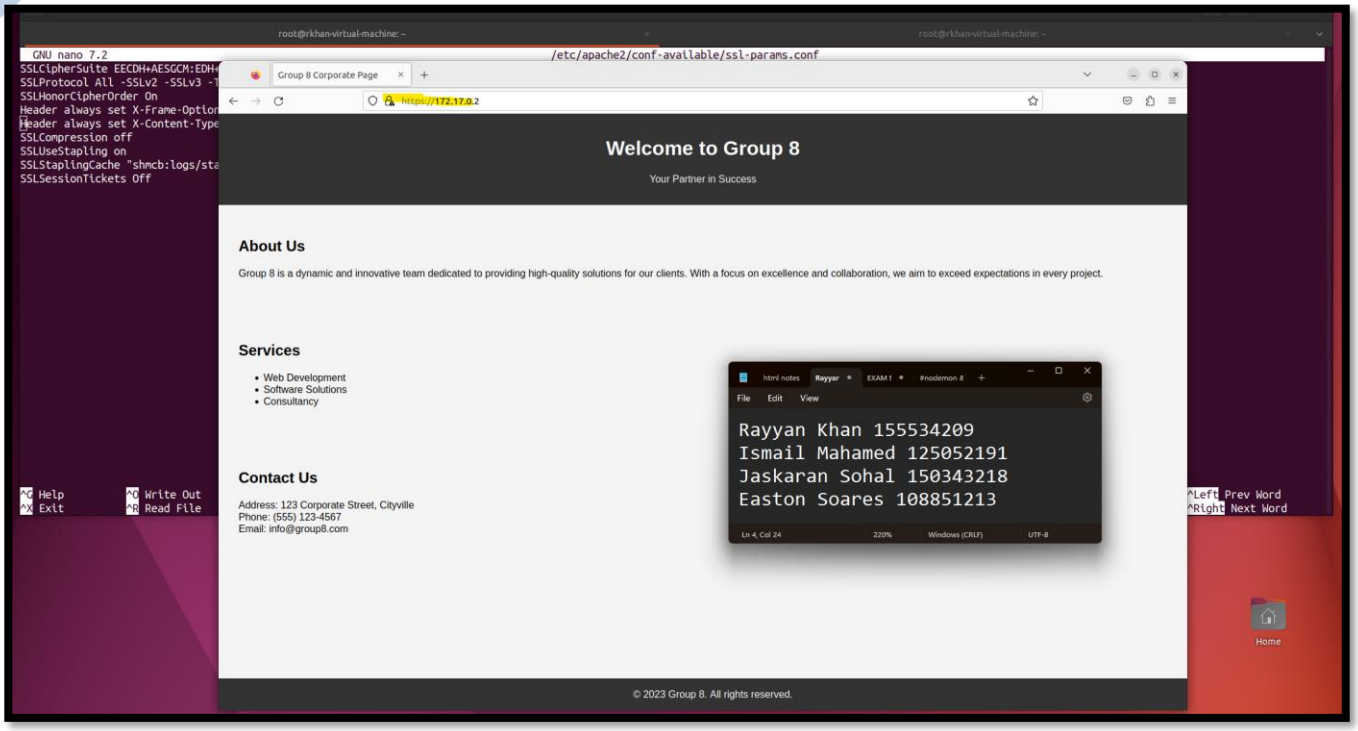


Figure 17: Accessing Webserver via HTTPS

DNS

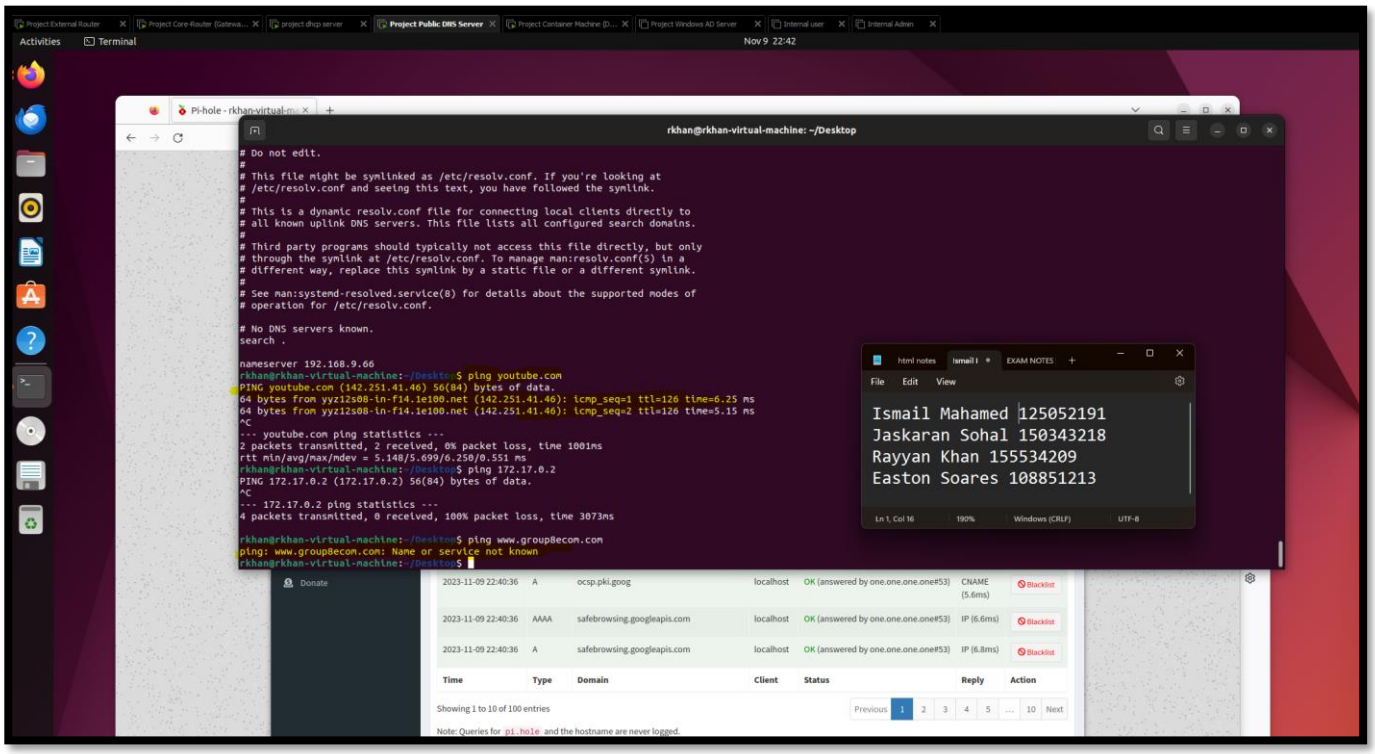


Figure 18: PublicDNS

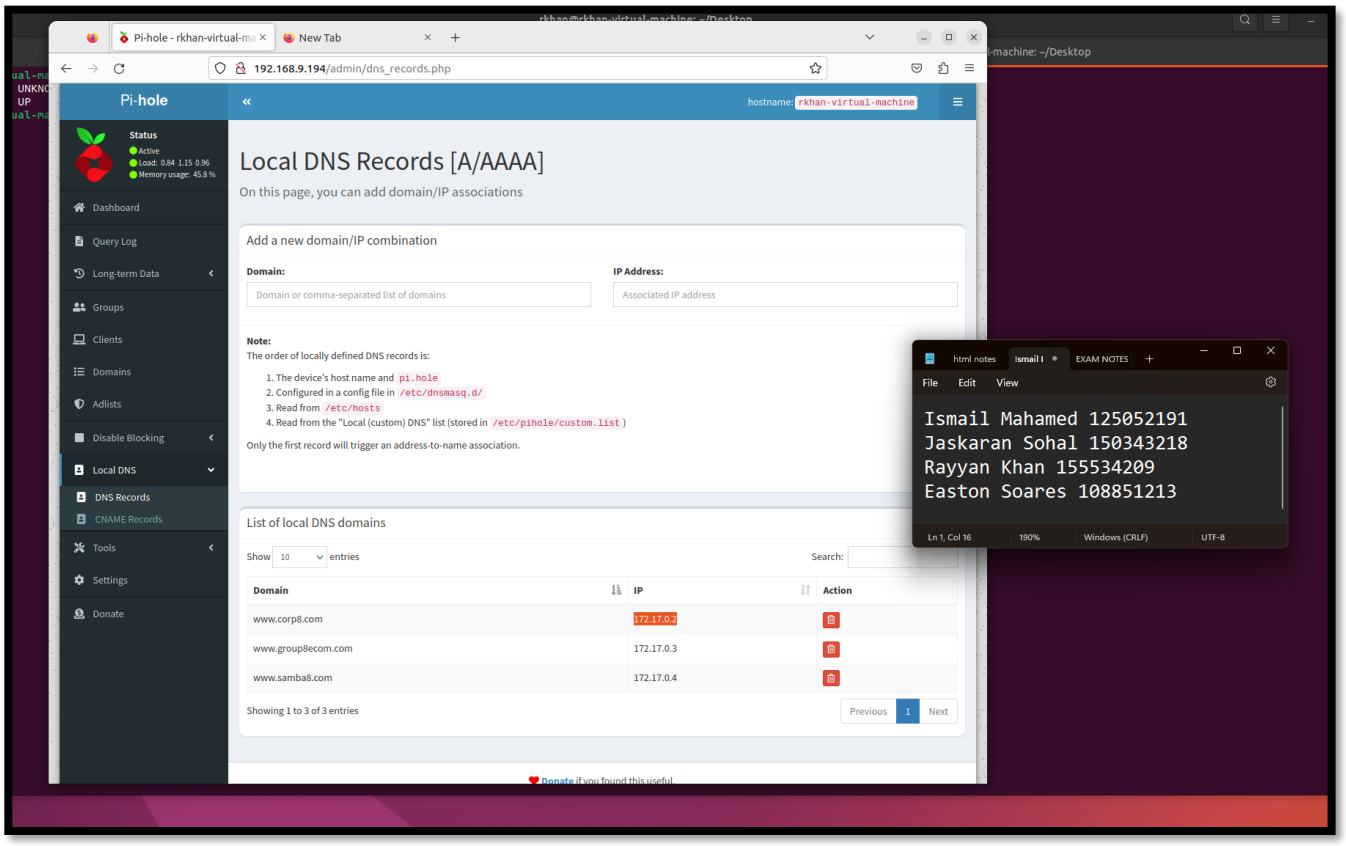


Figure 19: PrivateDNS

We've got our Pi-hole set up as our local DNS server, which gives us control over DNS resolution within our network. With Pi-hole, we've defined local DNS records, allowing us to assign friendly domain names to the devices on our network for easy access. We've entered custom DNS associations, mapping domain names to their corresponding IPv4 and IPv6 addresses. This simplifies our internal network management and makes it more user-friendly. Moreover, Pi-hole serves as our network-wide ad blocker. By default, it blocks queries to known ad-serving domains, reducing unwanted content and improving network performance. By using Pi-hole's local DNS records feature, we also enhance our network security. We can prevent DNS requests to malicious sites by ensuring that only the domains we've specified can be resolved. This local DNS setup is part of our larger strategy to keep our network secure, efficient, and user-friendly. Let's ensure we maintain this list and keep our Pi-hole instance updated to keep our network running smoothly.

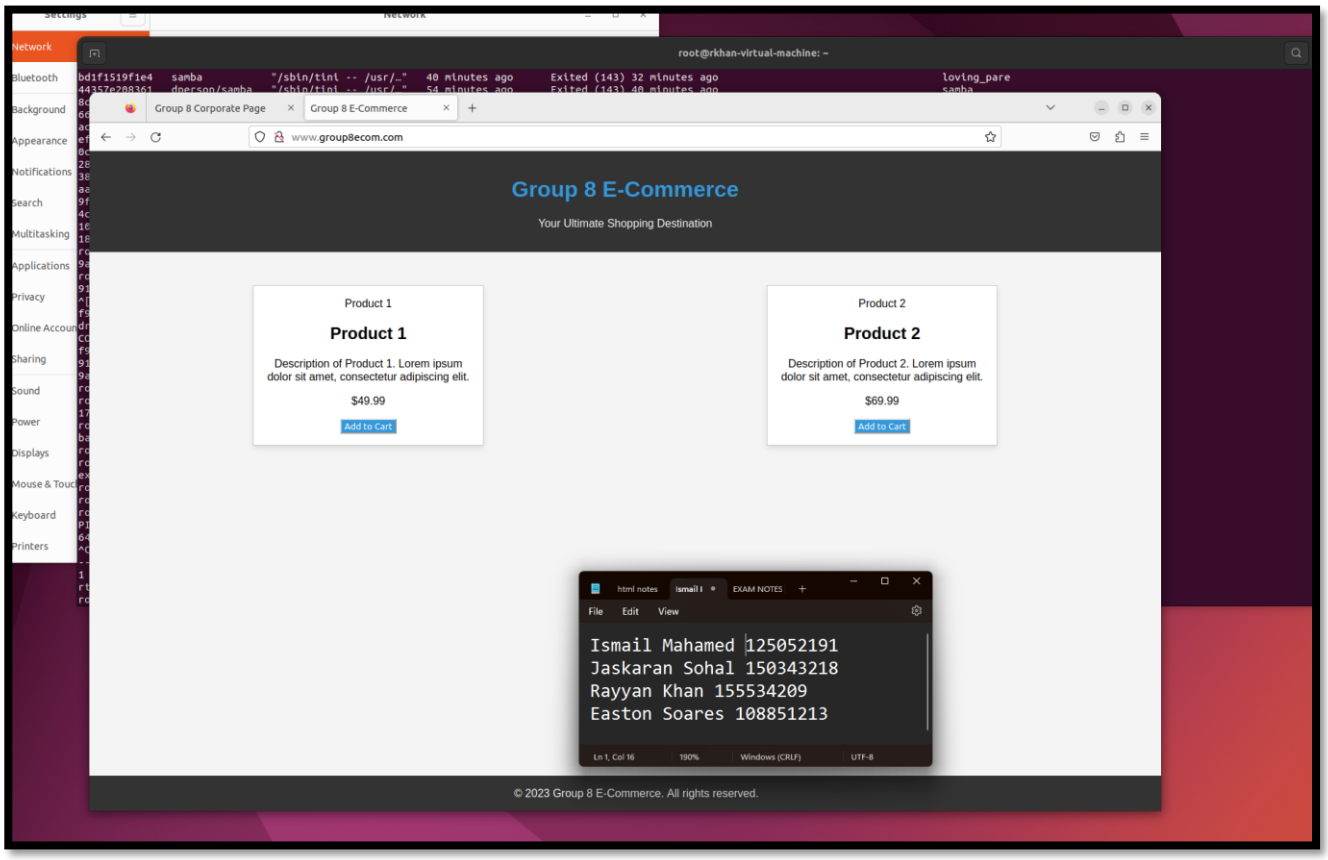


Figure 20: E-Commerce DNS Resolving

When going to our custom address, the DNS is able to resolve the issue fully.

Active Directory

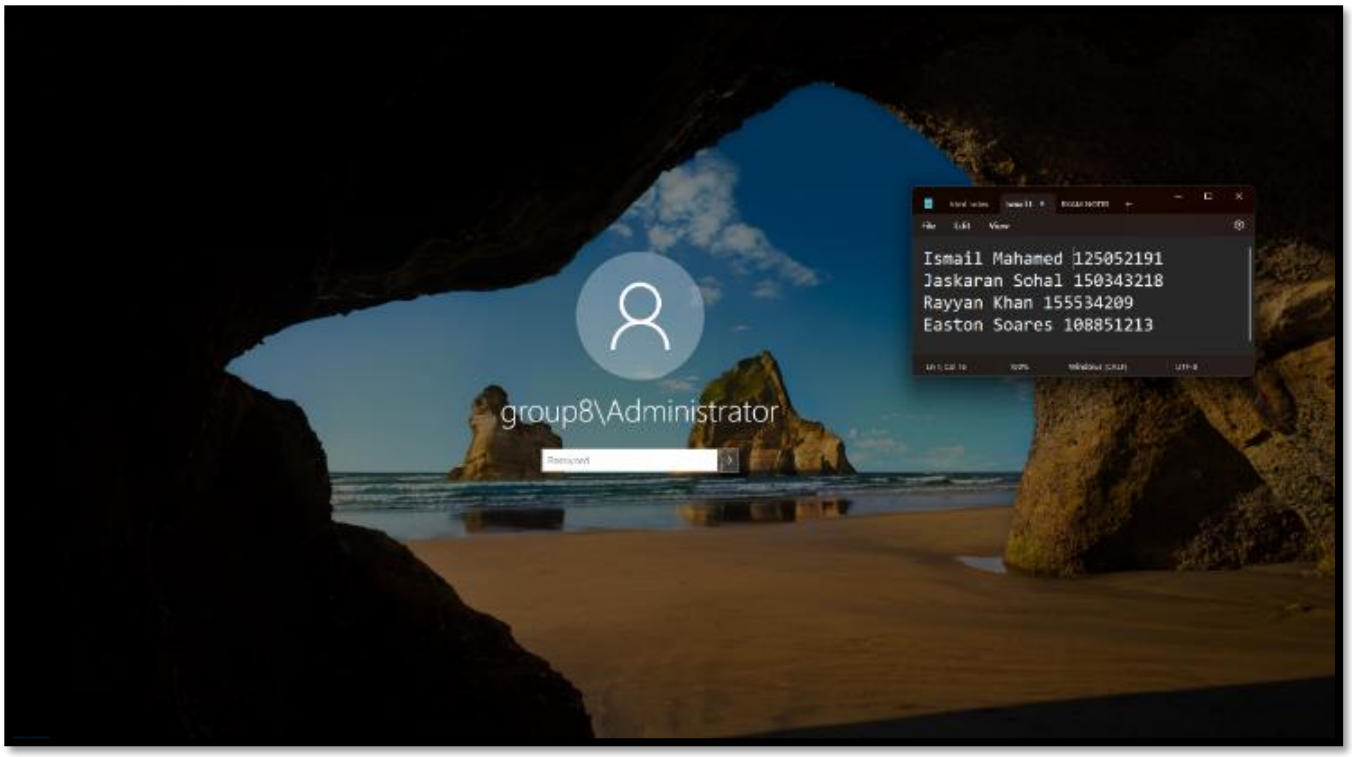


Figure 21: AD Login

This is the login screen of our Microsoft Server where we've configured Active Directory (AD) for our group's internal network management. We've set up a domain controller that allows us to manage and store information about network resources, enforce security policies, and authenticate and authorize all users and computers within our domain. Within our AD setup, we've created organizational units (OUs) that reflect our company's structure, making it easier to manage permissions and policies for different departments. We've also established group policies to automate the deployment of settings, software, and permissions across our network, ensuring a consistent working environment for all users.

For security, we've implemented strict password policies, ensuring complexity and regular changes to minimize the risk of unauthorized access. We've enabled auditing and monitoring to keep track of login attempts, access requests, and other changes within the network, which helps us maintain a high level of security. By integrating our AD with other services like DHCP, DNS, and file services, we've streamlined the network management, allowing for a centralized and efficient administrative experience. This setup has enabled us to maintain a robust, secure, and well-organized network infrastructure. Let's continue to monitor and refine our AD configuration to meet our evolving organizational needs.

Ansible & Firewalls

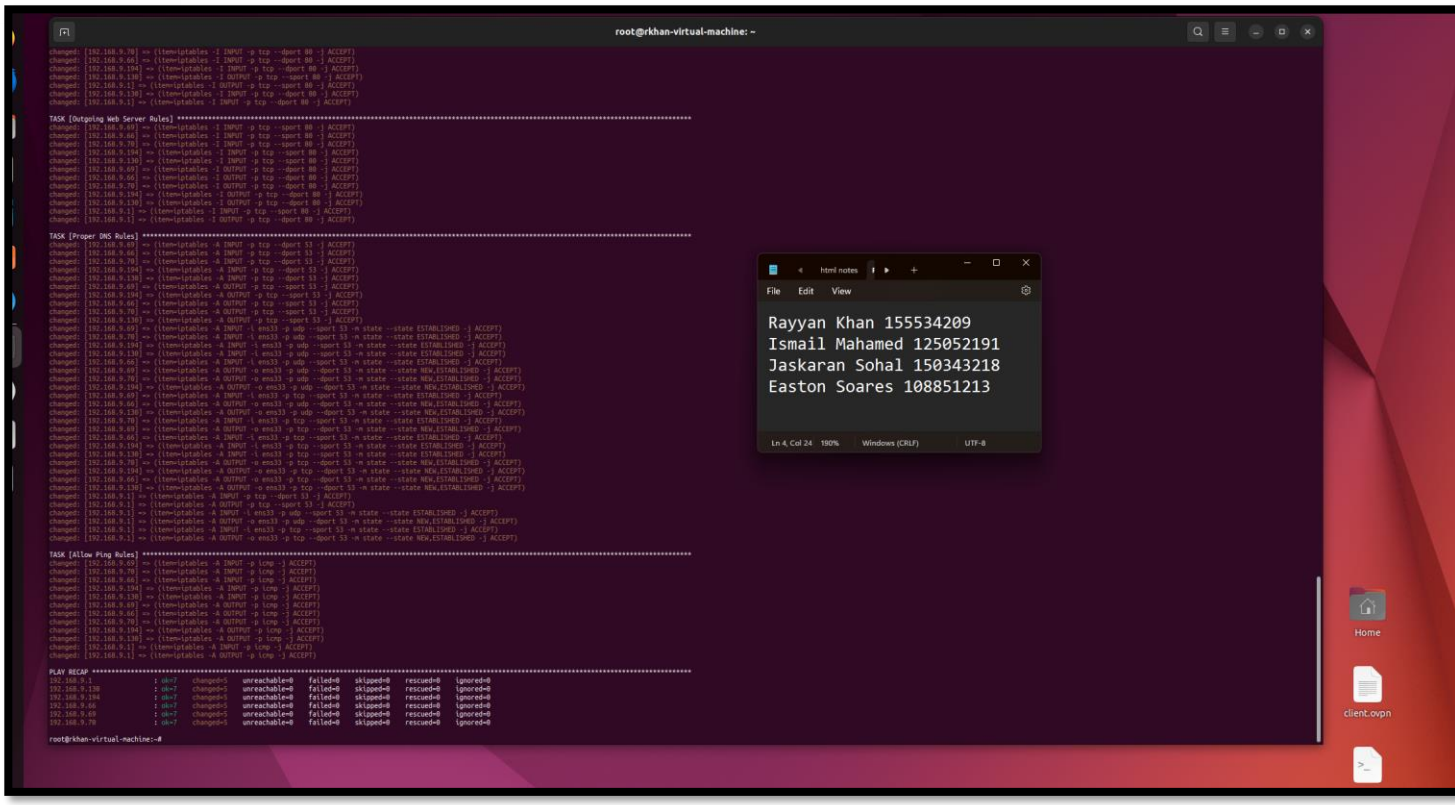


Figure 22: Ansible Playbook Running & Changing Firewalls

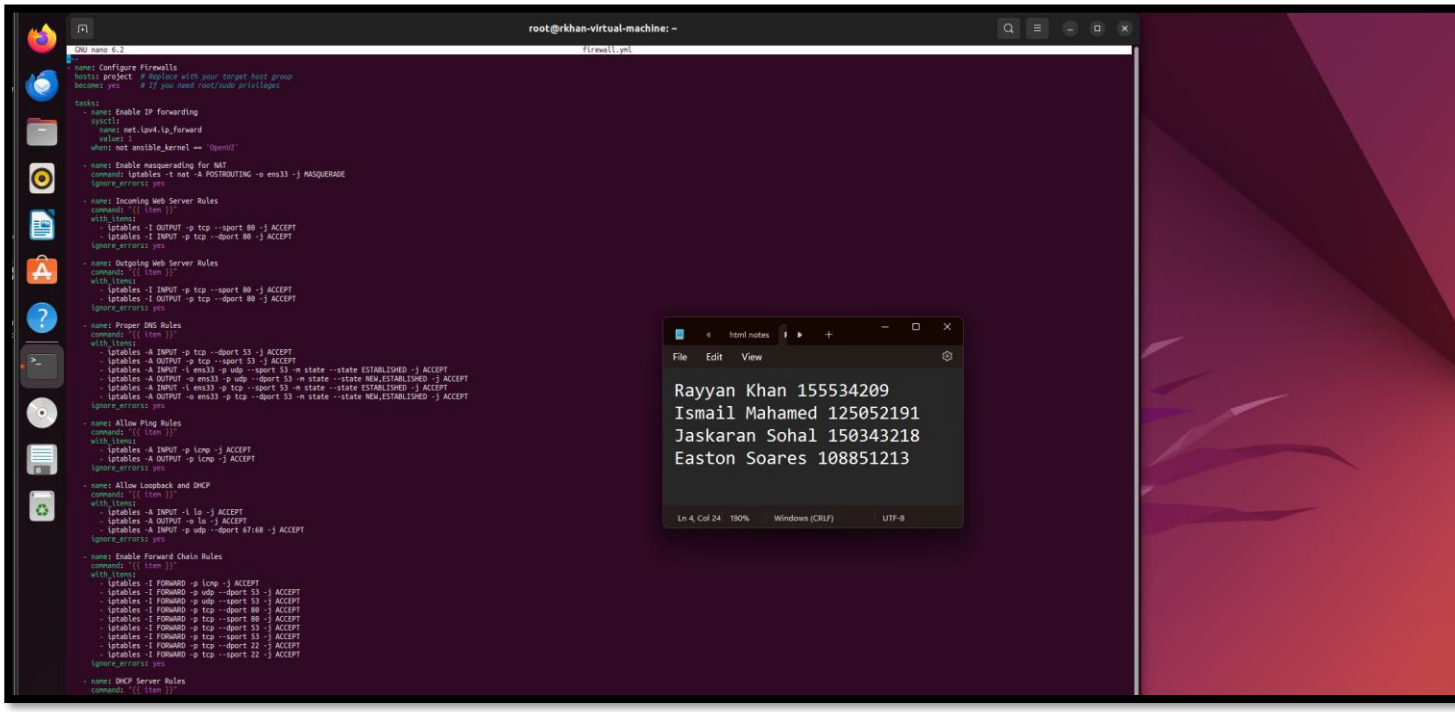


Figure 23: Firewalls.yml Ansible Playbook

As we look at our network's terminal screen, we can see the automation scripts we've created to manage our firewalls and network configurations. We've set up these scripts to run at specified intervals, ensuring that our network defenses are dynamically adjusted based on the latest security policies we've developed.

In these scripts, we've included commands to update our firewall rules, which control the traffic allowed to and from our network. We've automated the process of adding new rules, modifying existing ones, and removing outdated or unnecessary ones, ensuring our network remains secure against evolving threats.

We've also implemented scripts that automatically back up our firewall configurations. This means that we have a recovery point to revert to in case any changes cause issues. Our proactive monitoring system is set up to alert us if the network encounters any suspicious activity, such as repeated login failures or unusual traffic patterns, triggering specific scripts that respond to these events.

Additionally, we've utilized cron jobs to handle the timing of these scripts, ensuring that tasks like system updates, log reviews, and intrusion detection system checks are performed consistently without manual intervention. By automating these critical tasks, we ensure our network's integrity and security are maintained with minimal downtime and reduced human error.

Our approach to automation not only enhances our network security but also allows us to allocate our time and resources more efficiently, focusing on strategic tasks rather than routine maintenance. Let's continue to refine these scripts and automation processes to stay ahead of the network security curve.

DHCP

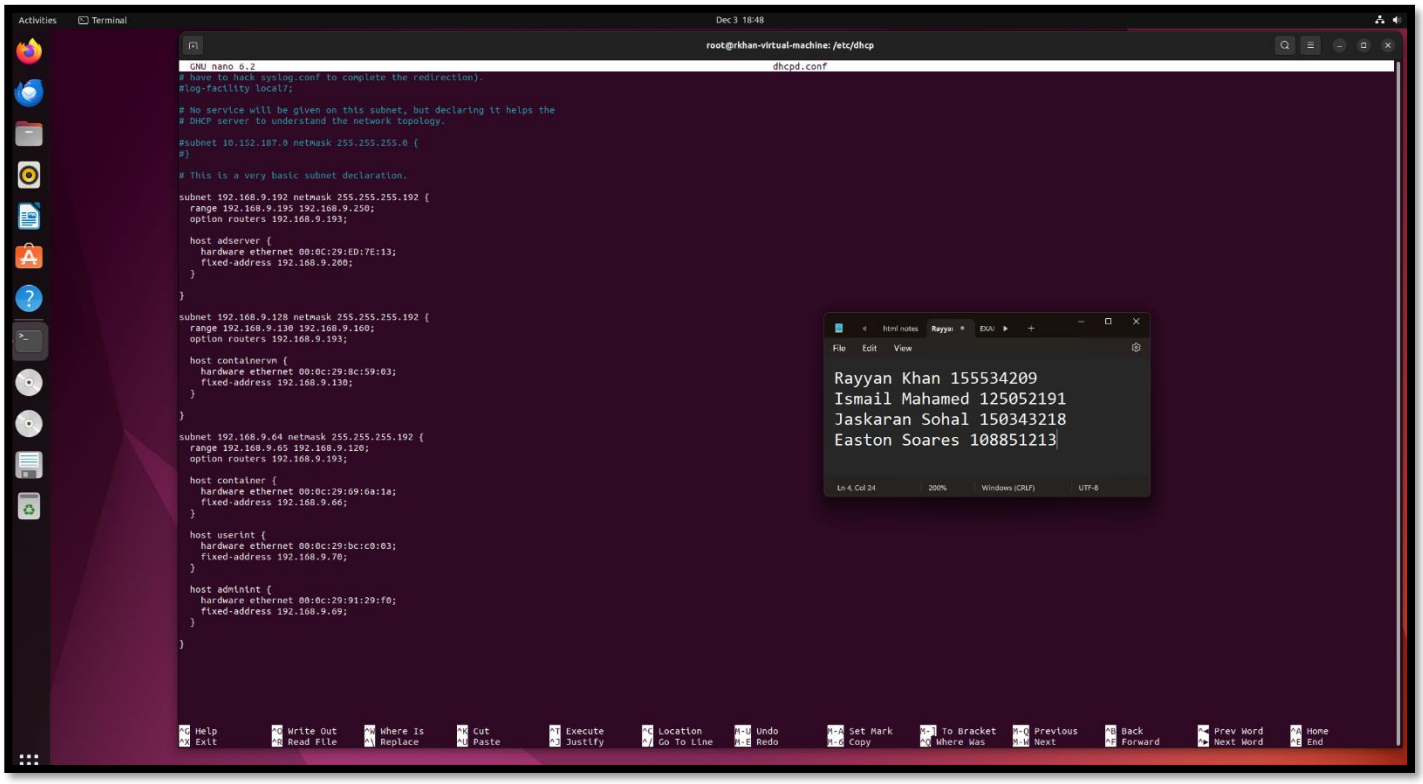


Figure 24: DHCP Running & Configs

Our Dynamic Host Configuration Protocol (DHCP) setup is a crucial part of our network infrastructure, managing IP address allocation and helping devices to connect seamlessly. We've tailored the DHCP configuration file to define different scopes and settings for various segments of our network.

In our DHCP configuration, we've outlined specific subnets, each with a range of IP addresses that are assigned to devices based on their network segment. For example, we've set aside a block of IPs for our servers, another for our office workstations, and yet another for our guest network. Within each subnet declaration, we've specified the range of IPs to be leased, the subnet mask, and any specific options like default gateways and DNS servers.

We've also set fixed IP addresses for critical infrastructure like servers and network printers, ensuring that these devices have consistent network addresses for reliability and ease of access. We've done this by binding specific MAC addresses to designated IPs within the DHCP server so that each time these devices boot up, they receive the same IP address.

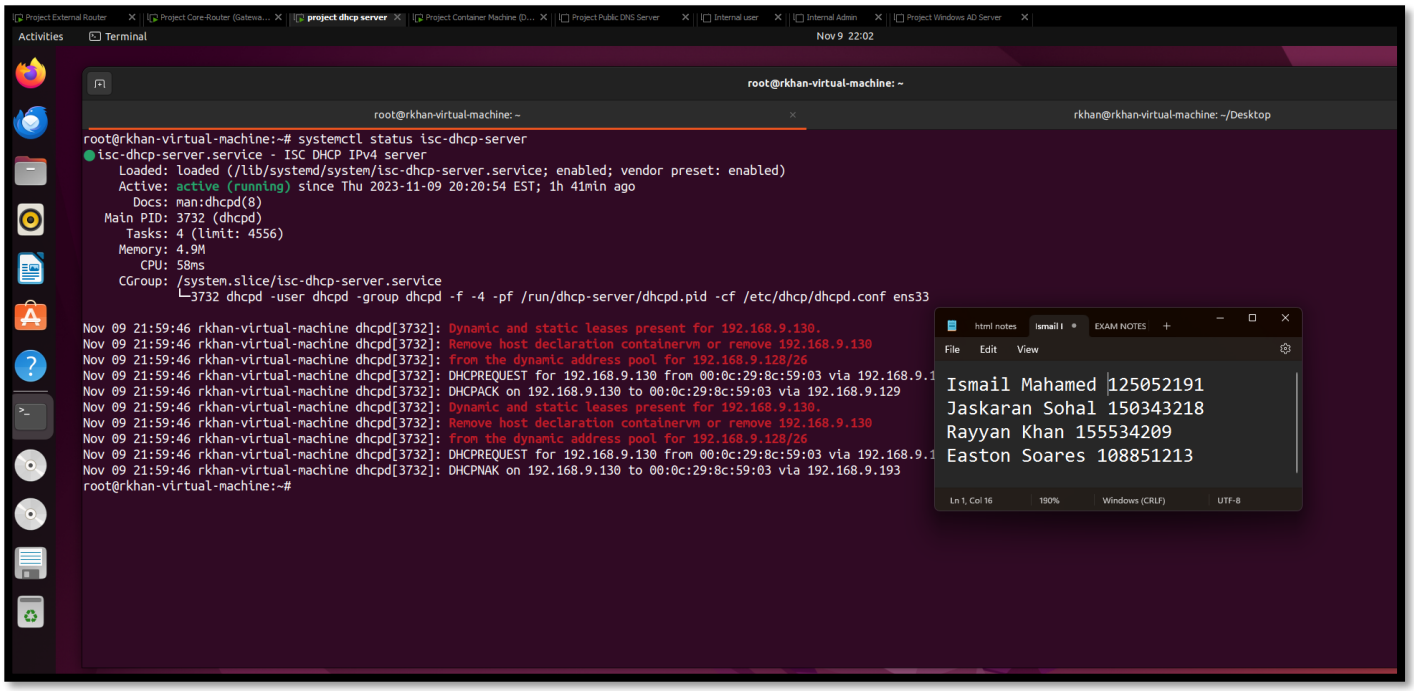


Figure 25: DHCP Status

It's a real-time glimpse into the DHCP leases being issued and renewed. Here we can see the log files filling up with entries as devices join the network, each entry showing the assigned IP address, the MAC address of the device, and the lease duration. This logging is crucial for network management and troubleshooting; it allows us to track which devices are on our network and when they connected.

Our DHCP setup not only simplifies network management but also enhances our security posture by providing us with visibility and control over the devices connecting to our network. By carefully segmenting our network and assigning IPs methodically, we've built a robust and secure environment that supports our group's operational needs

Docker Container

We've leveraged Docker to streamline and isolate our application environment, ensuring consistency and efficiency across our development and production stages. Our Docker containers are configured to run our web applications and services in a lightweight, portable fashion.

We have set up a Docker environment where we can see the command-line interface showing the active containers. We've pulled images from Docker Hub, which are the base images for our services, and we've used Docker Compose to define and run multi-container Docker applications. With our `docker-compose.yml` file, we specify all the necessary configurations, like the container names, ports, volumes, and environment variables. Running `docker-compose up`, we bring our environment to life. This command starts and runs our entire app. We can see in the terminal output the logs that show our containers booting up, with Docker networking setting up the connections between containers, and volumes being mounted to persist our data.

The web browser image shows one of our web services running from a Docker container, accessible via the host machine's web browser. This demonstrates our web server container in action, serving our web app with ease. In the third screenshot, it's the build process of our containers, where we compile, build, and prepare our services. We've tagged our images for version control, and we're pushing them to our private registry or Docker Hub for deployment or further development.

Our Docker setup encapsulates our applications, providing a uniform environment for them to operate. This ensures that our software runs the same, regardless of where it is deployed. It's a testament to our commitment to DevOps best practices, and it supports rapid deployment and scaling of our services.

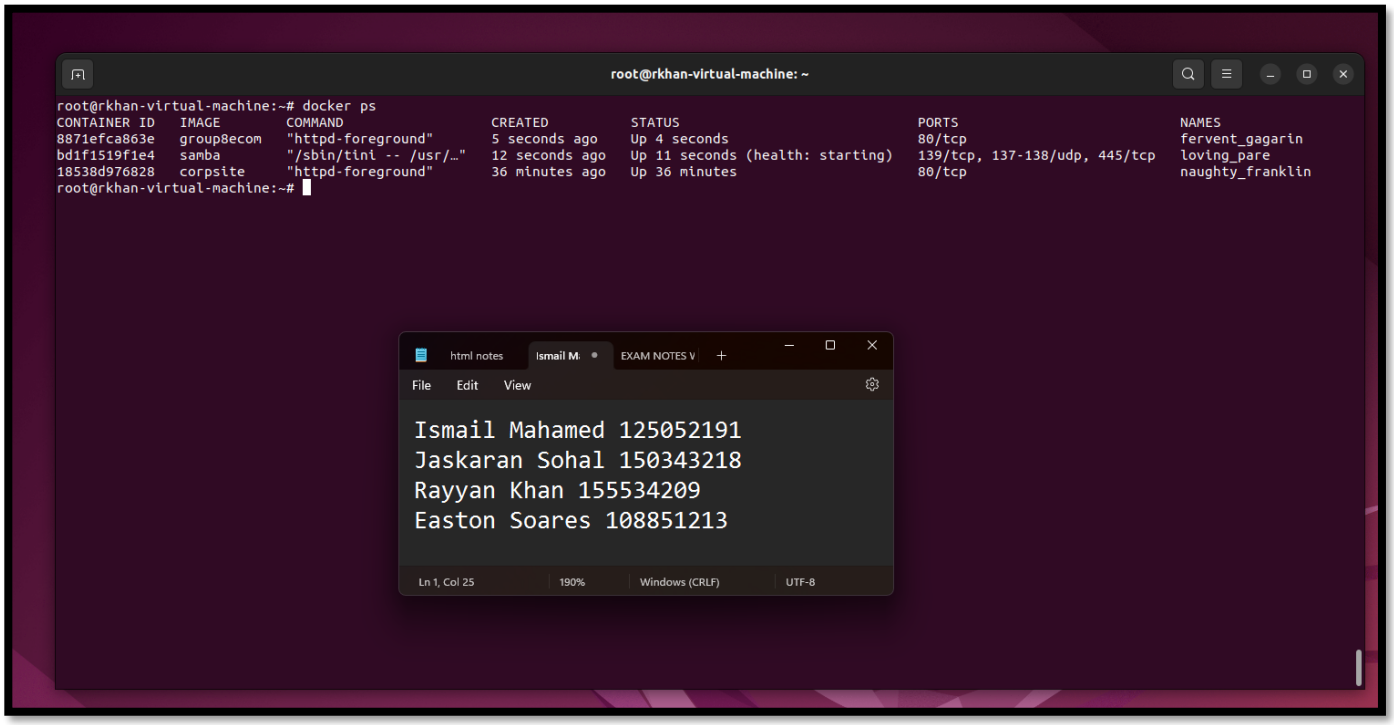


Figure 26: Docker Container Status

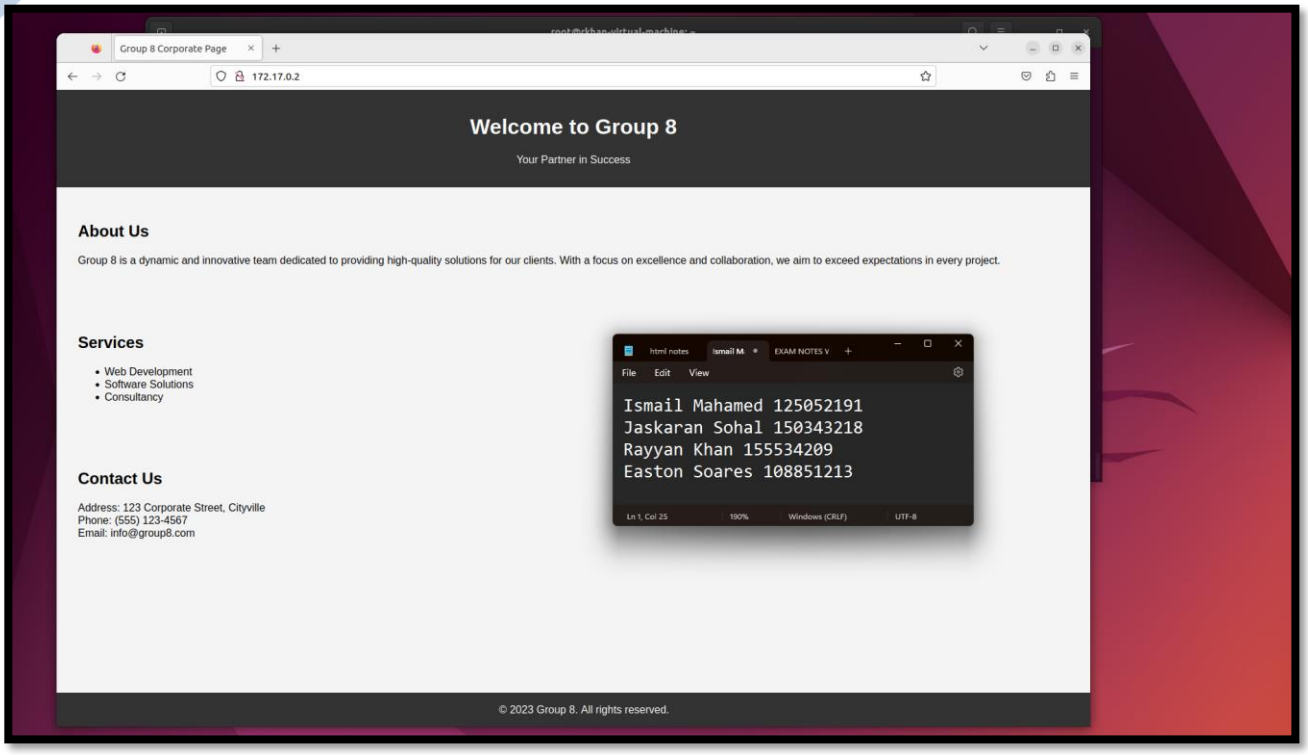


Figure 27: Docker CompSite

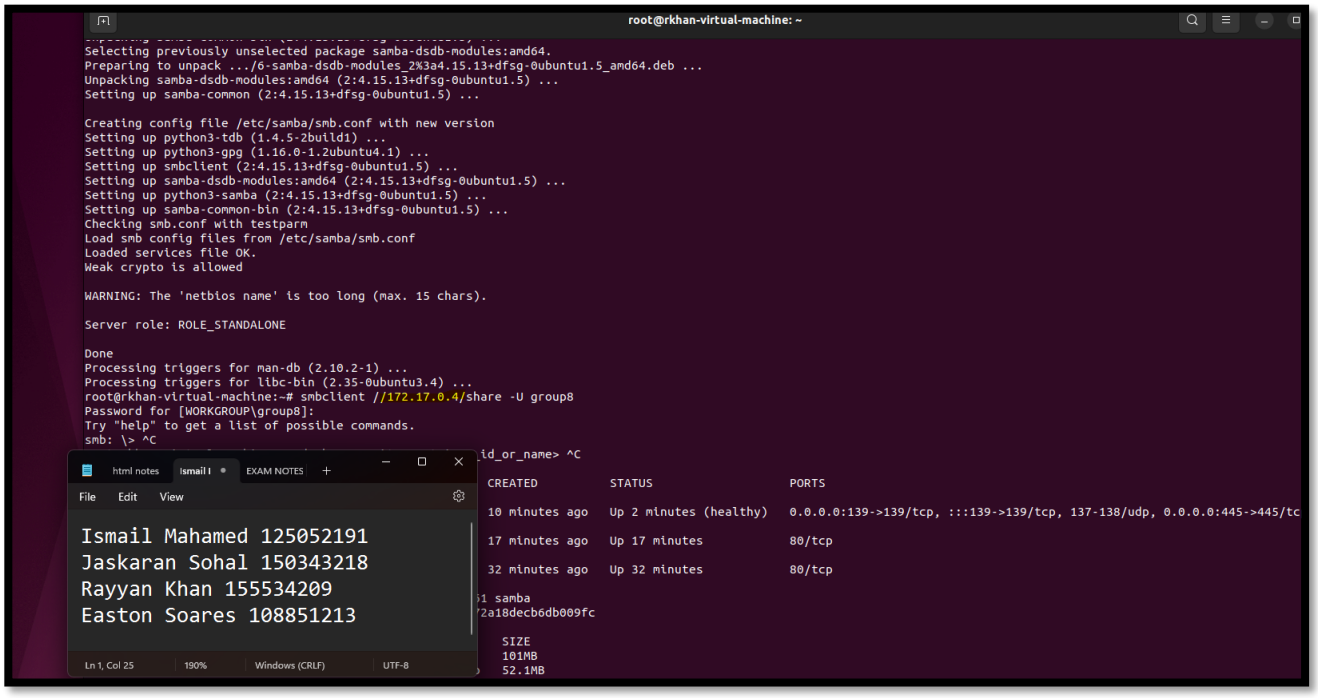


Figure 28: Docker E-Commerce Website

Industrial Practices

➢ IT Infrastructure and Hosting

As a team, we have diligently constructed a robust IT framework designed to cater to the multifaceted needs of our organization and clients. This infrastructure encompasses a wide array of components, including high-performance servers, secure data storage solutions, and comprehensive networking systems, all seamlessly integrated to ensure the highest levels of performance and reliability. Our commitment to state-of-the-art technology and continuous improvement positions us to handle substantial workloads, protect critical data, and provide an uninterrupted service experience.

Moreover, the hosting services we provide are tailored to support the dynamic requirements of our diverse clientele. The team's expertise in managing and maintaining these services guarantees optimal system uptime and efficient resource allocation. By leveraging virtualization technologies and cloud-based solutions, we offer scalable and flexible hosting environments. This adaptability is key in our pursuit to provide services that not only meet but exceed the expectations of those we serve. Through vigilant monitoring and proactive management, we maintain a hosting ecosystem that is both secure and capable of adapting to the ever-evolving landscape of IT demands.

➢ Automation

Automation in IT infrastructure is a transformative force, and as a team, we have embraced this innovation to streamline our operations and elevate our service offerings. Utilizing sophisticated configuration management tools and scripts, we have automated routine tasks, reducing the likelihood of human error and freeing our staff to focus on more complex, strategic initiatives. From automating server setups and network configurations to deploying updates and security patches, our infrastructure is designed to be resilient and self-sufficient. This has not only optimized our resource utilization but also enabled us to maintain a consistent and secure environment for our applications and services.

Further enhancing our automation capabilities, we've integrated monitoring and alerting systems that proactively identify and respond to potential issues before they escalate. These systems are configured to track the health of our servers, network performance, and security posture, ensuring that any anomalies are detected in real-time. Using automated scripts and predefined policies, we can swiftly address these alerts, often without the need for manual intervention. This level of automation extends into our security practices as well, where we employ tools to continuously scan for vulnerabilities and enforce compliance standards. The culmination of these efforts is a robust, dynamic IT ecosystem that supports our commitment to delivering uninterrupted, high-quality services to our clients.

➢ Fault Condition

Addressing fault conditions within IT infrastructure is a critical component of maintaining system integrity and availability. Our team has implemented a comprehensive strategy to monitor, detect, and respond to fault conditions as they arise. Using advanced diagnostic tools and alert systems, we can promptly identify issues ranging from system outages to security breaches. These tools are configured to monitor system performance metrics and log entries, flagging any anomalies that deviate from established patterns. This proactive stance allows us to anticipate and mitigate potential failures before they impact operations.

Once a fault condition is detected, our automated response protocols are activated. These include predefined scripts and processes that execute remediation actions, such as restarting services, isolating affected systems, and deploying patches. Our incident response team is also promptly alerted to ensure that more complex issues are addressed with the appropriate level of expertise and urgency. Post-event analyses are conducted to understand the root cause of the fault condition, leading to continuous improvements in our system architecture and response procedures. By learning from each incident, we strengthen our defenses and enhance our ability to maintain a resilient and reliable IT environment for our clients.

➢ Micro Segmentation Patterns

Micro-segmentation patterns within IT infrastructure are a critical component of network security strategy. Our team has been at the forefront of implementing these patterns, which involve subdividing the data center into distinct security segments down to the individual workload level. This allows for more granular control of security policies and helps in containing potential breaches within small perimeters, effectively reducing the overall attack surface.

Through the use of virtualization and cloud technologies, we have crafted a network design that supports flexible and dynamic micro-segmentation. The configurations, as evident in the network and security system screenshots provided, show how policies are enforced and managed. By assigning specific security rules to individual workloads, we can ensure that only legitimate traffic is allowed, while unauthorized access is blocked. This is further complemented by our sophisticated monitoring systems that continuously analyze traffic flow and detect suspicious patterns, enabling us to respond rapidly to any security threats.

In essence, micro-segmentation has allowed us to provide tailored security that adapts to the operational needs of each segment, ensuring that critical services are isolated and protected. Our team's approach not only secures the IT environment but also provides the flexibility needed to scale and adapt our security measures in line with the evolving landscape of cyber threats. This proactive and nuanced approach to security is a testament to our team's commitment to safeguarding our IT ecosystem.

➢ Industrial Practices

Industrial practices in IT cover a broad range of activities, from the development and implementation of software and systems to the maintenance and optimization of these assets. A critical aspect of industrial practices in IT is adhering to best practices and standards to ensure reliability, security, and performance. This encompasses everything from structured coding methodologies, rigorous testing procedures, deployment strategies, and continuous integration and delivery pipelines, to name a few.

In the context of IT operations and infrastructure, industrial practices may include the application of IT service management frameworks such as ITIL to streamline IT services, using project management

methodologies like Agile or Waterfall to guide system development, and employing security standards such as ISO 27001 to safeguard information assets. These practices are designed to promote efficiency, reduce risks, and ensure that IT services meet the needs of the business and its customers.

➢ Controls

1. **MA-2 Controlled Maintenance**:

   We have maintenance routines for updating and configuring various components, such as OpenVPN servers and firewalls. We ensure that these maintenance activities follow documented procedures. The use of specific configuration files (e.g., OpenVPN server/client configurations) and automation scripts (e.g., Ansible playbooks) in our lab contributes to controlled maintenance. Regular monitoring of logs, like those in Snort IDS or ModSecurity, helps verify the proper functioning of controls after maintenance actions.

2. **CM-2 Baseline Configuration**:

   To maintain consistency and reliability within our lab, we've established a baseline configuration for our systems. We document and maintain this configuration under strict control. We periodically review and update the baseline configuration based on organizational needs, specific circumstances, or when system components are installed or upgraded. This practice helps us ensure that our systems are always running with the correct and secure settings.

3. **CA-8 Penetration Testing**:

   Security is a top priority in our lab, and we conduct regular penetration testing to identify vulnerabilities that could be exploited by adversaries. Penetration testing is carried out by skilled teams to assess our systems' security comprehensively. We use this testing to validate vulnerabilities and assess our systems' resistance to attacks within defined constraints, such as time, resources, and skills. This is especially important when we transition to new technologies, ensuring that our security measures remain robust.

4. **AC-2 Account Management**:

   Effective account management is essential for maintaining a secure and organized lab environment. We define and document the types of accounts allowed and prohibited within our systems. Each account has designated managers, and we have specific prerequisites and criteria for group and role membership. We meticulously specify authorized users, group and role memberships, access authorizations, and associated attributes for each account. Account creation, modification, and removal follow our well-defined policy, procedures, prerequisites, and criteria. We also require approvals from designated personnel or roles for account creation.

5. **AC-17 Remote Access**:

   We take remote access seriously and establish clear guidelines to control it securely. We've documented usage restrictions, configuration/connection requirements, and implementation guidance for each type of remote access allowed in our lab. Before allowing any remote connections, we ensure proper authorization and validation. These measures help us maintain the integrity and security of our lab environment, preventing unauthorized access and potential threats.