Home Lab Project Portfolio

Alexander B. Glider
(267) 582-8971
alexander.b.glider@gmail.com
LinkedIn: alexander-glider
Date: October 2024

Home Lab Project – Overview

Project Objective:

The objective of this home lab project was to build a fully functional IT environment to develop and showcase practical skills in virtualization, networking, system administration, and cybersecurity. By using industry-standard tools and configurations, this project simulates real-world scenarios, providing hands-on experience in network management, service deployment, and security operations. The project highlights the ability to deploy, manage, and troubleshoot virtualized environments, secure networks, and critical IT services.

Project Scope

The home lab project was designed to provide hands-on experience with IT infrastructure, networking, and cybersecurity, simulating real-world scenarios. Key objectives included:

- **Disk Partitioning and Resource Allocation**: Partitioned a 1TB hard drive into dedicated sections for virtual machines, logs, and backups, ensuring efficient resource management and a structured storage system.
- **Comprehensive Networking Configurations**: Implemented and tested multiple networking modes—**Host-Only**, **NAT**, and **Bridged Networking**—to evaluate different security models and their impact on connectivity and isolation, providing flexibility for testing both internal and external network access scenarios.
- **Virtual Machine Setup and Management**: Installed and configured virtual machines using **VirtualBox**, tailored to host critical IT services with custom resource allocations (RAM, CPU, storage) to simulate production environments.
- **Deployment of Security and Network Monitoring Tools**: Integrated industry-standard tools such as **Wireshark**, **Tshark**, **Nmap**, **Metasploit**, and **OpenVPN** to monitor network traffic, perform security assessments, and analyze vulnerabilities, enabling a robust platform for security operations.
- **Service Management and Hosting**: Configured **Apache** as the primary web server to host services and applications, and used **Docker** to manage containerized environments, ensuring flexibility in deploying and testing applications in isolated environments.
- **Database Management and Log Aggregation**: Deployed **PostgreSQL** for secure database management, storing system and security logs for analysis. Configured **Graylog** and **Splunk** to aggregate and analyze logs from multiple sources, enhancing the project's monitoring and event management capabilities.
- **Secure Remote Access**: Set up **OpenVPN** to allow secure remote access to the internal network, ensuring that data transmission and client-server connections remain encrypted and protected, facilitating remote testing and management.

Core Components

1. **VirtualBox and Drive Partitioning**
   The project began by partitioning a 1TB hard drive into three separate partitions (E, F, and G), dedicated to hosting virtual machines and project files. VirtualBox (version 7.0.20) was installed and configured to create and manage virtual machines, which were stored on the E partition.

2. **Networking Configurations**
   To test different network access scenarios, three distinct networking modes were employed:

   - **Host-Only Networking**: Isolated VMs from external networks, ideal for internal testing.
   - **NAT (Network Address Translation)**: Allowed VMs to access external networks while remaining isolated from the host machine.
   - **Bridged Networking**: Enabled VMs to behave as if they were on the same network as the host, facilitating access to external services, such as log servers.

3. **Ubuntu Server VM Setup**
   An Ubuntu Server VM was installed with 4GB of RAM and 2 CPU cores. This VM was used as the main platform for running network and security monitoring tools. Custom IP configurations ensured reliable network connectivity for these tools.

4. **Tools and Services**

   - **Network Monitoring Tools**
     - **Wireshark/Tshark**: Installed for capturing and analyzing network traffic. These tools monitored various protocols (e.g., mDNS, SSDP) and SSH traffic for real-time insights.
     - **Nmap**: Deployed to scan the network and identify devices and services, highlighting potential vulnerabilities and network configurations.
   - **Penetration Testing Tools**
     - **Metasploit**: Installed via Snap for conducting penetration testing. The tool was successfully configured and tested to validate its capabilities.
   - **Security Information and Event Management (SIEM) Tools**
     - **Graylog**: Configured to aggregate and analyze log data from different sources, including SSH login attempts and network protocol logs. Its web interface was utilized for in-depth log analysis.
     - **Splunk**: Installed for real-time machine data analysis, enabling dashboard creation and event monitoring for system and security events.

- **Database Management**
  - **PostgreSQL**: Set up as the primary database engine for storing data, used in combination with log management tools to manage and store large datasets securely.
- **Web Server and Service Tools**
  - **Apache**: Installed and configured to host web services. The default Apache web server was successfully set up and tested, providing a platform for hosting web-based services and applications.
- **Containerization and Service Deployment Tools**
  - **Docker**: Installed to manage and deploy lightweight containers for running services in isolated environments, streamlining service management and testing.
- **VPN and Remote Access Tools**
  - **OpenVPN**: Configured to provide secure remote access to the internal network. This allowed encrypted VPN connections from external clients to the lab network, ensuring secure data transmission.

Conclusion

This home lab project showcases a strong foundation in IT infrastructure management, network configurations, and cybersecurity tool deployments. By utilizing a wide array of tools and services such as **Wireshark**, **Nmap**, **Graylog**, **Splunk**, **Apache**, and **OpenVPN**, the project highlights expertise in configuring, monitoring, and securing virtual environments.

The skills demonstrated in troubleshooting, system administration, and security analysis are directly applicable in real-world IT and cybersecurity roles. This project stands as a testament to the ability to design, implement, and manage complex technical environments with precision and efficiency.

Home Lab - Hard Drive Partitioning and VirtualBox Setup for Optimal Data Management

---

Primary Objective:

The goal was to create a structured and organized storage environment by partitioning the 1TB hard drive into three dedicated sections. Each partition—E for VirtualBox and VM storage, F for backups, and G for logs—was designed to isolate specific functions, ensuring efficient data management and streamlined workflows for the home lab setup. This approach enables clear separation of resources, simplifies system management, and enhances overall project organization.

Step 1: Partitioning the Hard Drive

The objective was to partition the 1TB hard drive into three separate drives to efficiently organize virtual machine files, backups, and logs. The drives were labeled as E, F, and G, each serving a distinct purpose.

This step involved GUI-based actions within Disk Management:

1. Opened **Disk Management** by right-clicking on the Start menu.
2. Selected the 1TB hard drive to be partitioned.
3. Right-clicked on the unallocated space and chose **New Simple Volume**.
4. Repeated the process to create three partitions:
   - **Drive E:** Main partition for VirtualBox installations and VM storage (500GB).
   - **Drive F:** For backup files (300GB).
   - **Drive G:** For logs and network traffic captures (130GB).
5. Verified the creation of the partitions in **This PC**.

These GUI steps ensured that the drives were properly partitioned with the intended sizes and labels.

No issues were encountered during the partitioning process. All steps proceeded smoothly.

Step 2: Installing VirtualBox on the E Drive

The objective was to install VirtualBox on the E drive to ensure that all program files and virtual machine data are stored in an organized and easily accessible partition dedicated to virtualization.

This step involved GUI-based actions:

1. Downloaded the latest version of VirtualBox (v7.0.20 r163906) from the official VirtualBox website.
2. Launched the installer and selected **Custom Installation**.
3. Set the installation path to **E:\VirtualBox** during the setup to ensure the program files are stored on the correct partition.
4. Completed the installation process, then opened VirtualBox.
5. Navigated to **File > Preferences > General > Default Machine Folder** and set the folder path to **E:\VMs** for storing all future virtual machine files.

These steps ensured that VirtualBox was installed on the correct partition, and all VMs would be stored in a dedicated folder on the E drive.

No issues were encountered during the installation process. VirtualBox installed correctly, and the default machine folder was set without any problems.

Step 3: Creating the Ubuntu Server Virtual Machine

The objective was to create an Ubuntu Server virtual machine in VirtualBox, allocating proper resources and ensuring the VM files are saved to the designated E drive.

This step involved GUI-based actions:

1. Opened VirtualBox and clicked **New** to create a new virtual machine.
2. Named the VM "Ubuntu Server".
3. Set the machine folder to **E:\VMs\Ubuntu Server\Ubuntu Server.vdi**.
4. Allocated **4GB of RAM** and **2 CPU cores** for the VM.
5. Downloaded the Ubuntu Server ISO from the official website and selected it during VM creation.
6. Mounted the ISO file under **Controller: IDE** to boot the VM from the Ubuntu Server image.
7. Set up a dynamically allocated hard disk with a size of **50GB**, with the file stored at **E:\VMs\Ubuntu Server\Ubuntu Server.vdi**.

These steps ensured that the virtual machine was created with the necessary resources and properly stored on the E drive.

No issues were encountered during the creation of the Ubuntu Server VM. All steps were successfully completed.

Step 4: Installing VirtualBox Guest Additions

The objective was to install VirtualBox Guest Additions to enhance the virtual machine's functionality by enabling features such as improved display settings, clipboard sharing, and seamless mouse integration.

This step involved GUI-based actions:

1. Started the Ubuntu Server VM.
2. Selected **Devices > Insert Guest Additions CD Image** from the VirtualBox menu.

Guest Additions did not work as expected. The main goal was to enable copy-paste functionality between the host and VM, but this feature was not functioning properly.

1. Attempted multiple troubleshooting steps, including:
   - Reinstalling Guest Additions.
   - Ensuring the necessary dependencies were installed on the Ubuntu Server VM.
   - Verifying that the **Shared Clipboard** setting was enabled in the VirtualBox settings.
2. Despite these efforts, Guest Additions did not fully resolve the issue. To address the copy-paste functionality, opted to use **SSH** for remote management of the VM. SSH allowed copying and pasting text between the host and VM, bypassing the issue with Guest Additions.

Step 5: Post-Installation Configuration

The objective was to ensure the virtual machine was fully functional by enabling virtualization in the host system's BIOS and configuring the appropriate network settings for communication between the VM and the host machine.

No commands were run for this section:

1. **Enabling Virtualization:**
   - Restarted the computer and accessed BIOS/UEFI by pressing **F2** during boot.
   - Navigated to **Advanced > CPU Configuration** and enabled **Virtualization**. Saved and exited BIOS.
2. **Configuring Network Settings:**
   - Initially used **NAT Networking** for basic communication between the VM and the host.

- Later switched to **Bridged Networking** to allow the VM to communicate directly with the host's local network.

No major issues were encountered during this process. Network communication worked as expected, first using NAT and later using Bridged Networking.

Conclusion

Partitioning the 1TB hard drive into separate drives for virtual machine files, backups, and logs allowed for better organization and efficient resource allocation. By installing VirtualBox on the E drive and configuring it to store VMs there, the setup supports a streamlined environment for managing virtual machines and related data, ensuring that resources are well-organized for the home lab project.

Home Lab - Configuring Host-Only Networking for Ubuntu VM

Primary Objective:

To configure and verify Host-Only networking between the Windows host machine and the Ubuntu VM, ensuring reliable communication over a private network (192.168.56.101). This setup isolates the VM from external networks while allowing direct access from the host, facilitating SSH connections and efficient remote management.

Step 1: Enabling VirtualBox Host-Only Network Adapter

The objective was to enable and configure the Host-Only Network Adapter in VirtualBox, ensuring that the Ubuntu VM can communicate with the host machine through a private network interface.

This step involved GUI-based actions:

1. **Open VirtualBox**
   Navigate to the VirtualBox main window.
2. **Access the Host Network Manager**
   From the top menu, click on **File** > **Host Network Manager**.
3. **Verify the Existence of Host-Only Adapter**
   In the Host Network Manager window, check if an adapter (typically named vboxnet0) is listed.
4. **Configure the Adapter Settings (if needed)**
   If the adapter exists, click on it to view and configure the settings.
   - Example settings:
     - **IPv4 Address**: 192.168.56.1
     - **IPv4 Network Mask**: 255.255.255.0 You may also configure DHCP settings here if needed.
5. **Create a New Host-Only Adapter (if none exists)**
   If no adapter is listed, click **Create**. A new adapter (vboxnet0) will be generated with default settings. You can modify the IP address or DHCP configuration as needed.
6. **Ensure Adapter is Active**
   Make sure the status of the Host-Only adapter is set to **Enabled**.

No issues were encountered during the configuration process. All steps proceeded smoothly.

Step 2: Configuring the Ubuntu VM's Network Adapter for Host-Only

The objective was to configure the Ubuntu VM to use the Host-Only Adapter, enabling communication between the host machine and the VM over the private 192.168.56.101 network.

This step involved GUI-based actions:

1. **Open VirtualBox and Access VM Settings**
   In the VirtualBox Manager, select the Ubuntu VM (stored at E:\VMs\Ubuntu Server\Ubuntu Server.vdi), then click on **Settings**.

2. **Navigate to the Network Tab**
   In the VM Settings window, select the **Network** tab.

3. **Configure Adapter 1 as Host-Only**
   a. Ensure **Adapter 1** is enabled by checking the box labeled **Enable Network Adapter**.
   b. In the **Attached to** dropdown, select **Host-Only Adapter**.
   c. Ensure that the correct host-only adapter is selected (typically vboxnet0).

4. **Verify Adapter Settings**
   Confirm that **Promiscuous Mode** is set to **Deny** and that **Cable Connected** is checked to simulate a connected network cable.

5. **Save Changes**
   Click **OK** to save the network adapter settings and close the VM settings window.

The Host-Only adapter did not appear as an option. This issue likely occurred because the adapter was either not correctly created in the previous step or VirtualBox didn't fully recognize it.

Since the Host-Only Adapter did not appear, revisited the **Host Network Manager** to ensure the adapter was created properly. Restarting VirtualBox after creating the adapter resolved the issue.

Step 3: Starting the Ubuntu VM and Configuring SSH

To enable SSH on the Ubuntu VM for remote access from the host machine, simplifying the management and configuration of the VM without needing to rely on the VirtualBox console.

1. **Powered on the Ubuntu VM** in VirtualBox.

2. **Installed OpenSSH Server**
   SSH is essential for remote access. To install the OpenSSH server, ran the following command inside the Ubuntu VM (bash):

   - sudo apt install openssh-server
   - This command installs the OpenSSH server package, which allows SSH access to the VM. It uses the apt package manager to download and install the necessary components.

3. **Verify SSH Service is Running**
   After installation, we confirmed that the SSH service was active by running (bash):
   - sudo systemctl status ssh
   - This command checks the current status of the SSH service, showing if it's active and running. It provides information about the service's state, helping to ensure that SSH is ready for remote connections.
4. **Test SSH Connection via localhost**
   To ensure that SSH was functioning correctly, we ran the following command to connect to the VM via localhost (loopback address) (bash):
   - ssh ubuntu@127.0.0.1
   - This command attempts to initiate an SSH connection to the local machine. Testing it this way ensures that SSH is working before connecting from the host machine.

The SSH service did not start automatically after installation, and local SSH connections were occasionally refused due to misconfigured firewall rules.

1. **Solution 1: Start SSH manually**
   Since the SSH service did not start automatically, started it manually (bash):
   - sudo systemctl start ssh
2. **Solution 2: Check UFW Firewall Settings**
   Checked if the Uncomplicated Firewall (UFW) was blocking SSH and allowed SSH through the firewall (bash):
   - sudo ufw allow ssh
3. After applying the rule, verified the status of UFW (bash):
   - sudo ufw status

After applying the firewall rule and manually starting the SSH service, the SSH connection was successfully established, allowing seamless communication between the host and the VM.

Step 4: Configuring the Network Interface on Ubuntu

The objective was to configure the Ubuntu VM's network interface (enp0s3) with a static IP address for the Host-Only network. This ensures reliable communication between the host machine and the Ubuntu VM over the private 192.168.56.101 network.

Commands Run During Configuration

1. **List Available Network Interfaces**
   To confirm that the Host-Only network interface (enp0s3) was recognized by the Ubuntu VM (bash):

- ip a
- This command displays all available network interfaces on the system, along with their IP addresses and statuses. This was used to check if the enp0s3 interface had been created and was ready for configuration.

2. **Edit the Netplan Configuration File**
   To assign a static IP address to the enp0s3 interface, edited the network configuration file using nano (bash):
   - sudo nano /etc/netplan/00-installer-config.yaml
   - Netplan is the default network configuration tool on Ubuntu. Editing this YAML file allows us to define network settings such as static IP addresses. In this case, we added the necessary configuration for the enp0s8 interface.
3. **Configure the Static IP for Host-Only Adapter**
   In the YAML file, added the following lines to assign the IP address 192.168.56.101 to the enp0s3 interface:

```
yaml

Copy code

network:

  ethernets:

    enp0s3:

      dhcp4: no

      addresses:

        - 192.168.56.101/24

  version: 2
```

This configuration assigns the static IP address 192.168.56.101 to the enp0s3 interface, with a subnet mask of /24. Disabling dhcp4 ensures that the interface will not attempt to obtain an IP address dynamically.

4. **Apply the Network Configuration**
   To apply the changes made to the Netplan configuration file (bash):
   - sudo netplan apply
   - This command applies the network settings defined in the Netplan configuration file. Once applied, the enp0s3 interface is configured with the static IP address.
5. **Verify the Configuration**
   After applying the changes, confirmed that the enp0s3 interface was correctly configured by running (bash):

- ip a
- Running ip a again verfied that the enp0s3 interface now had the IP address 192.168.56.101.

Encountered issues where the enp0s3 interface did not appear after booting the VM, and YAML syntax errors occurred in the Netplan configuration file due to incorrect indentation.

To resolve the issues, rebooted the Ubuntu VM to ensure the enp0s3 interface was correctly recognized and initialized. For the YAML syntax errors, reviewed the Netplan file carefully, ensuring all lines were properly indented with spaces instead of tabs. After correcting the indentation, applied the configuration using netplan apply and checked for any further errors.

Step 5: Testing the Host-Only Network Connection

The objective was to verify that the Host-Only network is properly configured, allowing the Windows host machine to communicate with the Ubuntu VM over the 192.168.56.101 network. Confirm this by using the ping command from the host and ensuring network connectivity.

Commands Run During Testing

1. **Ping the Ubuntu VM from the Host Machine**
   On the Windows host machine, opened a command prompt and ran the following command to test the connection to the Ubuntu VM (cmd):
   - ping 192.168.56.101
   - The ping command sends ICMP Echo Request packets to the specified IP address (in this case, the Ubuntu VM's static IP: 192.168.56.101). A successful response indicates that the VM is reachable from the host over the Host-Only network.

During testing, encountered an issue where the ping request timed out, which was likely caused by a firewall issue on either the host machine or the VM.

Firewall blocking ICMP traffic on Windows
The host machine was unable to ping the VM, suspected that Windows Defender Firewall was blocking ICMP (ping) requests. To resolve this, configured an inbound firewall rule on the host machine:

1. Opened **Windows Defender Firewall with Advanced Security**.
2. Navigated to **Inbound Rules** and created a new rule.
3. Selected **Custom Rule**, then **ICMPv4** for the protocol.
4. Allowed **Echo Request** from any IP address and applied the rule to the **Private** network profile.

It was possible that UFW on the Ubuntu VM was blocking ICMP traffic. To resolve this, allowed ICMP traffic by running (bash):

- sudo ufw allow proto icmp

After adding this rule, verified UFW's status:

- sudo ufw status

Confirmed that ICMP traffic was permitted.

With the above steps, successfully verified the host-only network connection by receiving ping responses from the Ubuntu VM.

Section 6: Troubleshooting Summary

Issues Encountered During Configuration

1. **No enp0s3 Network Interface**
    - When running ip a, the expected Host-Only network interface (enp0s8) was missing. This prevented the configuration of a static IP address for the Host-Only network.
    - The issue was resolved by:
        - Rechecking VirtualBox network settings to ensure that Adapter 1 was properly attached to the Host-Only Adapter (vboxnet0).
        - Rebooting the Ubuntu VM to ensure that the network adapter was recognized.
2. **YAML Indentation Error**
    - While editing the Netplan configuration file (/etc/netplan/00-installer-config.yaml), a YAML formatting error occurred. This caused netplan apply to fail due to incorrect indentation.
    - Resolution:
        - The YAML file was carefully reviewed for proper indentation. YAML requires strict adherence to space-based indentation, and tabs are not allowed.
        - After correcting the indentation, netplan apply was successful.
3. **Ping Request Timed Out**
    - When attempting to ping the Ubuntu VM from the Windows host, the request timed out. This indicated that ICMP (ping) traffic was being blocked, either by the Windows Defender Firewall or by the Ubuntu firewall (UFW).
    - Resolution:
        - On the Windows host machine, a custom inbound firewall rule was created to allow ICMP traffic (Echo Request) through Windows Defender Firewall.

    This rule was configured to apply to the private network profile, allowing pings to go through.

    - On the Ubuntu VM, the UFW firewall was configured to allow ICMP traffic. This was done by adding an allow rule for ICMP using the ufw command. After applying the rule, the ping test was successful.

4. **SSH Blocked by Firewall**
    - The SSH connection from the host machine to the Ubuntu VM was refused, suggesting that SSH traffic was being blocked.
    - Resolution:
        - On the host machine, we configured an inbound firewall rule to allow SSH traffic on port 22. This was done by creating a custom rule in Windows Defender Firewall, allowing TCP traffic on port 22 from any IP address and applying the rule to the private network profile.
        - After allowing SSH through the firewall, we successfully connected to the VM using SSH.

Conclusion

The Host-Only network setup was successfully completed after resolving the issues encountered with network interface recognition, YAML formatting, firewall rules, and SSH access. This setup allows isolated communication between the host machine and the Ubuntu VM over the 192.168.56.101 network, with SSH enabled for efficient remote management.

Home Lab - Configuring Network Address Translation (NAT) for Ubuntu VM

Primary Objective:

To configure Network Address Translation (NAT) on an Ubuntu virtual machine (VM) within a home lab environment. This setup will enable the VM to access external networks (such as the internet) while remaining isolated from the local network, ensuring both connectivity and enhanced security.

---

1. Overview of NAT Networking

To set up NAT (Network Address Translation) in VirtualBox for an Ubuntu Server VM, allowing the VM to access the internet while keeping it isolated from the host network.

This was done through the VirtualBox GUI. Here are the steps taken:

1. **Open VirtualBox**: Launch VirtualBox on the host machine.
2. **Select the VM**: Choose the Ubuntu Server VM from the list.
3. **Go to Settings**: Right-click on the VM and select **Settings**.
4. **Navigate to Network**: In the Settings window, click on the **Network** tab.
5. **Select Adapter 1**: Ensure **Adapter 1** is enabled and set to **NAT**.
6. **Save Changes**: Click **OK** to apply the settings.

No issues were encountered during this step.

---

2. Initial Setup

The objective was to configure VirtualBox for the Ubuntu Server VM, ensuring the correct resource allocation and setting up the network adapter to use NAT for internet connectivity.

This was done via the VirtualBox GUI, here are the steps:

1. **Open VirtualBox**: Launch VirtualBox.
2. **Select the VM**: Choose the Ubuntu Server VM from the list.
3. **Go to Settings**: Right-click on the VM and select **Settings**.
4. **Configure Resources**:
    - **RAM**: Set to 4GB in the **System > Motherboard** tab.
    - **CPU**: Set to 2 Cores in the **System > Processor** tab.

    - **Storage Location**: The VM was set to use the disk located at E:\VMs\Ubuntu Server\Ubuntu Server.vdi.

5. **Configure Network Adapter**:
    - Navigate to the **Network** tab.
    - Under **Adapter 1**, select **NAT** as the network mode.

No issues were encountered during this step.

---

3. NAT and Port Forwarding

The objective was to configure port forwarding in VirtualBox's NAT mode, allowing external access to services running on the Ubuntu Server VM (e.g., SSH) from the host machine.

This step was done via the VirtualBox GUI. Here are the detailed steps:

1. **Open VirtualBox**: Launch VirtualBox.
2. **Select the VM**: Choose the Ubuntu Server VM from the list.
3. **Go to Settings**: Right-click on the VM and select **Settings**.
4. **Navigate to Network**:
    - Under **Adapter 1**, click on **Advanced**.
    - Click on **Port Forwarding**.
5. **Add a New Rule**:
    - **Name**: SSH (or another service, as needed).
    - **Protocol**: TCP.
    - **Host IP**: Leave blank (binds to all interfaces on the host).
    - **Host Port**: Set to 2222 (or any unused port on the host).
    - **Guest IP**: Leave blank (binds to the VM's default NAT IP).
    - **Guest Port**: Set to 22 (for SSH service).
6. **Save the Rule**: Click **OK** to save the port forwarding rule.
7. **Start the VM**: Start the VM to apply the changes.

Encountered an issue where attempting to connect via SSH resulted in a "Connection Refused" error.

1. **Firewall Configuration**: The issue was resolved by adjusting the Windows Defender firewall settings on the host machine:

- Opened **Windows Defender Firewall**.
- Clicked on **Advanced settings**.
- In **Inbound Rules**, created a new rule allowing TCP traffic on port 2222.

2. After the firewall rule was applied, the SSH connection worked as expected.

---

4. Setting Up SSH with Key-Based Authentication

The objective was to set up SSH key-based authentication on the Ubuntu Server VM for secure access from the host machine, replacing password-based authentication.

1. **Generate SSH Key on the Host Machine**:
    - On the host machine, opened PowerShell or Command Prompt and ran the following command to generate a new SSH key pair (css):
    - ssh-keygen -t rsa -b 4096
        1. Generates a new SSH key pair using the RSA algorithm. The -t rsa option specifies the type of key to be RSA, and the -b 4096 option sets the key length to 4096 bits, providing strong encryption for secure communication.
    - Saved the key pair to C:\Users\Alex Glider\.ssh\ when prompted. No passphrase was set.
    - The public key was saved as id_rsa.pub, and the private key as id_rsa.
2. **Copy the SSH Public Key to the Ubuntu VM**:
    - Logged into the Ubuntu VM using the password-based SSH connection (css):
        1. ssh ubuntu@127.0.0.1 -p 2222
    - Created the .ssh directory on the VM and set the appropriate permissions (bash):
        1. mkdir -p ~/.ssh
            - Creates the .ssh directory in the user's home folder. The -p option ensures any missing parent directories are created and avoids errors if the directory already exists.
        2. chmod 700 ~/.ssh
            - Sets the permissions for the .ssh directory so that only the owner has read, write, and execute access, ensuring the directory is secure for storing SSH keys.
    - Displayed the contents of the public key on the host machine (bash):
        1. cat ~/.ssh/id_rsa.pub
            - Displays the contents of the id_rsa.pub file, which is the public SSH key stored in the .ssh directory. This key can be shared with remote servers to enable secure, key-based authentication.

- Pasted the public key into the VM's authorized_keys file (bash):

    1. echo "paste-public-key-here" >> ~/.ssh/authorized_keys
        - Appends the provided public SSH key to the authorized_keys file in the .ssh directory. This allows the corresponding private key to be used for SSH authentication on the system. The >> ensures that the key is added without overwriting the existing contents of the file.
    2. chmod 600 ~/.ssh/authorized_keys
        - Sets the permissions of the authorized_keys file so that only the owner can read and write to it. This ensures the file is secured, preventing other users from accessing or modifying the file, which is crucial for maintaining the security of SSH key-based authentication.

3. **Enable SSH Key-Based Authentication on the VM**:

    - Opened the SSH configuration file on the Ubuntu VM (bash):

        1. sudo nano /etc/ssh/sshd_config

    - Ensured the following settings were enabled (yaml):

        1. PubkeyAuthentication yes
        2. PasswordAuthentication no

    - Saved the changes and restarted the SSH service (bash):

        1. sudo systemctl restart ssh

4. **Test SSH Key-Based Authentication**:

    - Logged out of the current session and tested key-based authentication (css):

        1. ssh ubuntu@127.0.0.1 -p 2222

    - The SSH login succeeded without prompting for a password.

No issues were encountered during the setup of SSH key-based authentication.

---

5. Setting Up Firewall Rules on Host Machine

The objective was to configure Windows Defender Firewall on the host machine to allow SSH connections on the port used for port forwarding (2222).

This was done via the GUI, here are the steps:

1. **Open Windows Defender Firewall**:

    - Open **Control Panel** on the host machine.

    - Navigate to **System and Security** > **Windows Defender Firewall**.

2. **Go to Advanced Settings**:

- On the left-hand side, click on **Advanced Settings** to open the firewall's advanced configuration.

3. **Create a New Inbound Rule**:
    - In the **Inbound Rules** section, click **New Rule** on the right-hand panel.
4. **Configure the Rule**:
    - **Rule Type**: Select **Port**.
    - **Protocol**: Choose **TCP**.
    - **Specific Port**: Enter **2222** (the port used for SSH port forwarding).
    - **Action**: Choose **Allow the connection**.
    - **Profile**: Select all profiles (Domain, Private, Public) to ensure the rule applies regardless of the network type.
5. **Save the Rule**:
    - Name the rule **Allow SSH** and save it.
    - Ensure the rule is active and listed under the **Inbound Rules**.

No issues were encountered when creating the firewall rule.

---

6. Troubleshooting Summary

1. **Port Forwarding Rules Not Working**:
    - Initially, when attempting to SSH into the VM, encountered a "Connection Refused" error, even after configuring port forwarding.
2. **Network Interface Not Visible in Ubuntu VM**:
    - The enp0s3 interface was missing from the Ubuntu VM. Only lo, enp0s8, and docker0 were visible, which limited network connectivity options.

Resolution Steps:

1. **Resolving the Port Forwarding Issue**:
    - The issue was traced back to the Windows Defender firewall blocking incoming traffic on port 2222.
    - **Resolution**: Created an inbound rule in Windows Defender Firewall to allow TCP traffic on port 2222, as detailed in the previous section.
2. **Resolving the Missing Network Interface Issue**:
    - The enp0s3 interface was not visible due to incorrect network adapter settings.

- **Resolution**:
    1. Shut down the VM.
    2. Opened **VirtualBox** and went to **Settings > Network**.
    3. Ensured **Adapter 1** was set to **NAT**.
    4. Restarted the VM, and the enp0s3 interface appeared after rebooting.

Conclusion

Successfully configured NAT in VirtualBox for the Ubuntu Server VM, enabling SSH access via port forwarding, and setting up secure key-based authentication enhances both the functionality and security of the system. This setup allows seamless external network access while maintaining isolation from the local network. By implementing secure authentication practices and troubleshooting any issues along the way ensures a reliable and accessible system configuration that adheres to best practices for virtualized environments.

Final Working Configuration:

1. **NAT Networking**: The VM was configured to use NAT for internet connectivity through the host machine.
2. **Port Forwarding**: SSH port forwarding was successfully set up, allowing access to the VM from the host machine via port 2222.
3. **SSH Key-Based Authentication**: Secure access to the VM was established using SSH key-based authentication, eliminating the need for password-based login.
4. **Firewall Configuration**: The host machine's firewall was configured to allow SSH traffic on port 2222, enabling external access.
5. **Network Interface Resolution**: The missing enp0s3 interface issue was resolved, allowing full network functionality within the VM.

With the successful setup of NAT, port forwarding, SSH key-based authentication, and firewall configuration, the VM is now fully operational and accessible securely from the host machine.

Index

Home Lab - Configuring Bridged Networking for Ubuntu VM

Primary Objective:

To configure Bridged Networking for an Ubuntu VM in VirtualBox, allowing the VM to act as a fully integrated device on the same network as the host machine. This setup will enable the VM to obtain its own IP address from the network's DHCP server, facilitating direct communication with other devices on the network, including the host machine, as well as providing access to external networks. The goal is to establish a seamless network environment that mimics real-world scenarios, making it ideal for advanced testing, network monitoring, and services like OpenVPN, Graylog, and web servers within the home lab project. This configuration plays a critical role in creating a realistic infrastructure for SOC development and testing.

Introduction to Bridged Networking

Pre-Configuration Checklist

Verify that all system and network prerequisites were in place before switching to bridged networking to avoid potential issues later.

Commands and Configuration Steps:

1. **Check VirtualBox settings:**
   Ensured that the VM was configured with the appropriate network adapter.
   - Navigating to *Settings* > *Network* in VirtualBox confirming the adapter was set to *Bridged Adapter*.
2. **Check network connection (Wi-Fi or Ethernet):**
   Verify that the system was connected to a functioning network.
   - Commands used (bash):
     - nmcli device status
     - This command shows the status of network interfaces and confirms a working connection before switching network modes
3. **Identify available interfaces on the VM:**
   Before configuration, check for network interfaces like enp0s8 (for bridging) and enp0s3 (initially missing).
   - Commands used (bash):
     - ip addr
     - This lists all interfaces and their assigned Ips.

Issues Encountered:

- **Missing interface (enp0s3):**
  enp0s3, which was expected to be visible, was missing, while enp0s8 was functioning. This required further configuration to proceed.

Resolution Steps:

- **Proceed with enp0s8:**
  Since enp0s8 was active, it was used for bridging. Further troubleshooting for enp0s3 was postponed to focus on ensuring enp0s8 worked correctly.

Bridged Adapter Setup in VirtualBox

The objective was to switch the VM's networking mode from NAT to Bridged Adapter to allow the VM to behave like a physical device on the same network as the host, resolving issues with accessing services such as Graylog.

Commands and Configuration Steps:

1. **Switch from NAT to Bridged Adapter:**
    - In VirtualBox, *Settings* > *Network* > *Adapter 1* and changed *Attached to* from *NAT* to *Bridged Adapter*.
    - Selected the Wi-Fi as the network interface for the host system.
2. **Check adapter status in Ubuntu VM:**
   After switching to bridged networking, verified that the interface was active on the VM.
    - Commands used (bash):
        - ip addr
        - This confirmed the VM had an IP assigned from the same subnet as the host and that the correct interface (enp0s8) was visible.

Issues Encountered:

- **Missing enp0s3 interface:**
  The expected enp0s3 interface was not visible after switching to Bridged Adapter. Only enp0s8 was detected, leading to further troubleshooting.

Resolution Steps:

- **Use enp0s8:**
  Since enp0s8 was detected and working, it was chosen as the active interface for bridged networking. No further attempts were made to activate enp0s3 once enp0s8 was functioning properly.

Network Interface Configuration on the Ubuntu VM

The objective was to configure the network interface (enp0s8) and bridge interface (br0) on the Ubuntu VM to ensure proper connectivity using bridged networking.

Commands and Configuration Steps:

1. **Modify the Netplan configuration:**
   The network configuration file was updated to reflect the use of enp0s8 and the bridge interface br0.
   - Commands used (bash):
     - sudo nano /etc/netplan/01-netcfg.yaml
     - Configuration (yaml):

```
network:
 version: 2
 renderer: networkd
 ethernets:
  enp0s8:
   dhcp4: no
 bridges:
  br0:
   interfaces: [enp0s8]
   addresses: [192.168.1.155/24]
   gateway4: 192.168.1.1
   nameservers:
    addresses:
     - 8.8.8.8
     - 8.8.4.4
```

   - **Explanation:**
     - enp0s8: The physical network interface used for bridging.
     - br0: The bridge interface, configured to use the enp0s8 interface and assigned a static IP address.
2. **Apply the changes:**
   Once the configuration was edited, the changes were applied with the following command:

- Commands used (bash):
    - sudo netplan apply
    - This applied the new network configuration and ensure that the bridge interface (br0) is now active with the correct IP address.

Issues Encountered:

- **Configuring IP addresses:**
  There was some confusion with the IP address initially assigned to br0. Several addresses were seen during testing, such as 192.168.1.155 and 192.168.1.44, which required clarification.

Resolution Steps:

- **Set static IP for br0:**
  After observing dynamic IP assignments, a static IP (192.168.1.155/24) was set in the Netplan configuration to ensure stability and avoid confusion during testing.

Testing Network Connectivity

The objective was to verify that the bridged networking configuration was successful and that the VM could communicate with other devices on the same network as the host.

Commands and Configuration Steps:

1. **Verify IP address of the bridge interface (br0):**
   Confirm that the bridge interface (br0) was assigned the correct static IP address after applying the Netplan changes.
    - Commands used (bash):
        - ip addr show br0
        - This ensured that the bridge interface had the expected IP address (192.168.1.155), confirming the successful application of the Netplan configuration.
2. **Ping test:**
   Test the connectivity between the Ubuntu VM and the host network by sending pings to various devices.
    - Commands used (bash):
        - ping 192.168.1.1   # Ping the default gateway (router)
        - ping 8.8.8.8   # Ping an external DNS server to test internet connectivity
        - ping 192.168.1.100 # Ping another device on the local network (host or another machine)

        - This verified the VM's ability to communicate with both the local network and external internet resources.

3. **Check routing and DNS resolution:**
   After configuring bridged networking, it was important to verify that the VM could resolve domain names and had proper routing to external networks.

   - Commands used (bash):
     - route -n
       - route -n: Check routing table to ensure the gateway was properly configured.
     - dig google.com
       - dig google.com: Confirm that DNS resolution was working, meaning the VM could access the internet.

Issues Encountered:

- **IP Address Changes:**
  During the testing phase, different IP addresses were observed (192.168.1.155 and 192.168.1.44), which initially caused some confusion regarding network configuration stability.

Resolution Steps:

- **Setting a Static IP Address:**
  After observing fluctuating IP addresses, a decision was made to assign a static IP to the bridge interface (br0), ensuring a stable connection for future tests.

Troubleshooting Section

Issues Encountered and Troubleshooting Steps:

1. **Network Interface enp0s3 Not Visible:**
   - **Issue:**
     The expected interface enp0s3 was not available in the ip addr output after switching to bridged networking. Instead, enp0s8 appeared, which was unexpected and led to initial confusion.
   - **Diagnostic Steps:**
     - Used the following command to verify available interfaces (bash):
       - ip addr
       - Confirmed that enp0s3 was missing but enp0s8 was present.
       - Decided to proceed with enp0s8 for bridging since it was functional.

Outcome: This interface was successfully used for the bridge configuration without further issues. The missing enp0s3 was noted but did not impede progress.

2. **IP Address Confusion with Bridged Adapter:**
   - **Issue:**
     During testing, the VM's IP address fluctuated between 192.168.1.155 and 192.168.1.44, which caused difficulty in determining the correct network configuration.
   - **Diagnostic Steps:**
     - Verified the assigned IP addresses by using (bash):
       - ip addr show br0
       - Observed that the IP address was dynamically assigned by DHCP, resulting in changes during testing. This caused confusion, especially when trying to confirm network connectivity.
     - **Corrective Action:**
       - Decided to assign a static IP address to the bridge interface (br0) to avoid dynamic IP changes (yaml):

```
network:
 version: 2
 renderer: networkd
 bridges:
  br0:
   interfaces: [enp0s8]
   addresses: [192.168.1.155/24]
   gateway4: 192.168.1.1
   nameservers:
    addresses:
     - 8.8.8.8
     - 8.8.4.4
```

- Applied the new configuration with (bash):
  - sudo netplan apply

Outcome: After assigning a static IP address, the network became stable, and connectivity issues were resolved.

3. **OpenVPN Issues After Switching to Bridged Networking:**

    - **Issue:**
      OpenVPN initially failed to start correctly after switching to bridged networking, causing the service to be unreachable.

    - **Diagnostic Steps:**

            - Checked OpenVPN service status using (bash):

                - sudo systemctl status openvpn

                - Error logs indicated issues with the networking setup, particularly with the interface and IP range assigned for OpenVPN.

        - **Corrective Action:**

            - Modified OpenVPN's configuration to accommodate bridged networking, specifically by adjusting the tap0 and server-bridge settings (bash):

                - dev tap0
                - server-bridge 192.168.1.100 255.255.255.0 192.168.1.200 192.168.1.250

            - Restarted the OpenVPN service with (bash):

                - sudo systemctl restart openvpn

            - Verified that the service was active and running (bash):

                - sudo systemctl status openvpn

Outcome: Once the correct interface and server-bridge settings were applied, OpenVPN started successfully and bridged networking was confirmed to be working with the VPN.

4. **Bridge Interface Misconfiguration:**

    - **Issue:**
      The bridge interface (br0) occasionally showed misconfigured IP ranges during initial attempts to connect external services (e.g., Graylog).

    - **Diagnostic Steps:**

            - Used the following command to check the routing table and network settings (bash):

                - route -n

                - Found discrepancies between the expected and actual IP ranges assigned to the bridge interface.

        - **Corrective Action:**

        - Ensured the Netplan configuration was correct and re-applied changes (bash):
            - sudo netplan apply
        - Restarted the network service to refresh interface settings (bash):
            - sudo systemctl restart systemd-networkd

Outcome: After restarting the network services, the bridge interface worked with the correct IP range, and external services like Graylog were accessible again.

Summary of Troubleshooting Process:

- Used diagnostic commands (ip addr, systemctl status openvpn, route -n) to identify issues with missing interfaces, dynamic IPs, and service configurations.
- Applied corrective actions including setting static IP addresses, modifying OpenVPN configuration, and restarting network services to ensure that bridged networking was stable and operational.

Testing External Services

To confirm that external services like Graylog and OpenVPN were accessible and functional after switching the VM to bridged networking.

Commands and Configuration Steps:

1. **Graylog Testing:**
    - After configuring bridged networking, you needed to ensure Graylog was accessible via the web interface using the VM's new IP address.
    - Commands used:
        - Confirm Graylog service status (bash):
            - sudo systemctl status graylog-server
            - This ensured the Graylog service was running and ready to accept connections.
        - Access the Graylog web interface:
          Open a browser on the host machine and navigate to the VM's IP address:
            - http://192.168.1.155:9000
            - This confirmed that the Graylog web interface was accessible from the host machine.
2. **OpenVPN Testing:**

- After reconfiguring OpenVPN for bridged networking, you tested the ability to connect to the VPN using the updated settings.
- Commands used:
  - Verify OpenVPN service status (bash):
    - sudo systemctl status openvpn
    - This ensured that the OpenVPN service was running properly after the configuration changes.
  - Check OpenVPN logs for connection details (bash):
    - sudo journalctl -u openvpn
    - This command enabled the review of the OpenVPN logs for any connection attempts or errors.
  - Attempt to connect to OpenVPN from the host machine using the following command (bash):
    - sudo openvpn --config client.ovpn
    - Purpose: Establish a VPN connection from the host machine to confirm the service was functional.

3. **Ping External Services:**
   - To further verify connectivity, you tested the ability to ping external services from the Ubuntu VM after switching to bridged networking.
   - Commands used:
     - Ping Graylog web interface (bash):
       - ping 192.168.1.155
       - Purpose: Confirm connectivity between the host machine and the VM running Graylog.
     - Ping Google DNS to verify external internet access (bash):
       - ping 8.8.8.8
       - Purpose: Ensure that the VM had external internet access through bridged networking.

Issues Encountered:

- **Graylog Web Interface Access:**
  Initially, the Graylog web interface was not accessible after switching to bridged networking due to a misconfiguration in the network interface settings.

- **OpenVPN Connection Issues:**
  OpenVPN did not immediately work with the new bridged network configuration, failing to establish connections due to incorrect settings for the bridge.

Resolution Steps:

1. **Graylog:**
    - **Resolution:** Checked the Graylog service logs and confirmed that the issue was related to the VM's IP address not matching the configuration. After setting a static IP and restarting the Graylog service, the web interface became accessible.
2. **OpenVPN:**
    - **Resolution:** Adjusted OpenVPN's configuration to include tap0 and server-bridge settings. Restarted the service and confirmed successful connections using the OpenVPN client on the host machine.

Outcome: Both Graylog and OpenVPN were successfully tested after switching to bridged networking. The services were accessible from the host machine and external clients as expected.

Conclusion

To summarize the overall benefits of switching to bridged networking and reflect on the lessons learned during the setup process, particularly how it resolved access issues with services like Graylog and OpenVPN.

Summary of Benefits:

- **Resolved Service Access Issues:**
  Switching to bridged networking successfully resolved the issue where services like Graylog were unreachable over NAT. With bridged networking, the VM behaves as if it is on the same local network as the host machine, making it possible to access services directly via their IP addresses.
- **Improved Network Visibility:**
  The bridged adapter allowed the VM to obtain an IP address from the local network's DHCP server, ensuring that it could communicate directly with other devices on the same network without the limitations imposed by NAT. This was especially useful for accessing the Graylog web interface and using OpenVPN.
- **Stable Connectivity with Static IP Assignment:**
  Configuring a static IP address for the bridge interface (br0) ensured network stability, which was crucial for testing and accessing services consistently. This prevented the confusion caused by dynamic IP assignments seen earlier in the project.

Lessons Learned:

1. **Dealing with Missing Interfaces:**
   The unexpected absence of enp0s3 required adaptability. By switching to enp0s8, the

configuration was successfully completed, showing the importance of flexibility when encountering missing or misconfigured interfaces.

2. **Configuring Network Bridges:**
   Understanding how to configure a bridge (br0) using Netplan, and how to assign static IP addresses, proved essential for maintaining reliable connectivity. This knowledge will be valuable for future projects involving complex networking setups.
3. **Service Configuration Adjustments:**
   Switching to bridged networking required reconfiguring services like OpenVPN to use tap0 and the correct server-bridge settings. This showed how network changes can affect other services and the importance of adjusting configuration files accordingly.
4. **Troubleshooting Networking Issues:**
   Encountering and resolving issues such as fluctuating IP addresses and service connection failures provided hands-on troubleshooting experience. Using diagnostic commands like ip addr, systemctl status, and journalctl to analyze logs and resolve network issues was invaluable.

By configuring Bridged Networking for the Ubuntu VM in VirtualBox, the VM has been successfully integrated into the same network as the host machine, enabling it to act as an independent device. This configuration allowed the VM to obtain an IP address directly from the network's DHCP server, ensuring seamless communication with other network devices.

Throughout the home lab project, this setup has been instrumental in facilitating real-world testing scenarios, such as running services like OpenVPN, Graylog, and web servers. The bridged network environment has proven essential for simulating enterprise-level infrastructure, allowing for robust testing, monitoring, and system management in preparation for SOC development. The experience has provided valuable insights into configuring and managing network environments, critical for future cybersecurity and IT projects.

Index

Home Lab – Network Traffic Analysis with Wireshark & Tshark

Primary Objective:

To configure and utilize Wireshark and Tshark within the home lab environment to capture, analyze, and interpret network traffic. This setup will allow for the monitoring of data packets in real-time, providing valuable insights into network behavior, protocol usage, and potential security issues. Wireshark's GUI and Tshark's command-line interface offer comprehensive tools for analyzing network activity, which is essential for testing, troubleshooting, and enhancing the network infrastructure within the home lab, especially in the context of SOC development and cybersecurity practices.

---

1. Installing Wireshark and Tshark on Ubuntu Server

The goal of installing both Wireshark and Tshark was to provide the ability to capture and analyze network traffic in the home lab environment.

- **Wireshark** was installed as the primary graphical tool, intended for visually inspecting and analyzing network traffic due to its powerful GUI interface.
- **Tshark**, on the other hand, was chosen for its command-line capabilities, making it ideal for environments where a graphical interface isn't available or practical, such as on the Ubuntu Server VM.

Initially considered using Wireshark for network traffic analysis. However, since the Ubuntu Server VM lacked a graphical user interface (GUI), was unable to run Wireshark. This limitation led to **Tshark**, the command-line version of Wireshark, which enabled the capture and analysis of network packets directly from the terminal.

Installing both tools, maintained flexibility in traffic analysis, using **Tshark** for automated and scriptable traffic captures in the server environment, while reserving **Wireshark** for future use in environments where graphical analysis is feasible.

Commands Run During Installation:

1. **Update the package list (bash)**:
    - sudo apt update
    - **Purpose**: Ensures that the system's package manager has the latest list of available software. This step is necessary before installing any new packages to avoid installing outdated versions or missing dependencies.
2. **Install Wireshark and Tshark (bash)**:
    - sudo apt install wireshark tshark

- **Purpose**: Installs both Wireshark (the graphical tool) and Tshark (the command-line tool). Installing both ensures the ability to analyze traffic in a GUI for deeper inspection or from the terminal for quick or automated tasks.

3. **Configure Wireshark to allow non-root users to capture packets (bash)**:

    - sudo dpkg-reconfigure wireshark-common
    - **Purpose**: Changes the configuration to allow non-root users to capture network traffic. This is a security measure to avoid needing to run Wireshark as root, which could expose the system to unnecessary risks.
    - **Note:** At this point, after attempting to configure Wireshark, it became clear that the lack of a graphical interface in the Ubuntu Server VM prevented Wireshark from running. This realization led to the decision to use **Tshark** for all packet capturing and analysis tasks, as it could be run directly from the command line without requiring a GUI.

4. **Add the user to the Wireshark group (bash)**:

    - sudo usermod -aG wireshark ubuntu
    - **Purpose**: Adds the user ubuntu to the wireshark group. This grants the user permission to capture packets without needing root access, as packet capture usually requires elevated privileges. Although this group is named after Wireshark, it is also required for **Tshark**, as both tools share the same underlying packet capturing permissions. By adding the user to this group, it enabled **Tshark** to capture network traffic in a non-root environment.

5. **Reboot the system (bash)**:

    - sudo reboot
    - **Purpose**: Ensures that the changes, such as the user being added to the Wireshark group, take effect. Rebooting is necessary to apply group membership changes and ensure the configuration updates are live.

Issues Encountered During Installation:

1. **Inability to Run Wireshark Due to Lack of GUI:**

    - **Note:** At this point, after attempting to configure Wireshark, it became clear that the lack of a graphical interface in the Ubuntu Server VM prevented Wireshark from running. This realization led to the decision to use **Tshark** for all packet capturing and analysis tasks, as it could be run directly from the command line without requiring a GUI.

2. **Non-root Users Unable to Capture Packets**:

- After installing Tshark, the ubuntu user was unable to capture traffic. This was due to insufficient permissions, as packet capturing is usually restricted to root users or specific user groups.

3. **Network Interfaces Not Recognized**:

   - During the initial setup, the expected network interface enp0s3 wasn't showing up in Tshark, which led to difficulties capturing traffic.

Resolution Steps:

1. **Resolving the Non-root Access Issue**:

   - Resolved the issue by adding the ubuntu user to the wireshark group using the following command (bash):

     - sudo usermod -aG wireshark ubuntu

   - After rebooting, the user was able to capture traffic using **Tshark** without needing root privileges.

2. **Resolving the Network Interface Issue**:

   - The missing network interface (enp0s3) issue was resolved by switching the VM's networking configuration from NAT to Bridged Adapter. Once this change was made in VirtualBox, the enp0s8 interface was recognized, and traffic could be captured successfully.

---

2. Verifying Wireshark and Tshark Installation

The objective here was to confirm that both Wireshark and Tshark were correctly installed and functional on the Ubuntu Server. This step was essential to ensure that both tools were ready for capturing and analyzing network traffic after installation, without any additional configuration issues.

Commands Run During Verification:

1. **Verify Wireshark Version (bash)**:

   - wireshark --version

   - **Purpose**: This command checks if Wireshark is installed properly by displaying the version information. Successful execution of this command confirms that Wireshark is accessible and functioning.

2. **Verify Tshark Version (bash)**:

   - tshark --version

- **Purpose**: Similar to the Wireshark check, this command verifies that Tshark is installed and ready for use by showing its version. It's important to confirm that Tshark can be run from the terminal without issues.

3. **List Available Network Interfaces (bash)**:

- tshark -D
- **Purpose**: This command lists all the available network interfaces on the system that can be used for capturing traffic. It's essential to verify that the expected network interfaces (such as enp0s8) are recognized by Tshark, as this indicates that the tool is properly integrated with the system's networking.

Issues Encountered During Verification:

1. **Interface enp0s3 Missing**:

- The enp0s3 interface, which was expected to be available for capturing traffic, was not recognized by Tshark during the verification process.

How These Issues were Resolved:

1. **Adjusting Network Configuration**:

- Investigated the VirtualBox network settings and realized that the interface configuration was set to NAT, which limited the availability of certain network interfaces. To resolve this, switched the VM's networking to **Bridged Adapter**. After doing so, the enp0s8 interface became available for packet capture in Tshark.

2. **Final Check After Reconfiguration**:

- Re-ran the following command to confirm that enp0s8 was now available (bash):
- tshark -D
- **Result**: The enp0s8 interface was successfully listed, confirming that the network configuration changes had resolved the issue.

At this point, Tshark was verified to be installed and functional, and the expected network interfaces were available for capturing traffic.

---

3. Capturing Traffic with Tshark

The goal of capturing traffic with Tshark was to monitor and analyze specific types of network traffic (such as SSH and service discovery protocols like mDNS and SSDP) in a lightweight, command-line environment. This allowed for real-time traffic capture without needing the graphical interface provided by Wireshark.

Commands Run for Capturing Traffic:

1. **Capture SSH Traffic (bash)**:
   - tshark -i enp0s8 -f "tcp port 22" -w ssh_capture.pcap
   - **Purpose**: Captures all traffic on **TCP port 22**, which is used for SSH communication. The output is saved in a .pcap file for further analysis in Wireshark. This command was particularly useful in monitoring SSH traffic between the host machine and the VM.
2. **Capture mDNS and SSDP Traffic (bash)**:
   - tshark -i enp0s8 -Y "mdns || ssdp"
   - **Purpose**: Filters and captures traffic related to **mDNS** (Multicast DNS) and **SSDP** (Simple Service Discovery Protocol). These protocols are typically used by devices to announce their services on a local network. Capturing this traffic was useful for understanding service discovery within the home lab network.
3. **Save Captured Traffic to a File (bash)**:
   - tshark -i enp0s8 -w traffic_capture.pcap
   - **Purpose**: Saves all the captured traffic from the enp0s8 interface into a .pcap file, which can later be opened in Wireshark for more detailed analysis. This approach is particularly useful for reviewing traffic at a later time or sharing capture files.

Issues Encountered During Traffic Capture:

1. **Capturing Too Much Unrelated Traffic**:
   - Initially, the Tshark capture was flooded with a large amount of irrelevant network traffic, which made it difficult to isolate the desired SSH, mDNS, or SSDP packets.
2. **Large Capture File Sizes**:
   - When capturing a lot of traffic over an extended period, the resulting .pcap files became quite large, making them cumbersome to analyze.

Resolution Process:

1. **Refining Traffic Filters**:
   - To reduce the amount of unrelated traffic, refined the capture filters. For SSH traffic, limited the capture to **TCP port 22**, which greatly reduced the noise. Similarly, for service discovery traffic, added specific filters for **mDNS** and **SSDP**, ensuring only the relevant packets were captured (bash):

- tshark -i enp0s8 -f "tcp port 22" -w ssh_capture.pcap
- tshark -i enp0s8 -Y "mdns || ssdp"

2. **Managing File Sizes**:

   - To manage file sizes, captured traffic for shorter time periods or used Tshark's built-in options to limit the size of capture files. This ensured that the files were more manageable for analysis.

At this point, successfully captured relevant network traffic using Tshark, and the captured data was stored in .pcap files for later analysis.

---

4. Analyzing the Captured Traffic

The objective here was to analyze the network traffic that was captured using Tshark. By leveraging **Tshark** for command-line analysis, aimed to better understand the flow of network traffic, specifically focusing on SSH, mDNS, and SSDP traffic. The analysis helped identify and interpret the behavior of the network during different scenarios.

Tools Used for Analysis:

1. **Tshark**:

   - Used for quick command-line-based analysis of the captured traffic without needing to open the GUI.
   - Tshark is useful for extracting specific data fields or performing quick scans over large capture files.

Steps Taken for Analysis:

1. **Analyzing Packets in Tshark**:

   - Used Tshark for quick command-line analysis. The following commands were run to extract specific details from the captured traffic:
   - **Display Traffic in the Terminal (bash)**:
     - tshark -r traffic_capture.pcap
     - **Purpose**: Displays the captured packets in the terminal, providing a quick overview of the traffic.
   - **Extract Specific Fields (bash)**:
     - tshark -r traffic_capture.pcap -T fields -e ip.src -e ip.dst -e frame.time
     - **Purpose**: This command extracts specific fields such as source IP (ip.src), destination IP (ip.dst), and the timestamp of the packet (frame.time). It's

useful when looking for specific patterns or understanding traffic flow between devices.

Actual Experience During Analysis:

1. **SSH Traffic**:
    - Successfully captured and analyzed SSH traffic using **Tshark**, observed the TCP handshake process and encrypted payloads. Tshark provided detailed insights into the packet headers and flags, such as SYN and ACK, which helped in understanding the SSH connection lifecycle.
2. **mDNS and SSDP Service Discovery**:
    - Captured and inspected several mDNS and SSDP packets using **Tshark**. This traffic included device and service announcements, showing how devices on the network advertise their availability and services. Tshark’s field extraction feature provided a quick summary of the source and destination IPs, making it easy to analyze patterns in the multicast traffic.

Issues Encountered During Analysis:

1. **Large File Sizes Slowing Down Wireshark**:
    - When working with large .pcap files, Tshark’s performance slowed down, making it difficult to process and navigate through the traffic in real-time.

How These Issues Were Resolved:

1. **Splitting Capture Files**:
    - To address performance issues with large capture files, started capturing shorter traffic sessions or used **Tshark's** options to split the capture into smaller files. This ensured that **Tshark** could handle the analysis smoothly without performance degradation.

At this point, successfully analyzed the captured traffic using Tshark, providing insights into the behavior of SSH connections and service discovery protocols on the network.

---

5. Troubleshooting and Adjustments

The objective of this section is to document the issues encountered during the setup and use of Tshark, and the steps taken to resolve them. This ensures that any problems faced during installation, configuration, or traffic capture were addressed effectively to ensure smooth operation.

Issues Encountered During Installation and Usage:

1. **Non-root Users Unable to Capture Packets**:
    - Initially, the ubuntu user was unable to capture packets after the installation of Tshark. Packet capture requires special permissions, and this restriction led to permission denied errors.
2. **Network Interfaces Not Showing Up**:
    - The enp0s3 interface, expected to be available for traffic capture, was not listed when running tshark -D or trying to configure Wireshark.
3. **Capturing Too Much Unrelated Traffic**:
    - When capturing network traffic, Tshark initially collected too much data, including irrelevant traffic, which made isolating specific protocols (like SSH, mDNS, and SSDP) more difficult.
4. **Large Capture Files**:
    - When running traffic captures over extended periods, the .pcap files generated became very large, making them difficult to manage and slowing down analysis tools like Tshark.

How These Issues Were Resolved:

1. **Resolving Non-root User Permissions for Packet Capture**:
    - **Issue**: The ubuntu user could not capture packets because packet capturing requires elevated privileges.
    - **Solution**: Configured Tshark to allow non-root users to capture packets by adding the ubuntu user to the **wireshark group**: (bash):
        - sudo usermod -aG wireshark ubuntu
        - **Result**: After rebooting the system, the ubuntu user was able to capture packets without needing root privileges, resolving the issue.
2. **Resolving the Missing Network Interface Issue**:
    - **Issue**: The enp0s3 interface was not available for capturing traffic.
    - **Solution**: Investigated the VirtualBox network configuration and switched the VM from **NAT** to **Bridged Networking**. This change allowed the network interface enp0s8 to appear in both Wireshark and Tshark.
        - **Result**: After switching to bridged networking, confirmed that enp0s8 was available and ready for traffic capture, solving the issue with missing interfaces.
3. **Refining Traffic Filters**:

- **Issue**: Tshark was capturing a lot of unrelated traffic, making it difficult to focus on SSH, mDNS, or SSDP.
- **Solution**: Adjusted the filters used for capturing traffic, focusing on specific protocols and ports:
    - For SSH, filtered on **TCP port 22 (bash)**:
        1. tshark -i enp0s8 -f "tcp port 22" -w ssh_capture.pcap
    - For mDNS and SSDP, used protocol filters (bash):
    - tshark -i enp0s8 -Y "mdns || ssdp"
    - **Result**: These refined filters allowed for the capture only the relevant traffic, drastically reducing the noise and irrelevant data.

4. **Managing Large Capture Files**:
    - **Issue**: Long capture sessions produced large .pcap files, which slowed down Tshark during analysis.
    - **Solution**: To manage file sizes, used shorter capture durations and Tshark's built-in options to split capture files into smaller segments (bash):
        - tshark -i enp0s8 -a filesize:100000 -w traffic_capture.pcap
        - **Result**: This approach reduced file sizes, allowing for smoother analysis in Tshark and better overall management of captured data.

Summary of Adjustments:

- **User permissions**: Configured non-root access for packet capturing.
- **Networking configuration**: Switched to bridged networking to resolve missing interface issues.
- **Traffic filters**: Refined Tshark capture filters to focus on relevant traffic, minimizing unrelated data.
- **File size management**: Used smaller capture sessions and file splitting to prevent performance issues during analysis.

---

6. Conclusion

The successful installation, configuration, and use of **Tshark** in the home lab environment enabled efficient network traffic capture and analysis, even without a graphical interface. By addressing key challenges such as non-root user permissions and network interface availability, ensured that Tshark could be used effectively for monitoring protocols like SSH, mDNS, and SSDP.

Throughout this process, gained valuable insights into network traffic behavior, packet analysis, and troubleshooting common network issues. The flexibility of Tshark in handling command-line-based packet captures allowed for seamless integration into the server environment, making it a critical tool for our network monitoring and security operations. By resolving performance issues with large capture files and irrelevant traffic, was able to streamline the analysis process and improve overall efficiency.

This experience has enhanced my understanding of packet capture tools in a real-world scenario, providing a foundation for future advanced network analysis in the home lab environment and future SOC development.

Summary of Key Takeaways:

1. **Successful Installation and Configuration**:
    - Wireshark and Tshark were installed and configured successfully on the Ubuntu Server VM. Through proper permissions setup, non-root users were able to capture traffic, ensuring security while allowing flexible access to network analysis tools.
    - Initially, considered using Wireshark for network traffic analysis. However, since the Ubuntu Server VM lacked a graphical user interface (GUI), was unable to run Wireshark. This limitation led to **Tshark**, the command-line version of Wireshark, which enabled the capture and analysis of network packets directly from the terminal.
2. **Traffic Capture and Analysis**:
    - Used Tshark for capturing traffic, focusing on SSH, mDNS, and SSDP protocols. Capturing was done using filtered commands to limit traffic to only the relevant data, making the analysis more efficient. Wireshark's graphical interface was used for in-depth analysis, providing detailed views of captured packets, while Tshark was used for quick command-line insights.
3. **Issues Encountered and Resolved**:
    - Several challenges were encountered during the setup, such as permission issues, missing network interfaces, and handling large amounts of irrelevant traffic. These were resolved by refining network filters, adjusting VirtualBox networking settings, and ensuring proper permissions for the ubuntu user.
4. **Optimizing Traffic Capture**:
    - Through the use of refined capture filters and file size management, ensured that Tshark captured only the necessary traffic, which made subsequent analysis easier and faster. This optimization also helped avoid performance issues when analyzing large .pcap files.

Next Steps:

- **Automation and Advanced Analysis**:
    - For future improvements, automating traffic capture using cron jobs or scripts could enhance monitoring efficiency. Additionally, more advanced traffic analysis techniques, such as protocol-specific deep dives or anomaly detection, can be explored.
- **Integration with Other Tools**:
    - Tshark's command-line outputs could be integrated into larger workflows or used with other monitoring tools such as Graylog or Splunk to provide continuous network traffic insights in a more centralized manner.

Index

Home Lab – Network Scanning and Discovery with Nmap

Primary Objective

The goal was to install and configure Nmap as a primary network scanning tool within the home lab environment. Nmap was crucial for conducting network discovery, identifying open ports and services, and gathering information about potential vulnerabilities across the network. It served as a foundational tool for understanding the network's structure and security, making it a key component of the overall network security and monitoring process.

1. Installation Commands

The purpose of installing **Nmap** was to provide the ability to scan and discover devices, open ports, and services in the home lab environment. **Nmap** helped identify potential vulnerabilities and supported the overall network analysis process.

1. **Update the package list (bash)**:
    - sudo apt update
    - **Purpose**: This command ensures that the most current version of the package index is being used, which contains all the available packages from repositories. It is a best practice before installing new software on Ubuntu.
2. **Install Nmap (bash)**:
    - sudo apt install nmap
    - **Purpose**: This command downloads and installs the latest stable version of Nmap directly from the Ubuntu repository.
3. **Verify the installation (bash)**:
    - nmap --version
    - **Purpose**: This verifies that the installation completed successfully and returns the version of Nmap installed. In this case, the output was:
        - Nmap version 7.80 (https://nmap.org)
        - Platform: Linux

Each of these commands is vital for ensuring a successful and smooth installation process. Initially, started by updating the system to avoid potential dependency issues. The apt install command pulled the package from Ubuntu's main repository, and checking the version confirmed that Nmap was correctly installed.

2. Issues Encountered

- There were no issues during the Nmap installation itself. However, after installation, encountered a limitation while using NAT networking. Nmap couldn't perform effective scans of external networks due to the NAT restriction, which only allowed communication within the VM's internal network.

---

3. Resolution Process

- The NAT networking issue was resolved by switching to **Bridged Networking** mode, allowing Nmap to access both internal and external networks more effectively. Below are the steps taken to resolve the issue:
    - **Switching to Bridged Networking in VirtualBox**:
        1. **Navigate to**: Settings > Network for the specific VM.
        2. **Change**: Adapter 1 from **NAT** to **Bridged Adapter**.
        3. **Select the correct network interface**: In this case, selected the Wi-Fi interface (enp0s8) to bridge the VM to the external network.
        4. **Apply the changes and reboot the VM**.
        5. After this change, confirmed that Nmap was able to scan external networks and hosts outside the VM.

This resolution enabled full utilization of Nmap's network scanning capabilities across all the devices within the home lab setup.

---

2. Basic Usage Commands

After the installation, several different types of Nmap scans were run to explore its capabilities and collect detailed information about the network.

1. **Ping scan (host discovery)** (bash):
    - nmap -sP 192.168.1.0/24
        - **Purpose**: This scan was used to identify all active hosts on the 192.168.1.0/24 subnet. The output provided a list of IP addresses for all reachable devices, along with their MAC addresses and vendor information. The result confirmed the presence of hosts like the router, other virtual machines, and the host machine itself.
2. **SYN scan (stealth scan)** (bash):
    - sudo nmap -sS 192.168.1.100
        - **Purpose**: This is one of the most commonly used scans, known as a "stealth scan" because it does not complete the full TCP handshake. It was used to identify open ports on the Ubuntu VM (192.168.1.100). The output revealed several open ports, including:

- **Output**:
- PORT   STATE SERVICE
- 22/tcp   open   ssh
- 80/tcp   open   http
- 5432/tcp open   postgresql
- 9000/tcp open   cslistener
- 9200/tcp open   wap-wsp

3. **Aggressive scan (OS detection and service discovery)** (bash):

   - sudo nmap -A 192.168.1.100

     - **Purpose**: This advanced scan was used to gather detailed information about the operating system, services running, and version details. The output included the following OS detection result:

Output:

OS: Linux 2.6.32, Linux 5.0 - 6.2 (Ubuntu)

These scans were selected to provide a comprehensive overview of hosts, services, and potential vulnerabilities within the home lab network. The commands identified which services were exposed and confirmed that the VM was configured correctly.

---

3. Network Discovery

The primary goal of network discovery was to identify all active hosts and devices within the home lab's network. This is an essential first step before conducting more targeted vulnerability or service scans. It helps map out the network and pinpoint devices that are reachable and worth further investigation.

Commands for Network Discovery:

- **Ping Scan (Host Discovery)** (bash):

  - nmap -sP 192.168.1.0/24

  - **Purpose**: This scan sends ICMP Echo requests to each IP address within the specified subnet (192.168.1.0/24), discovering which hosts are active. It provides basic information like the IP addresses and MAC addresses of the active hosts.

Issues Encountered During Network Discovery:

- **Issue**: Initially, the Ping Scan (nmap -sP) only discovered internal network devices because it was operating in NAT networking mode.

- **Resolution**: Switching to Bridged Networking made it possible to scan the broader network, including the host machine and other devices within the subnet.

---

4. Advanced Network Discovery:

After switching to Bridged Networking, the network discovery was expanded using the following command to get additional details (such as open ports and services) about the discovered hosts.

- **SYN Scan (bash)**:
    - sudo nmap -sS 192.168.1.0/24
    - **Purpose**: This command performs a SYN scan across the entire subnet, identifying both the active hosts and their open ports, which would later help identify services and potential vulnerabilities on each host.
        - **Results**:
        - Nmap scan report for 192.168.1.1
        - Host is up (0.017s latency).
        - Not shown: 992 closed tcp ports (reset)
        - PORT   STATE   SERVICE
        - 22/tcp   filtered ssh
        - 53/tcp   open   domain
        - 80/tcp   open   http
        - 443/tcp open   https
        - 4567/tcp filtered tram
        - 8022/tcp filtered oa-system
        - 8080/tcp open   http-proxy
        - 8443/tcp open   https-alt
        - MAC Address: 18:78:D4:31:A6:73 (Verizon)

        - Nmap scan report for 192.168.1.100
        - Host is up (0.00011s latency).
        - Not shown: 995 closed tcp ports (reset)
        - PORT   STATE SERVICE
        - 22/tcp   open   ssh
        - 80/tcp   open   http
        - 5432/tcp open   postgresql
        - 9000/tcp open   cslistener
        - 9200/tcp open   wap-wsp
        - This scan not only discovered hosts but also identified the open ports and services running on each host, which provided valuable information for deeper analysis in later steps.

Why Network Discovery Matters: Identifying live hosts is essential in network reconnaissance, and Nmap's discovery options enabled efficient mapping of the lab's network. The transition to Bridged Networking was necessary to fully explore the network, and the follow-up SYN scan helped gather more detailed information about services running on each discovered host.

5. Advanced Usage Commands

1. **Service version detection (bash)**:
    - nmap -sV 192.168.1.100
    - **Purpose**: This command was used to detect the versions of services running on specific ports, such as SSH and HTTP. The output identified that the VM was running:
        - 22/tcp   open ssh   OpenSSH 9.6p1 Ubuntu 3ubuntu13.5 (Ubuntu Linux; protocol 2.0)
        - 80/tcp   open http   Apache httpd 2.4.58 ((Ubuntu))
        - 5432/tcp open postgresql PostgreSQL DB 16.0 - 16.2
        - 9000/tcp open cslistener?
        - 9200/tcp open http   Elasticsearch REST API 7.17.24 (name: node-1; cluster: elasticsearch; Lucene 8.11.3)
2. **UDP scan (bash)**:
    - sudo nmap -sU 192.168.1.100
    - **Purpose**: Used this command to perform a UDP scan to detect any open or filtered UDP ports on the same VM. The scan revealed the following service:
        - 514/udp open|filtered syslog
        - The "open|filtered" status means that Nmap couldn't determine definitively whether the port is open or filtered (blocked by a firewall), but it did detect some form of response.
3. **Nmap scripting engine (NSE) for vulnerability detection (bash)**:
    - nmap --script vuln 192.168.1.100
    - **Purpose**: This command runs a series of pre-defined scripts to check for vulnerabilities on the target host. The scan output highlighted no critical vulnerabilities but provided detailed information on services that could potentially be misconfigured.
    - **Open Ports and Vulnerability Scan Results:**
    - 22/tcp (SSH): No specific vulnerabilities found.
    - 80/tcp (HTTP):
    - CSRF vulnerabilities: None found.
    - DOM-based XSS vulnerabilities: None found.
    - Stored XSS vulnerabilities: None found.
    - Interesting folder: /server-status/ was identified as potentially interesting.

- 5432/tcp (PostgreSQL): No specific vulnerabilities reported.
- 9000/tcp (Graylog Web Interface): No specific vulnerabilities reported.
- 9200/tcp (Elasticsearch): No specific vulnerabilities reported.
- The scan found no major vulnerabilities, but it did flag the /server-status/ folder on the HTTP service, which could potentially expose server information if it's accessible.

Troubleshooting Summary

Issue 1: Network Access Limited to Internal Network (NAT)

- **Problem**: After successfully installing Nmap, initial scans were restricted to the internal network of the VM because **NAT networking** was configured. This prevented Nmap from discovering or scanning external hosts on the broader network.
- **Cause**: NAT (Network Address Translation) limits the VM's ability to interact with external network devices directly, allowing only internal traffic to be scanned.
- **Symptoms**: Running Nmap scans (e.g., nmap -sP or nmap -sS) only returned results for internal network devices (127.0.0.1 or local virtual network IP addresses) and failed to reach hosts outside the VM's subnet.

Resolution Steps:

1. **Switched from NAT to Bridged Networking**:
   - In **VirtualBox**, navigated to **Settings > Network** for the VM.
   - Changed **Adapter 1** from **NAT** to **Bridged Adapter**.
   - Selected the correct physical network interface (Wi-Fi interface wlp3s0 in this case).

2. **Rebooted the VM**:
   - After applying the change, rebooted the VM to ensure the new network settings took effect.
3. **Tested Nmap Scans Again**:
   - After rebooting, Nmap was able to scan external hosts on the home network, confirming the resolution.

Outcome: The switch to **Bridged Networking** allowed Nmap to access and scan all hosts on the 192.168.1.100 subnet and beyond, eliminating the network restrictions.

---

10. Conclusion

Nmap was successfully installed and configured as a core tool in the home lab environment. It was used to perform a variety of network scans, ranging from basic host discovery to more advanced service detection, OS fingerprinting, and vulnerability assessments. The scans provided valuable insights into the network's architecture, services running on hosts, and potential security weaknesses. Nmap's versatility and ease of use made it an essential part of the network reconnaissance phase.

Key Takeaways:

1. **Switching to Bridged Networking**: This was a critical step in enabling effective scanning beyond the VM's internal network. Without this change, Nmap would have been limited to scanning internal hosts.
2. **Comprehensive Network Scanning**: Nmap's ability to perform different types of scans—such as SYN scans, UDP scans, and aggressive service detection—allowed for a deep understanding of the home lab network's services and potential vulnerabilities.
3. **Real-World Relevance**: The project demonstrated Nmap's importance in real-world scenarios for network discovery and vulnerability assessment, proving its value as a foundational tool in network security.

In summary, Nmap played an integral role in the network scanning process, and the successful configuration and scanning of external hosts laid the groundwork for future security analysis within the home lab.

Home Lab – Metasploit Vulnerability Assessment and Exploitation Framework Setup

---

Primary Objective

The goal was to install and configure Metasploit as a key penetration testing tool within the home lab environment. Metasploit was critical for assessing vulnerabilities across the network and testing security defenses. By simulating real-world attacks, it provided insight into potential weaknesses in systems and allowed for the testing of various exploits. This setup laid the foundation for more advanced security analysis, complementing other tools like Nmap for network discovery and Tshark for traffic monitoring, making it an integral part of the overall security testing and analysis process.

---

1. **Installation Steps:**

   1. **Install Metasploit using Snap (bash)**:

      - sudo snap install metasploit-framework
      - **Explanation**: This command installs Metasploit from the Snap store. Snap allows for easier management of the tool, including automatic updates.

   2. **Verify the installation (bash)**:

      - msfconsole --version
      - **Explanation**: This command checks if Metasploit was installed correctly and verifies the version.

Issues Encountered:

1. **Snap Installation Delays**: The Snap installation took longer than expected, and at one point, it seemed to hang.
2. **Permissions Errors**: After installation, running msfconsole initially resulted in permission errors due to insufficient user privileges.

Resolution Process:

1. **Snap Installation Delays**:
   - **Resolution**: Restarted the Snap service to resolve the delay (bash):
     - sudo systemctl restart snapd
   - Verified Snap status to ensure it was functioning correctly (bash):

        - sudo systemctl status snapd

2. **Permissions Errors**:
    - **Resolution**: Added the ubuntu user to the necessary groups to resolve permission issues (bash):
        - sudo usermod -aG sudo ubuntu

---

2. **Network Configuration**

The objective was to configure the network settings in VirtualBox to ensure that Metasploit could properly interact with the target machine, which required switching from NAT networking to Bridged Networking.

1. **Switch from NAT to Bridged Networking**:
    - **Explanation**: In VirtualBox, the network adapter settings were changed from NAT (which isolates the VM) to Bridged Networking (which allows the VM to communicate directly with devices on the local network). This was necessary for Metasploit to scan and interact with the Ubuntu VM.
2. **Verify the network interface (bash)**:
    - ip a
    - **Explanation**: This command was used to verify that the VM had the correct network interface and IP address after switching to Bridged Networking.

Initial Network Isolation: Using NAT networking initially caused issues with reaching the target VM for scanning and exploitation, as the NAT configuration blocked network traffic to the VM.

Resolution: Switched to Bridged Networking in VirtualBox to allow proper communication between Metasploit and the target machine. This allowed the Metasploit scans and exploits to work successfully.

---

3. Basic Test Run

The goal was to verify the functionality of Metasploit by attempting to exploit a known vulnerability, the **vsftpd 2.3.4 backdoor**, on the target Ubuntu Server VM. Additionally, tested if any FTP services were running on the target system to confirm its vulnerability status.

1. **Start the Metasploit Console (bash)**:
    - msfconsole

- **Purpose**: This command starts the **Metasploit Framework Console**, which is the command-line interface for interacting with Metasploit's various modules, including exploits, payloads, and auxiliary functions.

2. **Select the vsftpd 2.3.4 Backdoor Exploit (msfconsole)**:
    - use exploit/unix/ftp/vsftpd_234_backdoor
    - **Purpose**: The use command loads the specific Metasploit exploit module for the **vsftpd 2.3.4 backdoor** vulnerability. This exploit allows attackers to gain a shell if the vulnerable service is running on the target machine.
    - **vsftpd_234_backdoor**: This exploit targets the vsftpd version 2.3.4, which contained a malicious backdoor that could allow unauthorized access.
3. **Set the Target Machine's IP Address (msfconsole)**:
    - set RHOSTS 192.168.1.100
    - **Purpose**: The set command assigns values to specific parameters required by the loaded exploit. In this case:
        - RHOSTS stands for "Remote Hosts" and is used to define the target machine's IP address, here 192.168.1.100, which is the Ubuntu VM.
4. **Run the Exploit (**msfconsole):
    - run
    - **Purpose**: This command executes the exploit, attempting to take advantage of the **vsftpd 2.3.4** backdoor vulnerability by connecting to the target system on the FTP service (port 21) and creating a malicious connection.

Outcome: The exploit failed as the target machine did not have the vulnerable FTP service running or reachable.

---

4. Verify FTP Service on Target Using Nmap:

After the exploit failed, verified whether the target machine had an FTP service running (bash).

- sudo nmap -sV -p 21 192.168.1.100

- **Purpose**: This command uses **Nmap** to check if port 21 (FTP) is open and what version of the service (if any) is running on the target machine.
    - sudo: This ensures elevated privileges are used for running Nmap, which is required for some network operations like version detection.
    - nmap: The Nmap tool is used to scan networks and discover open ports, services, and versions.
    - -sV: This option is for **service version detection**, which attempts to determine the version of the services running on the specified port.

- -p 21: This option restricts the scan to **port 21**, which is commonly used for FTP services.
- 192.168.1.100: This is the IP address of the target Ubuntu VM.

Outcome: The scan results showed that **port 21 (FTP)** was **closed**, meaning there was no FTP service running on the target system. As a result, the system was not vulnerable to any FTP-related exploits, including **vsftpd 2.3.4**.

---

5. Troubleshooting and Further Configuration

The goal was to ensure that Metasploit functioned without errors by resolving any issues related to Snap installation and permissions.

1. **Check Snap service status (bash)**:
   - sudo systemctl status snapd
   - **Explanation**: Checked the status of the Snap service to ensure it was running.
2. **Restart Snap service (bash)**:
   - sudo systemctl restart snapd
   - **Explanation**: Restarted the Snap service to resolve delays during the installation.
3. **Verify user permissions (bash)**:
   - sudo usermod -aG sudo ubuntu
   - **Explanation**: Added the ubuntu user to the sudo group to resolve permission issues when running Metasploit.
4. **Run Metasploit console (bash)**:
   - msfconsole
   - **Explanation**: Verified that the console ran without errors after resolving permission issues.

Issues Encountered

1. **Snap Installation Delays**: The Snap installation hung temporarily, but restarting the service resolved the issue.
2. **Permissions Errors**: Insufficient permissions initially prevented the msfconsole command from running, but this was fixed by adding the user to the sudo group.

Resolution Process

1. **Snap Installation**: Restarted the Snap service using systemctl commands.

- sudo systemctl restart snapd
- The systemctl command is used to control system services. The restart option restarts the snapd service, which is responsible for managing Snap packages on Ubuntu. Restarting the service can resolve any temporary issues or delays that occurred during the installation.

2. **Permissions**: Updated user group permissions to allow Metasploit to run properly.
   - **Explanation:**
     The usermod command modifies a user's account settings. The -aG option appends the user to a group, in this case, the sudo group, which grants administrative privileges.
     - -aG stands for "append to group"—meaning that the user will be added to the specified group without removing them from any other groups.
     - $USER is a system environment variable that automatically represents the currently logged-in user's name.

---

6. Conclusion

The installation and configuration of Metasploit in the home lab provided hands-on exposure to essential penetration testing tools, highlighting both technical challenges and successful solutions. The project encountered minor issues, including Snap installation delays and permissions errors, which were resolved through service restarts and user group adjustments. These efforts allowed for the successful setup of Metasploit and the execution of various network scans.

While the attempt to exploit the **vsftpd 2.3.4 backdoor vulnerability** was unsuccessful due to the FTP service being unavailable on the target machine, this underscored the importance of correctly identifying services running on the system before launching any attack. The verification using **Nmap** to check for an open FTP port further confirmed the system's secure state with regard to this vulnerability.

The process also emphasized the importance of **proper network configuration**, as switching from NAT to Bridged Networking was crucial for effective communication between Metasploit and the target machine. Overall, this project offered a deeper understanding of both **Metasploit's capabilities** and the system and network settings that support its operation within a home lab environment.

Key Achievements:

- Successfully installed Metasploit via Snap, despite initial delays.
- Resolved issues related to Snap installation and user permissions, ensuring proper operation.

- Conducted a **basic vulnerability assessment**, including an **attempted exploit** of the **vsftpd 2.3.4 backdoor** on the Ubuntu Server VM.
- Verified the target's vulnerability status using **Nmap**, which confirmed that the FTP service was not running.
- Configured VirtualBox from NAT to Bridged Networking, allowing Metasploit to communicate with the target VM directly.

Lessons Learned:

- **Snap Package Management**: While Snap provides a convenient method for managing software like Metasploit, it can occasionally cause delays or hangs. Learning how to manage Snap services (e.g., restarting **snapd**) is essential for smooth installation and operation.
- **Permissions Management**: Proper user permissions are crucial when running penetration testing tools. Ensuring that the correct group memberships are assigned, such as adding the user to the **sudo** group, prevents unnecessary roadblocks.
- **Network Configuration**: Correct network settings are critical for penetration testing. Switching from **NAT** to **Bridged Networking** enabled Metasploit to interact properly with the target machine, emphasizing the importance of understanding and configuring network settings in a virtualized environment.

Next Steps:

- Future testing will involve more advanced vulnerability scans, testing custom payloads, and potential integration with other SOC tools such as **Graylog** for logging and reporting.
- Additional exploits and services will be tested on the Ubuntu Server VM to further explore Metasploit's capabilities in identifying potential vulnerabilities.

Home Lab – Graylog: Centralized Log Management Setup

Primary Objective

The goal was to install and configure Graylog as a centralized log management solution within the home lab environment. Graylog played a critical role in aggregating, analyzing, and managing logs from various sources, including the Ubuntu Server VM. By providing a structured way to collect and search log data, it enhanced the ability to monitor system and network activities in real-time. The Graylog setup enabled advanced log analysis, detection of anomalies, and correlation of security events. It worked in conjunction with other tools like Splunk for Security Information and Event Management (SIEM) and Tshark for network traffic analysis, making Graylog a foundational component in the home lab's security monitoring and log management infrastructure.

Section 1: Installation of Graylog

To install and configure Graylog as the primary log management and analysis tool for the home lab. Graylog enables the collection, indexing, and real-time analysis of log data from multiple sources, which is crucial for monitoring and troubleshooting network activities and system performance.

Commands Used During Installation

1. **Update system and install required dependencies (bash):**
   - sudo apt update && sudo apt upgrade -y
   - sudo apt install openjdk-11-jre-headless uuid-runtime pwgen

   - This updates the package list and upgrades installed packages. The openjdk-11-jre-headless package is needed because Graylog runs on Java. uuid-runtime and pwgen are required for generating unique secrets during Graylog configuration.
2. **Install MongoDB (required for Graylog to store data) (bash):**
   - wget -qO - https://www.mongodb.org/static/pgp/server-4.2.asc | sudo apt-key add -
   - echo "deb [arch=amd64] https://repo.mongodb.org/apt/ubuntu bionic/mongodb-org/4.2 multiverse" | sudo tee /etc/apt/sources.list.d/mongodb-org-4.2.list
   - sudo apt update
   - sudo apt install -y mongodb-org
   - sudo systemctl enable --now mongod

   - This installs MongoDB, which is necessary for Graylog to store its configuration and metadata. The command ensures MongoDB is started and enabled to run at boot.
3. **Install Elasticsearch (used for indexing logs) (bash):**
   - wget -qO - https://artifacts.elastic.co/GPG-KEY-elasticsearch | sudo apt-key add -

- echo "deb https://artifacts.elastic.co/packages/7.x/apt stable main" | sudo tee /etc/apt/sources.list.d/elastic-7.x.list
- sudo apt update
- sudo apt install elasticsearch
- sudo systemctl enable --now elasticsearch

- Elasticsearch is used by Graylog to index and store logs. This command installs Elasticsearch, adds its repository, and ensures it starts automatically.

4. **Install Graylog (the log management tool) (bash):**

   - wget https://packages.graylog2.org/repo/packages/graylog-4.2-repository_latest.deb
   - sudo dpkg -i graylog-4.2-repository_latest.deb
   - sudo apt update
   - sudo apt install graylog-server

   - This installs Graylog by downloading the repository file, updating the package list, and installing the server.

5. **Configure Graylog (bash):**

   - sudo nano /etc/graylog/server/server.conf

   - This command opens the configuration file where the password_secret and root_password_sha2 parameters need to be set to secure Graylog.

6. **Start the Graylog server (bash):**

   - sudo systemctl enable --now graylog-server

   - This command starts the Graylog server and ensures it will start automatically on reboot.

Issues Encountered During Installation

1. **Networking Issue:** Initially, NAT networking was configured for the VM, which blocked access to the Graylog web interface.
2. **Disk Space Issue:** Later, disk space became a problem due to log retention settings, which led to the Graylog web interface becoming inaccessible.

Resolutions to Installation Issues

1. **Networking Issue:** Resolved the access problem by switching from NAT to bridged networking. This allowed access to the Graylog web interface via the host machine's IP address.
2. **Disk Space Issue:** Expanded the root partition to 48GB and configured journald to manage log storage more efficiently, ensuring the disk space issue would not reoccur.

Section 2: Configuration of Graylog

The objective was to properly configure Graylog to ensure it works in conjunction with its dependencies (Elasticsearch, MongoDB), and to adjust network settings, log inputs, and retention policies to manage system and network logs efficiently. Configuration was also necessary to secure the system with a strong password and integrate log collection from different sources.

Commands Used During Configuration

1. **Set up the Graylog configuration file (bash):**
    - sudo nano /etc/graylog/server/server.conf
    - This command opens the Graylog server configuration file for editing. The file contains settings for password security, server ports, and log management.
2. **Generate a secret for securing the server (bash):**
    - pwgen -N 1 -s 96
    - This generates a 96-character secret used for the password_secret parameter in the Graylog configuration file, which is necessary for encryption.
3. **Generate a SHA2 hash for the admin password:**
    - echo -n "yourpassword" | sha256sum
    - This generates a SHA256 hash of the password, which is then added to the root_password_sha2 field in the Graylog configuration file to secure the Graylog web interface.
4. **Change log retention policies in Graylog:**
    - This was configured within the Graylog web interface (Settings > System > Indices) to manage how long logs are retained to avoid disk space issues. This ensures that old logs are automatically deleted after a specified period, keeping the system healthy and within storage limits.

Issues Encountered During Configuration

1. **Networking Issue:** Initially was unable to access the Graylog web interface due to incorrect networking settings.
2. **Log Input Confusion:** There was some confusion around configuring log inputs for Graylog, particularly with SSH and system logs.

Resolutions

1. **Networking Issue:**

- Resolved this by switching from NAT networking to bridged networking. This allowed the Graylog web interface to be accessible via the correct IP address.

2. **Log Input Confusion:**

    - The issue with configuring log inputs was resolved by correctly setting up the Syslog UDP input in Graylog (via the web interface) to collect logs from the system. The inputs were tested by sending logs manually from the command line using (bash):

        - logger "Test log message"

        - This sent a test log message to Graylog, confirming that logs were being successfully received and indexed.

Section 3: Log Management and Retention in Graylog

The objective was to configure log retention policies to prevent the system from running out of disk space and to ensure logs are efficiently managed. By setting up proper retention, older logs could automatically be removed while maintaining a healthy storage system for Graylog to function without interruptions.

Commands Used During Log Management and Retention Configuration

1. **Check disk space usage (bash):**

    - df -h

    - This command checks disk space usage and showed that the system was running low on space due to growing log files.

2. **Expand the root partition to 48GB:**

    - Expanded the root partition using VirtualBox’s disk resizing feature, followed by (bash):

        - sudo growpart /dev/sda 1
        - sudo resize2fs /dev/sda1

        - These commands resized the file system to use the newly available space, preventing further issues with Graylog due to lack of storage.

3. **Configure journald for better log management (bash):**

    - sudo nano /etc/systemd/journald.conf

    - This opens the journald configuration file, where the settings for log rotation and retention were adjusted. Configured it to limit the size of the journal logs by editing (journald):

        - SystemMaxUse=1G
        - SystemKeepFree=2G

- These parameters ensure that the journald log system doesn't consume too much disk space by limiting it to 1GB and always keeping 2GB of free space available on the system.

Issues Encountered During Log Management and Retention

1. **Disk Space Issue:**
   The Graylog web interface became inaccessible due to the server running out of disk space. This was caused by logs growing uncontrollably without proper retention policies in place.

Resolutions

1. **Disk Space Issue:**
   - Resolved this by expanding the root partition from its original size to 48GB and resizing the file system to use the additional space.
2. **Journald Log Management:**
   - To prevent the disk space issue from reoccurring, configured journald to limit the size of logs stored. This ensured that older logs were rotated out and only a limited amount of space was used by the logging system.
   - Additionally, configured Graylog's built-in retention settings through the web interface, specifying that logs older than a certain number of days would be deleted automatically.

Section 4: Automation of Graylog Startup and Log Backup

The objective was to automate the startup of the Graylog service and ensure that log backups are conducted regularly without manual intervention. Automating these tasks improves system reliability and ensures logs are securely stored for future analysis, even in case of unexpected issues or reboots.

Commands Used for Automation

1. **Enable Graylog to start on system boot (bash):**
   - sudo systemctl enable graylog-server
   - This command ensures that the Graylog service starts automatically every time the system boots, removing the need to manually start the service after a reboot.
2. **Configure cron job for daily log backups (bash):**
   - sudo crontab -e

- This opens the cron configuration file to schedule automated tasks. Added the following line to back up logs daily at 2 AM (bash):
- 0 2 * * * tar -czf /mnt/g-drive/log-backups/graylog-logs-$(date +\%F).tar.gz /var/log/graylog/
- This command compresses the Graylog logs and stores them on the G drive every day. The $(date +\%F) ensures each backup file is named with the current date for easy identification.

3. **Check cron logs (bash):**
   - grep CRON /var/log/syslog
   - This command checks if the cron job ran successfully by searching the system log for entries related to cron.

Issues Encountered During Automation

1. **Permissions Issue:**
   - Initially, there was a permissions issue when trying to store the backups on the G drive, which prevented the cron job from running successfully.

Resolutions

1. **Permissions Issue:**
   - The issue was resolved by adjusting the ownership and permissions on the G drive to ensure the backup process had the necessary rights to write to the directory. Used the following commands (bash):
   - sudo chown -R ubuntu:ubuntu /mnt/g-drive/
   - sudo chmod -R 755 /mnt/g-drive/
   - This set the appropriate permissions, allowing the backup process to write to the G drive without further issues.

Section 5: Troubleshooting Graylog

The objective of troubleshooting was to identify and resolve various issues that occurred during the setup, configuration, and operation of Graylog. This was essential to ensure continuous log monitoring and system functionality without interruptions. Key areas of troubleshooting included network access, disk space management, and service availability.

Commands Used During Troubleshooting

1. **Check Graylog service status (bash):**
   - sudo systemctl status graylog-server
   - This command shows the current status of the Graylog server, helping to identify if it was active or if there were issues with starting the service.
2. **Check disk space usage (bash):**
   - df -h
   - This command checks how much disk space is being used, which helped diagnose the issue when the system ran out of space, causing the Graylog web interface to become inaccessible.
3. **View Graylog logs (bash):**
   - sudo journalctl -u graylog-server
   - This command checks Graylog-specific logs to identify issues, such as startup errors or other service-related problems.
4. **Restart Graylog service (bash):**
   - sudo systemctl restart graylog-server
   - Used when Graylog failed to start properly, this command restarts the Graylog server after making necessary configuration changes or fixes.
5. **Check open ports and network configuration (bash):**
   - sudo netstat -tuln | grep 9000
   - This command checks if the Graylog web interface port (9000) is open and listening, which helped verify if Graylog was running and accessible via the web interface.
6. **Test network connectivity (bash):**
   - ping 192.168.1.100
   - This command was used to test connectivity between the host machine and the Graylog VM to ensure networking was properly set up, especially after switching to bridged networking.

Issues Encountered During Troubleshooting

1. **Web Interface Inaccessibility:**
   - Initially, the Graylog web interface was inaccessible due to NAT networking settings and later due to disk space issues.
2. **Disk Space Issue:**

- The Graylog server stopped functioning because the system ran out of disk space, which was being consumed by logs.

3. **Backup Permissions Issue:**

   - When setting up automated log backups, there was a permissions issue that prevented logs from being saved to the G drive.

Resolutions

1. **Web Interface Inaccessibility (Networking Issue):**

   - Switched from NAT networking to bridged networking to ensure the Graylog web interface was accessible through the host machine's IP. This allowed external access to the interface and resolved the initial inaccessibility.

2. **Disk Space Issue:**

   - After identifying that the root partition was full, expanded the root partition to 48GB and resized the filesystem using (bash):

   - sudo growpart /dev/sda 1
   - sudo resize2fs /dev/sda1

   - Also configured log retention policies in both journald and Graylog to prevent excessive log accumulation.

3. **Backup Permissions Issue:**

   - The permissions issue was resolved by modifying the ownership and permissions of the G drive using (bash):

   - sudo chown -R ubuntu:ubuntu /mnt/g-drive/
   - sudo chmod -R 755 /mnt/g-drive/

   - This ensured that the cron job could successfully save log backups to the G drive.

Conclusion

This project successfully established Graylog as the central log management solution within the home lab, integrating key components like Elasticsearch and MongoDB to create a robust system for real-time log collection and analysis. The setup process, while not without its challenges, provided valuable insights into optimizing system performance and ensuring long-term reliability.

Challenges and Solutions

Throughout the project, several obstacles were encountered:

- **Network Access Issues:** Accessing the Graylog web interface initially posed problems due to the use of NAT networking. Switching to bridged networking resolved this issue, enabling seamless access and management through the host machine's IP address.
- **Disk Space Constraints:** As log data grew, the home lab faced disk space limitations. Expanding the root partition and configuring effective log retention policies, ensured that disk usage remained manageable, preventing future space-related issues.
- **Automation and Permissions:** Automating the startup and backup of Graylog required resolving permission issues with the G drive. Adjusting file ownership and permissions allowed backups to execute smoothly, ensuring that logs were regularly archived without manual intervention.

Outcomes and Future Potential

The final configuration of Graylog now enables continuous monitoring and log analysis in the home lab environment, with automated log retention and backup processes providing added resilience. The system is designed to prevent overload through efficient log rotation, and the automation of key processes ensures minimal manual oversight is required, improving overall system reliability.

This project also sets the stage for more advanced monitoring capabilities in the future. As the home lab evolves toward becoming a full-fledged Security Operations Center (SOC), the centralized log management provided by Graylog will serve as a critical foundation for detecting and responding to potential security threats.

Overcoming the challenges encountered in this project has established a scalable, automated, and efficient logging infrastructure. This lays a strong groundwork for expanding the home lab into a comprehensive cybersecurity training and testing environment, enabling more advanced security operations, alerting, and incident response in the future.

Home Lab – Splunk: Security Information and Event Management (SIEM) Setup

Primary Objective

The goal was to install and configure Splunk as a central Security Information and Event Management (SIEM) tool within the home lab environment. Splunk was essential for aggregating, analyzing, and visualizing log data from various sources, providing real-time insights into system and network activities. By collecting logs from different devices, such as the Ubuntu Server VM, it enabled detailed event correlation and identification of potential security threats. This setup laid the foundation for a comprehensive logging and monitoring framework, complementing other tools like Graylog for centralized log management and Tshark for traffic analysis, making Splunk a crucial component in overall security operations and event detection within the home lab.

---

1. System Preparation and Dependency Installation

To prepare the Ubuntu Server VM by ensuring that necessary dependencies are installed for downloading and running Splunk. This includes verifying that the system supports secure communications and file transfers.

Commands Run:

1. **Install OpenSSL (bash):**
   - sudo apt-get install openssl
   - **Explanation**: OpenSSL is required to provide cryptographic functionality, enabling SSL/TLS support, which is important for secure communication channels, especially when using services like Splunk that require secure data transfer.
2. **Install wget and curl (bash):**
   - sudo apt-get install wget curl
   - **Explanation**: wget and curl are command-line utilities used to download files from the internet. These tools were needed to download the Splunk package from the official website.

No issues were encountered during the installation of these dependencies. The commands executed successfully, and the necessary tools were installed.

---

2. Downloading Splunk

Attempted to download Splunk version 9.3.0 from the official Splunk website onto the Ubuntu VM using the wget command, but encountered issues. As a result, the package was manually downloaded from the website and transferred to the VM for installation.

Commands Attempted:

1. Download the Splunk package (bash):
   - wget -O splunk-latest-linux-x86_64.deb "https://www.splunk.com/page/download_track?file=9.3.0/linux/splunk-9.3.0-51ccf43db5bd-linux-2.6-amd64.deb&ac=get"
   - Explanation: This command was supposed to download the latest version of Splunk for Linux (64-bit architecture) using wget. The -O option renames the downloaded file to splunk-latest-linux-x86_64.deb for easier reference.

Issues Encountered: The wget command failed to download the Splunk package. This required downloading the package directly from the Splunk website via a browser and then manually transferring it to the Ubuntu VM.

- **Error Message:**
- ERROR 403: Forbidden

Issue Resolution: Since the wget command failed to download the Splunk package, it was manually downloaded from the official Splunk website and transferred to the VM using a shared folder in VirtualBox.

1. **Downloaded the Splunk .deb package via a browser**:
   - Downloaded from Splunk Downloads.
2. **Transfer the package to the Ubuntu VM** using a shared folder:
   - In VirtualBox, **Settings** > **Shared Folders**, and added a folder on the host machine named **splunk_share**.
   - Mounted the shared folder in the VM using (bash):
     - sudo mount -t vboxsf splunk_share /mnt
   - Copied the Splunk package from /mnt to the Ubuntu VM (bash):
     - cp /mnt/splunk-latest-linux-x86_64.deb /home/ubuntu/

---

3. Installing Splunk

Objective: To install Splunk on the Ubuntu Server VM by unpacking and setting up the downloaded .deb package.

Commands Run:

1. **Install Splunk using dpkg (bash):**
   - sudo dpkg -i splunk-latest-linux-x86_64.deb

- **Explanation**: This command installs the .deb package that we downloaded in the previous step. The dpkg tool unpacks the files and installs the Splunk application on the system.

2. **Verify installation (bash):**

- sudo systemctl status splunk
- **Explanation**: This command checks the status of the Splunk service. This was to ensure that Splunk was properly installed and recognized by the system's service manager.

Issues Encountered: During installation, a dependency issue occurred, preventing the package from being fully installed. The error indicated that additional dependencies were required.

Issue Resolution: Resolved the issue by running the following command to install missing dependencies (bash):

- sudo apt-get install -f
- This command forces the system to fetch and install any missing dependencies required by the Splunk package. After running it, the installation was successfully completed.

---

4. Initial Splunk Setup

To initialize Splunk for the first time, accept the licensing agreement, and configure the admin credentials for accessing the web interface.

Commands Run:

1. **Start Splunk and accept the license (bash):**

- sudo /opt/splunk/bin/splunk start --accept-license
- **Explanation**: This command starts Splunk for the first time and prompts the user to accept the license agreement. We used the --accept-license flag to bypass the manual license acceptance prompt.

2. **Set up the admin credentials:** During the initial start-up, was prompted to set the admin username and password:

- **Username:** admin
- **Password:** [Set your password here]

Issues Encountered: No issues were encountered during this step. The license was accepted, and the admin credentials were successfully set up.

Issue Resolution: No troubleshooting was required at this stage.

---

5. Configuring Splunk to Start on Boot

To configure Splunk so that it starts automatically whenever the Ubuntu Server VM is rebooted, ensuring it runs as a service without manual intervention.

Commands Run:

1. **Enable Splunk to start on boot (bash):**
    - sudo /opt/splunk/bin/splunk enable boot-start
    - **Explanation**: This command configures Splunk to automatically start at system boot by installing an init script that will manage the service. This ensures that Splunk is running as soon as the VM is powered on.

Issues Encountered: No issues were encountered during this step.

Issue Resolution: No troubleshooting steps were required for this section.

---

6. Firewall and Web Access

To configure the firewall to allow access to the Splunk web interface and ensure that the Splunk service is reachable from a web browser on the host machine. The web interface provides a platform for managing and interacting with Splunk's features and data.

Commands Run:

1. **Open the firewall on port 8000 (Splunk Web Interface):** On Windows, opened the necessary firewall ports through Windows Defender Firewall settings.
    - **Explanation**: Splunk's web interface runs on port 8000 by default. It was necessary to ensure that Windows Defender allowed inbound traffic on this port to access the web interface.
2. **Access the Splunk Web Interface:**
    - After confirming that the firewall was properly configured, accessed the Splunk web interface by opening a browser on the host machine and navigating to (web):
        - http://192.168.1.100:8000
        - **Explanation**: The IP address 192.168.1.100 corresponds to the Ubuntu VM's bridged adapter IP address. Port 8000 is the default for the Splunk web interface, making it possible to log into the Splunk dashboard from any machine on the same network.

Issues Encountered: Initially encountered an issue where the Splunk web interface was not accessible. This was due to the firewall blocking traffic on port 8000.

Issue Resolution: Resolved this issue by checking the firewall settings on Windows Defender and manually adding a rule to allow inbound traffic on port 8000. After this, the web interface was accessible, and logging in using the admin credentials was successful.

7.. Troubleshooting Summary

Objective: To address any issues encountered during the installation and configuration of Splunk, particularly related to web access and network configuration.

Issues Encountered:

1. **Web Interface Not Accessible**:
    - **Problem**: Initially, the Splunk web interface was not accessible through the browser on the host machine.
    - **Cause**: The issue was traced to Windows Defender firewall blocking traffic on port 8000, which is required for accessing the Splunk web interface.
2. **Networking Issue with NAT**:
    - **Problem**: The initial setup used NAT networking in VirtualBox, which prevented direct access to the VM from the host machine.
    - **Cause**: NAT did not provide an IP address that could be directly accessed from the host network.

Issue Resolution:

1. **Firewall Configuration**:
    - **Solution**: Resolved the firewall issue by manually adding a firewall rule in Windows Defender to allow inbound traffic on port 8000. This allowed access to the Splunk web interface through the browser on the host machine.
2. **Switching to Bridged Adapter**:
    - **Solution**: Changed the network adapter in VirtualBox from NAT to Bridged Adapter. This allowed the VM to obtain an IP address from the local network, making it accessible from the host machine and resolving the network access issue.

Steps Taken:

- In VirtualBox, powered down the VM and changed the network settings to use the **Bridged Adapter** option.
- After rebooting, verified the new IP address by running (bash):
    - ip addr show
- The new IP address, 192.168.1.100, was used to access the Splunk web interface.

Conclusion

The installation and configuration of Splunk on the Ubuntu Server VM was successfully completed, integrating it as a key component of the home lab environment. After resolving network access and firewall configuration issues, Splunk was set to automatically start on boot and configured to monitor critical system logs (syslog). The web interface is fully operational, enabling real-time log monitoring and data analysis.

While this setup provides a solid foundation, it represents only a basic configuration of Splunk. Future steps could include setting up more advanced features such as custom dashboards, data ingestion from multiple sources, alerting for specific security events, and integrating Splunk with other tools like Graylog for enhanced log management. Additionally, configuring Splunk's indexing and searching capabilities for more sophisticated data analysis would further expand its functionality in a Security Operations Center (SOC) environment.

This process highlights the practical skills required to deploy and manage essential cybersecurity tools in a virtualized environment. The troubleshooting steps, particularly around networking and firewall adjustments, emphasize the problem-solving abilities needed to ensure smooth operation in complex systems. Overall, the successful implementation of Splunk enhances the lab's monitoring capabilities and strengthens its security infrastructure.

Home Lab - PostgreSQL Setup and Configuration

Primary Objective:

To install, configure, and optimize a PostgreSQL database environment as part of a Home Lab environment. This included implementing key database management practices such as user role creation, SSL encryption for secure connections, log monitoring, automated backups, and performance optimization using VACUUM. Additionally, the integration of PostgreSQL with Graylog for centralized log monitoring, ensuring that all database activities are tracked and available for analysis.

Key Accomplishments:

1. **Installation and Service Management**:
   - Successfully installed PostgreSQL and configured it to start automatically on system boot, ensuring that the database is always available.
   - Verified the installation and set up user roles for secure database access.
2. **Database and User Management**:
   - Created the homelab_db database and user roles with different access levels (read-only and read-write) to demonstrate secure and role-based access control.
   - Implemented indexing and optimization techniques to improve query performance.
3. **Backup and Data Protection**:
   - Set up manual and automated backup routines using pg_dump and cron jobs, ensuring the database is protected from data loss.
   - Tested and verified backups to ensure data could be restored when needed.
4. **Performance Optimization**:
   - Used VACUUM and VACUUM ANALYZE to optimize the database by removing dead tuples and updating query planner statistics for faster query execution.
5. **Log Monitoring and Graylog Integration**:
   - Enabled detailed logging in PostgreSQL and set up Filebeat to forward logs to Graylog. This allowed for real-time monitoring of database activity and a better understanding of how the database performs over time.
6. **SSL Configuration for Secure Connections**:
   - Enabled SSL in PostgreSQL and generated the necessary self-signed certificates and private keys to encrypt connections. This ensures data in transit between the

PostgreSQL server and clients is encrypted, providing better security for sensitive information.

Step 1: Installation of PostgreSQL

The objective was to install and configure PostgreSQL as the primary database system for the home lab setup. This step was necessary to create and manage databases, user roles, and provide a platform for secure data handling within the lab environment.

1. **Update the system's package list**:
    - Command (bash):
        1. sudo apt update
2. This command ensures that the system has the latest version information for all packages.
3. **Install PostgreSQL and its supporting tools**:
    - Command (bash):
        1. sudo apt install postgresql postgresql-contrib
4. This installs both the core PostgreSQL database server (postgresql) and additional useful features (postgresql-contrib), such as various extensions that enhance database functionality.
5. **Check the status of the PostgreSQL service**:
    - Command (bash):
        1. sudo systemctl status postgresql
6. This command ensures that the PostgreSQL service is up and running after installation. The expected output should indicate that the service is **active (running)**.
7. **Verify the PostgreSQL version installed**:
    - Command (bash):
        1. psql --version
8. This confirms that PostgreSQL has been installed correctly, and provides the version number of PostgreSQL.

No major issues were encountered during the installation process itself. The PostgreSQL package was installed smoothly, and the service started successfully.

Step 2: PostgreSQL Service Configuration

The objective of this step was to ensure that the PostgreSQL service starts automatically after system reboots, making it reliable for continuous use in the home lab environment. This step also involved verifying that the service was running correctly and that the necessary configurations were in place.

1. **Enable PostgreSQL to start on boot**:
   - Command (bash):
     1. sudo systemctl enable postgresql
2. This command ensures that PostgreSQL will automatically start whenever the system is booted or restarted.
3. **Check the current status of the PostgreSQL service**:
   - Command (bash):
     1. sudo systemctl status postgresql
4. This command checks if PostgreSQL is running. The expected output should show that PostgreSQL is **active (running)**.
5. **Verify if PostgreSQL is enabled to start on boot**:
   - Command (bash):
     1. sudo systemctl is-enabled postgresql
6. This verifies that the PostgreSQL service has been successfully configured to start automatically upon system boot. The expected output should be **enabled**.

There were no issues encountered during this configuration step. The PostgreSQL service started successfully, and the command to enable the service on boot worked as expected.

Step 3: Database and User Management

The objective was to create a new database and set up user roles with different levels of access. This enables the management of data securely by assigning specific permissions (read-only or read-write) to different users, ensuring controlled access to the database.

1. **Switch to the postgres user**:
   - Command (bash):
     1. sudo -i -u postgres
2. This command makes it possible to switch to the postgres user, who has administrative rights to manage databases and users.
3. **Open the PostgreSQL prompt**:
   - Command (bash):

   1. psql

4. This command opens the PostgreSQL interactive terminal (psql), where its possible to execute SQL commands.
5. **Create a new database**:
   - Command (sql):
      1. CREATE DATABASE homelab_db;
6. This creates a new database named homelab_db that will be used for managing and testing the PostgreSQL environment.
7. **Create a read-only user**:
   - Command (sql):
      1. CREATE ROLE readonly_user WITH LOGIN PASSWORD 'ubuntu1';
      2. GRANT CONNECT ON DATABASE homelab_db TO readonly_user;
      3. GRANT SELECT ON ALL TABLES IN SCHEMA public TO readonly_user;
8. This creates a user named readonly_user with a login password (ubuntu1). The user is granted permission to connect to the homelab_db database and is given **read-only** access to all tables in the public schema.
9. **Create a read-write user**:
   - Command (sql):
      1. CREATE ROLE write_user WITH LOGIN PASSWORD 'write_password';
      2. GRANT CONNECT ON DATABASE homelab_db TO write_user;
      3. GRANT INSERT, UPDATE, DELETE ON ALL TABLES IN SCHEMA public TO write_user;
10. This creates a user named write_user with a login password (write_password). The user is granted permission to connect to the homelab_db database and is given **read-write** access to all tables in the public schema (can insert, update, and delete data).
11. **Verify the roles and permissions**:
    - Command (sql):
       1. \du
12. This lists all roles (users) in the PostgreSQL environment, showing their attributes such as login privileges and permissions.

Encountered a small issue when switching back and forth between users to test permissions. Specifically, I initially forgot to log in as the PostgreSQL superuser (postgres) before running certain commands, which caused permission issues.

- **Solution**: Resolved the permission issue by ensuring that I was logged in as the postgres superuser before managing roles and permissions.
    - **Command (bash)**:
        - sudo -i -u postgres

Additionally, verified the roles by testing different user logins and checking access permissions with both readonly_user and write_user.

Step 4: Data Management and Optimization

The objective was to create tables, insert data, and optimize the performance of database queries. This step involved structuring data within the homelab_db database and using indexing techniques to improve the speed of query execution.

1. **Switch to the database (bash)**:
    - Command:
        1. psql -U postgres -d homelab_db
2. This command switches to the homelab_db database as the postgres superuser, making it possible to manage tables and data within the database.
3. **Create a table for storing user data**:
    - Command (sql):
        1.
```
CREATE TABLE users (
    id SERIAL PRIMARY KEY,
    username VARCHAR(50)
);
```
4. This creates a users table with two columns: id (a serial primary key that auto-increments) and username (a variable-length string).
5. **Insert data into the users table**:
    - Command (sql):
        1. INSERT INTO users (username) VALUES ('admin');
6. This inserts a row into the users table with the value admin in the username column.
7. **Query the users table**:
    - Command (sql):
        1. SELECT * FROM users;
8. This retrieves all rows from the users table, allowing verification of the inserted data.

9. **Create an index on the username column**:
   - Command (sql):
     1. CREATE INDEX idx_username ON users (username);
10. This creates an index on the username column, which helps improve the performance of queries that search by username.
11. **Analyze the performance of a query**:
    - Command (sql):
      1. EXPLAIN SELECT * FROM users WHERE username = 'admin';
12. This command provides a query execution plan, showing how PostgreSQL would execute the query and whether the index is being used.
    - **Before Indexing**: Without an index, the output shows that PostgreSQL performs a **sequential scan**, which checks every row in the table.
    - **After Indexing**: With the index in place, the output shows an **index scan**, which is much faster for large datasets because it only checks relevant rows.
13. **Populate the users table with more data (bulk insertion)**:
    - Command (sql):
      1.
      ```
      DO $$
      BEGIN
          FOR i IN 1..1000 LOOP
              INSERT INTO users (username) VALUES ('user_' || i);
          END LOOP;
      END $$;
      ```
14. This block of code inserts 1,000 rows into the users table for performance testing. Each row contains a username like user_1, user_2, and so on.
15. **Verify the number of rows inserted**:
    - Command (sql):
      1. SELECT COUNT(*) FROM users;
16. This query returns the total number of rows in the users table, confirming the bulk insertion was successful.

No significant issues were encountered during this step. However, testing performance before and after indexing showed that the query was slower before the index was created. After the index was created, the query performance improved significantly.

There were no major issues that required troubleshooting. Followed a standard approach for performance optimization by:

1. Measuring the baseline query performance with EXPLAIN.
2. Adding an index to optimize the query.
3. Verifying the improvement using the EXPLAIN command again.

Step 5: Backup Configuration

The objective of this step was to create a reliable backup process for the homelab_db database. This included setting up manual and automated backups to ensure data can be restored in case of data loss or corruption.

Commands ran during the backup configuration

1. **Create a directory for storing backups**:
    - Command (bash):
        - mkdir -p /home/ubuntu/backups
2. This command creates a directory called backups in the ubuntu user's home directory to store PostgreSQL database backups.
3. **Perform a manual backup using pg_dump**:
    - Command (bash):
        - pg_dump -U postgres homelab_db > /home/ubuntu/backups/homelab_db_initial_backup.sql
4. This command creates a backup of the homelab_db database and saves it to a file called homelab_db_initial_backup.sql in the /home/ubuntu/backups directory. The pg_dump utility is used to export the database schema and data.
    - **Explanation**:
        - -U postgres: Specifies that the postgres user should perform the backup.
        - homelab_db: The name of the database to back up.
        - > /home/ubuntu/backups/homelab_db_initial_backup.sql: Redirects the backup output to a file.
5. **Verify that the backup file was created**:
    - Command (bash):
        - ls /home/ubuntu/backups/

6. This lists the contents of the /home/ubuntu/backups/ directory to ensure the backup file was created successfully. The expected output should include the file homelab_db_initial_backup.sql.
7. **Set up automated backups using cron**:
    - Command (bash):
        - crontab -e
8. This opens the crontab file for editing, where automated tasks are scheduled.
    - **Added the following line to schedule daily backups at 2 AM (bash)**:
        - 0 2 * * * pg_dump -U postgres homelab_db > /home/ubuntu/backups/homelab_db_$(date +\%F).sql

This cron job schedules a daily backup of the homelab_db database at 2 AM. The backup files will be saved in the /home/ubuntu/backups/ directory with the current date appended to the filename (e.g., homelab_db_2024-09-24.sql).

- **Explanation**:
    - 0 2 * * *: Schedules the command to run daily at 2:00 AM.
    - $(date +\%F): Appends the current date (in YYYY-MM-DD format) to the filename.

9. **Verify the cron job**:
    - Command (bash):
        - crontab -l
10. This command lists all active cron jobs to ensure that the automated backup job has been added correctly.

Initially, encountered a permission issue when running the pg_dump command as the default ubuntu user, which required using the postgres superuser.

- **Solution**: Switched to the postgres user and ran the pg_dump command as the postgres user:
    - Command (bash):
        - sudo -i -u postgres

This resolved the permission issue and allowed the backup process to proceed as expected. Additionally, ensured that the cron job was configured correctly to automate the backup process.

Step 6: Performance Optimization (VACUUM)

The objective of this step was to optimize database performance by cleaning up dead tuples (rows marked for deletion) and updating statistics using the VACUUM command. Regular maintenance of the database using VACUUM ensures that queries perform efficiently, and resources such as disk space are used effectively.

Commands Used During Performance Optimization:

1. **Run a manual VACUUM to clean up dead tuples**:
    - Command (sql):
        - VACUUM;
2. This command removes dead tuples from the database, freeing up space and making the database more efficient. It performs a lightweight cleanup but does not update statistics used by the PostgreSQL query planner.
3. **Run VACUUM ANALYZE for full optimization**:
    - Command (sql):
        - VACUUM ANALYZE;
4. This command performs a full cleanup by vacuuming the database and analyzing it to update query statistics. This helps the PostgreSQL query planner make better decisions on how to execute queries efficiently.
    - **Explanation**:
        - **VACUUM**: Removes dead tuples and frees up space.
        - **ANALYZE**: Updates query planner statistics to improve query performance.
5. **Verify the VACUUM and analyze its impact**:
    - Command (sql):
        - EXPLAIN SELECT * FROM users WHERE username = 'admin';
6. Running the EXPLAIN command before and after VACUUM ANALYZE allows for the comparison of the performance of queries. The output should indicate whether query performance has improved after the database has been vacuumed and analyzed.
7. **Automate VACUUM using cron for regular maintenance**:
    - Command (bash):
        - crontab -e
8. This opens the crontab for scheduling regular database maintenance.
    - **Add the following line to schedule a weekly VACUUM ANALYZE at 3 AM every Sunday (bash)**:
        - 0 3 * * 0 /usr/bin/vacuumdb -z -d homelab_db

This cron job automates the VACUUM ANALYZE process, ensuring the database is optimized weekly without manual intervention.

- **Explanation**:
  - 0 3 * * 0: Schedules the job to run every Sunday at 3:00 AM.
  - /usr/bin/vacuumdb: Utility to run VACUUM and ANALYZE.
  - -z: Tells the vacuumdb utility to perform both VACUUM and ANALYZE.
  - -d homelab_db: Specifies the homelab_db database to vacuum.

9. **Verify the cron job**:
   - Command (bash):
     - crontab -l
10. This command lists the active cron jobs to confirm that the VACUUM ANALYZE job has been scheduled properly.

There were no major issues during the VACUUM process. The commands executed as expected, and the performance improvement was noticeable after running VACUUM ANALYZE.

Step 7: Log Monitoring and Integration with Graylog

The objective was to set up detailed logging for PostgreSQL and configure Filebeat to forward these logs to Graylog for monitoring and analysis. This ensures that all database activity is tracked and logged, providing valuable insights for system monitoring and troubleshooting.

1. **Enable detailed logging in PostgreSQL**:
   - Command (bash):
     1. sudo nano /etc/postgresql/16/main/postgresql.conf
2. This opens the PostgreSQL configuration file for editing.
   - **Update the following logging-related settings** to ensure detailed logs are generated (bash):
     1. logging_collector = on
     2. log_directory = '/var/log/postgresql'
     3. log_filename = 'postgresql-%Y-%m-%d.log'
     4. log_statement = 'all'
     5. log_duration = on

These settings enable detailed logging of all SQL statements executed in PostgreSQL, including their execution times. Logs are stored in the /var/log/postgresql/ directory with the current date appended to the filename.

3. **Create the log directory (if not already existing)**:
    - Command (bash):

```bash
sudo mkdir -p /var/log/postgresql
sudo chown postgres:postgres /var/log/postgresql
```

4. This command ensures the log directory exists and is owned by the postgres user so that PostgreSQL can write logs to it.
5. **Restart PostgreSQL to apply the changes**:
    - Command (bash):

```bash
sudo systemctl restart postgresql@16-main
```

6. This restarts the PostgreSQL service, applying the new logging settings.
7. **Install Filebeat**:
    - Command (bash):

```bash
sudo apt install filebeat
```

8. Filebeat is a lightweight log forwarder used to send PostgreSQL logs to Graylog.
9. **Configure Filebeat** to send PostgreSQL logs to Graylog:
    - Command (bash):

```bash
sudo nano /etc/filebeat/filebeat.yml
```

10. This opens the Filebeat configuration file.
    - **Add the following configuration to forward PostgreSQL logs (yaml)**:

```yaml
filebeat.inputs:
- type: log
  enabled: true
  paths:
    - /var/log/postgresql/*.log

output.logstash:
  hosts: ['<GRAYLOG_SERVER_IP>:5044']
```

This configuration tells Filebeat to monitor the PostgreSQL log files in the /var/log/postgresql/ directory and forward them to Logstash (or directly to Graylog) running on <GRAYLOG_SERVER_IP> over port 5044.

11. **Restart Filebeat** to apply the changes:
    - Command (bash):
      1. sudo systemctl restart filebeat
12. This command restarts the Filebeat service, ensuring it starts forwarding logs to Graylog.
13. **Verify that logs are being sent to Graylog**:
    - Command (bash):
      1. sudo tail -f /var/log/postgresql/postgresql-$(date +\%Y-\%m-\%d).log
14. This command monitors the PostgreSQL log file in real-time, ensuring that logs are being written correctly.
    - In Graylog, checked the input and confirmed that the logs were arriving from Filebeat by searching for PostgreSQL log entries in the Graylog web interface.

Troubleshooting

- **Issue 1**: Initially encountered a conflict in Filebeat where both the Elasticsearch and Logstash outputs were enabled, causing the service to fail.
  - **Solution**: Disabled the Elasticsearch output in the filebeat.yml file by commenting out the following line (yaml):
    - #output.elasticsearch:
- **Issue 2**: The PostgreSQL logs were not being written to the expected pg_log directory.
  - **Solution**: Corrected the log directory setting in the postgresql.conf file to (bash):
    - log_directory = '/var/log/postgresql'

and created the directory manually. After restarting PostgreSQL, the logs appeared correctly.

- **Solution to Filebeat issue**: Edited the filebeat.yml file to ensure that only the Logstash output was enabled, which allowed Filebeat to start successfully.
- **Solution to log directory issue**: Corrected the log directory setting and manually created the required directory, resolving the issue with log file locations.

Step 8: Enabling SSL and Generating Keys for PostgreSQL

The objective was to secure connections to PostgreSQL by enabling SSL (Secure Sockets Layer). This ensures that data transmitted between the PostgreSQL server and clients is encrypted, providing enhanced security for sensitive information.

1. **Enable SSL in the PostgreSQL configuration file**:
   - Command (bash):
     1. sudo nano /etc/postgresql/16/main/postgresql.conf
2. This opens the PostgreSQL configuration file for editing.
   - **Update the following SSL-related settings** to enable SSL (bash):
     1. ssl = on
     2. ssl_cert_file = '/etc/ssl/certs/server.crt'
     3. ssl_key_file = '/etc/ssl/private/server.key'

These settings enable SSL, specifying the certificate file (server.crt) and the private key file (server.key) that PostgreSQL will use for encrypted connections.

3. **Generate the SSL private key**:
   - Command (bash):
     1. sudo openssl genrsa -out /etc/ssl/private/server.key 2048
4. This command generates a 2048-bit private key and stores it in /etc/ssl/private/server.key.
5. **Set the appropriate permissions for the private key**:
   - Command (bash):
     1. sudo chmod 600 /etc/ssl/private/server.key
6. This ensures that the private key file is only accessible by the root user, improving security.
7. **Generate the SSL certificate**:
   - Command (bash):
     1. sudo openssl req -new -x509 -key /etc/ssl/private/server.key -out /etc/ssl/certs/server.crt -days 365 -subj "/CN=localhost"
8. This command generates a self-signed SSL certificate valid for 365 days and stores it in /etc/ssl/certs/server.crt. The -subj "/CN=localhost" option sets the Common Name (CN) to localhost.
9. **Set the appropriate permissions for the SSL certificate**:
   - Command (bash):
     1. sudo chmod 644 /etc/ssl/certs/server.crt
10. This allows the certificate to be readable by the PostgreSQL service and other users but keeps it secure.
11. **Restart PostgreSQL to apply SSL settings**:
    - Command (bash):

    1. sudo systemctl restart postgresql@16-main

12. This command restarts the PostgreSQL service to apply the new SSL configuration.
13. **Verify SSL is enabled in PostgreSQL**:
    - Command (sql):
        1. SHOW ssl;
14. This SQL command confirms that SSL is enabled in PostgreSQL. The expected output should be:
    - bash
    - Copy code
    - ssl
    - ----
    - on
15. **Test the SSL connection**:
    - Command (bash):
        1. psql -U readonly_user -d homelab_db -h localhost
16. This command tests the connection to the PostgreSQL server using the readonly_user account. After logging in, you can verify the connection is encrypted with SSL.
    - **Command to verify SSL connection (sql)**:
        1. \conninfo

This command displays information about the current connection, including whether SSL is being used. The output should indicate that SSL is in use, similar to (bash):

- SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, compression: off)

Troubleshooting

- **Issue 1**: When attempting to restart PostgreSQL after enabling SSL, the service failed due to missing SSL files.
    - **Solution**: I had forgotten to generate the necessary SSL certificate and private key before enabling SSL. After generating the required files and setting appropriate permissions, PostgreSQL started successfully.
- **Issue 2**: Incorrect permissions on the private key file initially caused PostgreSQL to fail at startup.

    - **Solution**: Corrected the permissions of the private key file to ensure it was only accessible by root using the following command (bash):
        - sudo chmod 600 /etc/ssl/private/server.key

- **Solution to missing SSL files**: Generated a self-signed SSL certificate and private key using OpenSSL, then set the necessary permissions for PostgreSQL to access them.
- **Solution to private key permissions**: Adjusted the file permissions to restrict access to the private key, allowing PostgreSQL to start without security concerns.

Troubleshooting Overview

During the PostgreSQL setup and configuration, encountered several issues that required troubleshooting. Below is a detailed account of the problems, the steps taken to diagnose them, and how they were resolved.

Issue 1: Peer Authentication Failed for the postgres User

Description:

When I attempted to log in to the PostgreSQL database using the postgres superuser, I encountered an authentication failure caused by the default peer authentication method.

Symptoms:

- Error message (bash):
    - psql: error: connection to server on socket "/var/run/postgresql/.s.PGSQL.5432" failed: FATAL: Peer authentication failed for user "postgres"

Steps Taken to Diagnose:

1. Checked the PostgreSQL authentication method in the pg_hba.conf file:
    - Command (bash):
        1. sudo nano /etc/postgresql/16/main/pg_hba.conf
    2. Identified that the postgres user was using **peer** authentication for local connections, which checks for a matching system username.

Resolution:

1. Modified the authentication method for the postgres user from peer to md5 (password-based authentication).
    - Change made in pg_hba.conf (bash):
        1. local   all   postgres   md5
2. Restarted the PostgreSQL service to apply the changes:

- Command (bash):
    1. sudo systemctl restart postgresql@16-main
3. Verified that it was now possible to log in as the postgres user using a password:
    - Command (bash):
        1. psql -U postgres -h localhost

Issue 2: Filebeat Service Failing to Start

Description:

While trying to start the Filebeat service to forward PostgreSQL logs to Graylog, the service failed due to a configuration conflict where both the Elasticsearch and Logstash outputs were enabled.

Symptoms:

- Filebeat failed to start with the following error message (bash):
    - Exiting: error unpacking config data: more than one namespace configured accessing 'output'

Steps Taken to Diagnose:

1. Checked the Filebeat log for errors:
    - Command (bash):
        1. sudo journalctl -u filebeat
2. Identified that both the **Elasticsearch** and **Logstash** outputs were configured, which caused a conflict.

Resolution:

1. Edited the Filebeat configuration file to disable the Elasticsearch output and retain the Logstash output.
    - Command (bash):
        1. sudo nano /etc/filebeat/filebeat.yml
    2. Commented out the following lines (yaml):
        1. #output.elasticsearch:
2. Restarted the Filebeat service:
    - Command (bash):
        1. sudo systemctl restart filebeat

3. Verified that Filebeat was running successfully:
   - Command (bash):
     1. sudo systemctl status filebeat

Issue 3: PostgreSQL Logs Not Appearing in the Expected Directory

Description:

After configuring PostgreSQL to store logs in a specific directory (pg_log), discovered that logs were not being written to the expected location.

Symptoms:

- When checking the /var/log/postgresql/pg_log directory, no logs were present.

Steps Taken to Diagnose:

1. Checked the PostgreSQL configuration to verify the log directory setting.
   - Command:

bash

Copy code

sudo nano /etc/postgresql/16/main/postgresql.conf

2. Found that the log_directory was set incorrectly to pg_log.

Resolution:

1. Modified the log_directory setting in the PostgreSQL configuration file to (bash):
   - log_directory = '/var/log/postgresql'
2. Created the /var/log/postgresql directory manually:
   - Command (bash):
     1. sudo mkdir -p /var/log/postgresql
     2. sudo chown postgres:postgres /var/log/postgresql
3. Restarted PostgreSQL to apply the changes:
   - Command (bash):
     1. sudo systemctl restart postgresql@16-main
4. Verified that logs were now being written to the correct location:
   - Command (bash):
     1. sudo ls /var/log/postgresql/

Issue 4: PostgreSQL Service Failing to Start After Enabling SSL

Description:

After enabling SSL in the PostgreSQL configuration file, the service failed to start due to missing SSL certificate and key files.

Symptoms:

- Error message when restarting PostgreSQL (bash):
  - FATAL: could not load private key file "/etc/ssl/private/server.key": No such file or directory

Steps Taken to Diagnose:

1. Checked the PostgreSQL log for details about the error:
   - Command (bash):
     1. sudo tail -f /var/log/postgresql/postgresql-$(date +\%Y-\%m-\%d).log
2. Identified that the issue was caused by the missing SSL certificate and private key files.

Resolution:

1. Generated the required SSL private key and certificate using OpenSSL:
   - **Generate the private key (bash)**:
     1. sudo openssl genrsa -out /etc/ssl/private/server.key 2048
   - **Generate the self-signed certificate (bash)**:
     1. sudo openssl req -new -x509 -key /etc/ssl/private/server.key -out /etc/ssl/certs/server.crt -days 365 -subj "/CN=localhost"
2. Set the correct file permissions for the private key and certificate:
   - Command (bash):
     1. sudo chmod 600 /etc/ssl/private/server.key
     2. sudo chmod 644 /etc/ssl/certs/server.crt
3. Restarted PostgreSQL to apply the SSL configuration:
   - Command (bash):
     1. sudo systemctl restart postgresql@16-main
4. Verified that SSL was enabled by running the following SQL command:
   - Command (sql):

```
1. SHOW ssl;
```

Conclusion

Throughout the project, several challenges were encountered:

- **Authentication issues** were resolved by modifying the PostgreSQL pg_hba.conf file to switch from peer to md5 authentication.
- **Filebeat configuration issues** were resolved by disabling the conflicting Elasticsearch output, allowing the service to start successfully.
- **Log file issues** were resolved by correcting the PostgreSQL log directory settings and ensuring logs were written to the correct location.
- **SSL configuration issues** were addressed by generating the necessary SSL certificate and private key, then applying the correct permissions to ensure PostgreSQL could access them securely.

This PostgreSQL project demonstrates a strong understanding of database management, security practices, and system integration. The skills gained here—especially in SSL configuration, log monitoring, and automated backups—will be directly applicable in future projects. With the foundation laid in this PostgreSQL project, the skills and configurations can be extended to more complex projects such as building a full **Security Operations Center (SOC)**. The integration of secure database practices, automated backups, log monitoring, and performance optimization aligns with the operational needs of a SOC, where data security, availability, and monitoring are critical.

Home Lab - Configuring Apache Web Server for Ubuntu VM

Primary Objective:

To set up and configure the Apache web server on an Ubuntu virtual machine within a home lab environment. The goal is to provide a functional web server accessible from the VM's network, enabling basic web hosting capabilities and facilitating future web application testing.

Step 1: Installation of Apache1. Objective of Installing Apache

The goal of installing Apache was to set up a reliable and customizable web server environment as part of the home lab project. Apache would act as the primary web server for hosting services and tools within the lab, providing web-based access for future configurations, monitoring tools, and other SOC-related applications.

Commands for Apache Installation

- To install Apache, the following commands were run:
    1. **System Update**: First, the package list and system were updated (bash):
        - sudo apt update && sudo apt upgrade
    2. **Install Apache**: Installed Apache using the apt package manager:
    - sudo apt install apache2

There were **no issues** during the initial installation of Apache. The installation process was straightforward, and Apache was installed without any errors.

Step 2: Starting and Enabling Apache

After installation, the objective was to start the Apache service and ensure that it automatically starts every time the system boots, providing continuous availability for any services hosted on the web server.

Commands for Starting and Enabling Apache

1. **Start Apache (bash)**:
    - sudo systemctl start apache2
2. **Enable Apache to Start on Boot (bash)**:
    - sudo systemctl enable apache2
3. **Check Apache Status**: Confirmed that Apache was running successfully (bash):

- sudo systemctl status apache2

Port Conflict: Apache initially failed to start due to a port conflict with another service (Nextcloud) that was using port 80.

Nextcloud is a file-sharing and collaboration platform installed on the server. It was occupying port 80, which is the default port for Apache.

The issue was resolved by stopping the Nextcloud service that was using port 80, freeing the port for Apache:

1. **Stop Nextcloud (bash)**:
    - sudo snap stop nextcloud
2. **Restart Apache (bash)**:
    - sudo systemctl restart apache2

After this, Apache successfully started, and the service was confirmed to be running.

Step 3: Testing Apache

After starting and enabling Apache, the goal was to verify that Apache was functioning properly by accessing the web server through a browser, ensuring it could serve the default web page.

Commands for Testing Apache

1. **Test Locally**: Tested Apache from the Ubuntu server using the curl command (bash):
    - curl http://localhost
    - This confirmed that Apache was serving the default web page on the local server.
2. **Test Externally**: The web server was also tested from a browser on the host machine by navigating to the Ubuntu VM's IP address (bash):
    - http://192.168.1.100

No issues were encountered during testing. The default Apache welcome page was successfully displayed, confirming that the web server was functioning as expected.

For enhanced security, a firewall can be configured using UFW to allow traffic on port 80 (HTTP) and port 443 (HTTPS). This wasn't performed during the initial setup but can be added to protect access in future configurations.

Conclusion

The Apache web server was successfully installed and configured within the Ubuntu Server VM as part of the home lab project. The setup process was smooth, aside from a port conflict with

Nextcloud, which was resolved efficiently by stopping the conflicting service. Once resolved, Apache was able to bind to port 80 and serve the default web page, verifying that the web server is operational.

This configuration is pivotal for hosting web-based services, providing the foundational infrastructure for future projects such as Security Operations Center (SOC) tools that require a web interface.

Apache is now fully configured as the primary web server for the home lab environment, capable of supporting various web-based tools and services to be deployed in future phases of the project.

Key Resolutions:

1. **Issue Encountered:** A port conflict with Nextcloud was the only major challenge encountered during the setup process.
2. **Troubleshooting Steps:** The steps to resolve the conflict involved stopping Nextcloud and restarting Apache. These steps were documented for clarity, ensuring any future conflicts can be addressed quickly.

Further Considerations:

- **Firewall Configuration (Optional):** Although UFW wasn't configured during this setup, adding firewall rules could further enhance security. This step could be revisited later to strengthen the server's defenses.
- **Advanced Testing (Optional):** The initial test confirmed that Apache was serving the default web page. Future updates could include more advanced testing, such as configuring virtual hosts, setting up SSL, or hosting sample applications. These steps were left for future consideration but can greatly expand the web server's functionality.
    - **Virtual Hosts**: Virtual hosts allow Apache to host multiple websites on a single server. More information can be found in the Apache Documentation on Virtual Hosts.
    - **SSL Setup**: Securing the web server using SSL can protect data in transit. For basic SSL setup with Apache, see the Let's Encrypt Guide for SSL.

These steps were left for future consideration but can greatly expand the web server's functionality.

Home Lab - Setting Up Docker on Ubuntu VM

Primary Objective:

To install, configure, and manage Docker on an Ubuntu VM for the purpose of running containerized services. This project focuses on deploying Docker to isolate and manage applications efficiently, specifically setting up WordPress and MySQL services within Docker containers. The goal is to ensure seamless deployment, scalability, and portability of services within a containerized environment. By the end of this project, Docker and Docker Compose will be fully operational, enabling the creation and management of multi-container applications, with troubleshooting steps documented to address any issues encountered during the configuration.

1. Installing Docker

The goal was to install and configure Docker as the primary platform to manage containerized services, specifically to run WordPress and MySQL, as part of the home lab setup. Docker was chosen to create isolated environments for services, ensuring easy deployment and scaling within the lab.

Commands Used During Installation

The following commands were run step by step to install Docker:

1. **Update the package list (bash)**:
    - sudo apt update
2. **Install Docker (bash)**:
    - sudo apt install docker.io
3. **Start Docker and enable it to run on startup (bash)**:
    - sudo systemctl start docker
    - sudo systemctl enable docker
4. **Verify Docker installation (bash)**:
    - docker --version

After running these commands, Docker was successfully installed and set to start automatically on system boot.

No issues were encountered during the installation of Docker.

2. Docker Compose Installation

The objective of installing Docker Compose was to manage multi-container applications, specifically to set up and run WordPress and MySQL services in separate containers but with shared configurations. Docker Compose simplifies managing services by allowing the use of a single YAML configuration file for defining and running these containers.

Commands Used During Installation

The following commands were used to install Docker Compose:

- **Download Docker Compose binary (bash)**:
  - sudo curl -L "https://github.com/docker/compose/releases/download/2.20.2/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
- **Apply executable permissions to the binary (bash)**:
  - sudo chmod +x /usr/local/bin/docker-compose
- **Verify Docker Compose installation (bash)**:
  - docker-compose --version

These commands ensured that Docker Compose was installed and ready to be used for managing the containers in the lab setup.

Issues Encountered During Installation: There was a small issue where the first attempt to install Docker Compose resulted in the default system version (1.29.2) being installed, which led to incompatibility with some of the features required for the WordPress and MySQL setup.

How Issues Were Resolved: To resolve the issue, Docker Compose was manually updated by downloading the latest binary version (2.20.2) from GitHub. The older version was removed, and the latest version was installed, ensuring that all required features were available. After reinstalling and verifying the version, Docker Compose was fully functional.

---

3. WordPress and MySQL Container Setup

The objective was to set up WordPress and MySQL in Docker containers using Docker Compose. The goal was to create a fully functional WordPress site, backed by a MySQL database, that could be easily managed and tested in the home lab environment. Docker Compose was used to define and run the services, ensuring they communicated with each other through a network bridge.

Step 1: Create a docker-compose.yml file

- nano docker-compose.yml

The following docker-compose.yml file was created to define the services (yaml):

```
version: "3"
services:
 db:
  image: mysql:5.7
  environment:
   MYSQL_ROOT_PASSWORD: example
   MYSQL_DATABASE: wordpress
   MYSQL_USER: wordpress
   MYSQL_PASSWORD: wordpresspass

 wordpress:
  image: wordpress:latest
  ports:
   - "8080:80"
  environment:
   WORDPRESS_DB_HOST: db
   WORDPRESS_DB_NAME: wordpress
   WORDPRESS_DB_USER: wordpress
   WORDPRESS_DB_PASSWORD: wordpresspass
```

Step 2: Start the containers

The containers were started with the following command (bash):

- `sudo docker-compose up -d`

This command launched the WordPress and MySQL containers in detached mode, allowing them to run in the background.

Issues Encountered During Setup

1. **Connection Issues Between WordPress and MySQL**:
   - **Error**: "Error establishing a database connection."
   - **Cause**: This occurred because WordPress tried to connect to MySQL before MySQL was fully initialized, resulting in a connection failure.

2. **Database Initialization Delay**:
   - **Cause**: MySQL wasn't fully initialized when WordPress attempted to connect, which led to connection issues.
3. **Missing Environment Variables**:
   - **Cause**: Some environment variables (such as WORDPRESS_DB_HOST, WORDPRESS_DB_NAME, etc.) were not properly passed to the WordPress container, causing fallback values (e.g., "example username" and "example password") to be used.
4. **Missing Tables in MySQL**:
   - **Cause**: There were warnings about missing tables in the MySQL logs because the MYSQL_DATABASE environment variable wasn't properly set, preventing the WordPress database from being created.

How Issues Were Resolved

1. **Connection Issues Between WordPress and MySQL**:
   - **Resolution**: The depends_on directive was added to the docker-compose.yml file to ensure that WordPress would wait for MySQL to start before attempting to establish a connection.
     - nano docker-compose.yml

The following changes were made to the docker-compose.yml file:

```
services:
  wordpress:
    depends_on:
      - db
```

This directive ensured that the WordPress container would not start until the MySQL container was running.

2. **Database Initialization Delay**:
   - **Resolution**: The same depends_on directive added to resolve the connection issue also resolved this problem by ensuring that WordPress would not attempt to connect until MySQL was fully initialized.
3. **Environment Variable Misconfiguration**:

- **Resolution**: The environment variables for the database connection were corrected in the docker-compose.yml file. The correct environment variables were added as follows:
    - nano docker-compose.yml

```
environment:
  WORDPRESS_DB_HOST: db
  WORDPRESS_DB_NAME: wordpress
  WORDPRESS_DB_USER: wordpress
  WORDPRESS_DB_PASSWORD: wordpresspass
```

After making these changes, the containers were rebuilt to apply the new configuration using the following commands (bash):

- sudo docker-compose down
- sudo docker-compose up -d

4. **Missing Tables in MySQL**:
    - **Resolution**: There were warnings about missing tables in the MySQL logs because the MYSQL_DATABASE environment variable was not set correctly. This was fixed by ensuring that the MYSQL_DATABASE variable was set to wordpress in the docker-compose.yml file:
        - nano docker-compose.yml

```
environment:
  MYSQL_DATABASE: wordpress
```

After restarting the containers with the corrected environment variables, MySQL successfully created the necessary tables, and the warning messages disappeared.

Step 3: Verifying the Setup

Once the containers were running, the WordPress site became accessible via the following URL:

- http://<server-ip>:8080

At this point, the WordPress installation screen appeared. After filling in the necessary details (such as site title, username, and password), the WordPress site was successfully set up and connected to the MySQL database.

By using Docker and Docker Compose, WordPress and MySQL were successfully deployed as containerized services. Issues related to connection delays, missing environment variables, and missing database tables were resolved by using Docker Compose directives and proper configuration adjustments.

Conclusion

The successful installation and configuration of Docker and Docker Compose allowed for the seamless deployment of a WordPress site and its MySQL database in a containerized environment. By leveraging Docker, we achieved the following:

- **Isolated Services**: WordPress and MySQL were deployed as independent containers, allowing them to run in isolation while ensuring efficient communication between them.
- **Reproducibility**: The use of Docker Compose provided a simplified, reproducible method for configuring and starting multiple services, ensuring that the entire setup could be easily replicated or deployed elsewhere.
- **Troubleshooting Efficiency**: Docker's logging features, combined with methodical troubleshooting steps, made it easy to identify and resolve issues related to database initialization, environment variables, and container communication.
- **Scalability**: This containerized approach provides the flexibility to scale services or add new applications to the environment without significant changes to the underlying infrastructure.

By addressing all issues encountered during setup, including database connectivity errors and configuration mismatches, we ensured the stability and reliability of the deployed services. This project demonstrates the practical use of containerization technology in managing services and applications efficiently, forming a solid foundation for future development within the home lab.

Home Lab - Setting Up OpenVPN on Ubuntu VM

Primary Objective:

To install, configure, and secure an OpenVPN server to allow remote VPN connections to the internal network. This setup will provide secure access to internal resources for authorized clients while ensuring that network traffic is encrypted and securely routed through the VPN tunnel. The project focuses on generating necessary cryptographic certificates, configuring the OpenVPN server, enabling firewall rules, and testing the VPN connection to ensure secure remote access.

By the end of this project, the OpenVPN server should be fully operational, allowing clients to connect securely, with troubleshooting steps documented along the way to resolve any issues encountered during the configuration.

1. Installation of OpenVPN

The objective of installing OpenVPN was to set up a secure VPN server that would allow clients to remotely and securely access internal network resources, ensuring encrypted data transmission and secure connections for the home lab setup.

Commands run during the installation:

1. **Update and Install OpenVPN and Easy-RSA (bash):**
   - sudo apt update
   - sudo apt install openvpn easy-rsa

These commands ensure the system is up to date and install the necessary packages for OpenVPN and Easy-RSA, which are required to generate certificates.

2. **Set up the PKI (Public Key Infrastructure) Directory (bash):**
   - make-cadir ~/openvpn-ca
   - cd ~/openvpn-ca
   - ./easyrsa init-pki

This created a PKI directory, initialized it, and moved into the directory to begin configuring Easy-RSA for certificate generation.

3. **Edit the vars file (bash):**
   - nano vars

This command opened the Easy-RSA configuration file where the certificate details were customized, such as, country, province, city, organization, and email.

Issues encountered during the installation:

- **Certificate Authority already exists error:**
  During the certificate generation process, encountered an error stating "Unable to create a CA as you already seem to have one set up."

How the issues were resolved:

- **Resolution for CA error:** This was resolved by running (bash):
    - ./easyrsa init-pki

This command re-initialized the PKI, allowing the certificate generation process to start from scratch.

---

2. Certificate Generation

The objective of the certificate generation step was to create the necessary certificates and keys for both the server and client to ensure secure communication through the VPN. This included generating the CA (Certificate Authority), server certificate, client certificate, and Diffie-Hellman parameters, all essential for establishing encrypted connections.

Commands run during certificate generation:

1. **Build the Certificate Authority (CA) (bash):**
    - ./easyrsa build-ca

This command created the root Certificate Authority, which is the foundation of trust for the VPN.

2. **Generate the Server Certificate and Key (bash):**
    - ./easyrsa gen-req server nopass
    - ./easyrsa sign-req server server

The first command generated the server's private key and certificate signing request, while the second command signed the server's certificate using the CA.

3. **Generate Diffie-Hellman Parameters (bash):**
    - sudo openssl dhparam -out /etc/openvpn/dh.pem 2048

This command created the Diffie-Hellman parameters, which are essential for securely exchanging cryptographic keys.

4. **Copy Server Certificates and Keys (bash):**
    - sudo cp /home/ubuntu/openvpn-ca/pki/ca.crt /etc/openvpn/
    - sudo cp /home/ubuntu/openvpn-ca/pki/issued/server.crt /etc/openvpn/

- sudo cp /home/ubuntu/openvpn-ca/pki/private/server.key /etc/openvpn/
- sudo cp /etc/openvpn/dh.pem /etc/openvpn/

These commands moved the generated certificates and keys to the appropriate directory for OpenVPN.

5. **Generate the Client Certificate and Key (bash):**

   - ./easyrsa gen-req ubuntu nopass
   - ./easyrsa sign-req client ubuntu

Similar to the server, these commands generated the client's certificate and private key, and then signed the client certificate using the CA.

6. **Copy Client Certificates and Keys (bash):**

   - sudo cp /home/ubuntu/openvpn-ca/pki/issued/ubuntu.crt /etc/openvpn/
   - sudo cp /home/ubuntu/openvpn-ca/pki/private/ubuntu.key /etc/openvpn/

Issues encountered during certificate generation:

- **Certificate Already Exists Error:**
  During the process of generating certificates encountered an error stating that the Certificate Authority already existed, preventing the creation of a new one.

How the issues were resolved:

- **Resolution for CA Already Exists Error:** Resolved the issue by re-initializing the PKI (bash):
  - ./easyrsa init-pki

After re-initialization, the CA certificate was successfully built.

---

3. OpenVPN Server Configuration

The objective of this section was to configure the OpenVPN server to use the generated certificates and keys for secure communication, and to define the VPN network, encryption methods, and other essential server settings.

Commands run during OpenVPN server configuration:

1. **Create and Edit the OpenVPN Server Configuration File:** Created and edited the OpenVPN server configuration file to specify key parameters. (bash)

- sudo nano /etc/openvpn/server.conf

The following configuration was added (yaml):

```
port 1194
```

```
proto udp
dev tun
ca ca.crt
cert server.crt
key server.key
dh dh.pem
server 10.8.0.0 255.255.255.0
push "redirect-gateway def1 bypass-dhcp"
push "dhcp-option DNS 8.8.8.8"
push "dhcp-option DNS 8.8.4.4"
keepalive 10 120
cipher AES-256-CBC
persist-key
persist-tun
user nobody
group nogroup
status /var/log/openvpn-status.log
log-append /var/log/openvpn.log
verb 3
```

2. **Enable IP Forwarding:** IP forwarding is essential for routing traffic between the VPN clients and the internal network. Edited the sysctl configuration file to enable IP forwarding. (bash)

    - sudo nano /etc/sysctl.conf

Ensured that the following line was present and uncommented (yaml):

- net.ipv4.ip_forward = 1

After editing the file, applied the changes with (bash):

- sudo sysctl -p

3. **Configure Firewall Rules for OpenVPN:** UFW (Uncomplicated Firewall) was configured to allow OpenVPN traffic on port 1194 (UDP). (bash)

    - sudo ufw allow 1194/udp
    - sudo ufw allow OpenSSH

- sudo ufw disable
- sudo ufw enable

Verified that UFW was active and that the correct rules were in place (bash):

- sudo ufw status

Issues encountered during server configuration:

- **Server Fails to Start:** When attempting to start the OpenVPN server, encountered an error indicating that the service failed to start due to configuration issues or missing certificates.

How the issues were resolved:

1. **Troubleshooting Service Startup Failure:** Used the following commands to check the status of the service and view logs for further troubleshooting (bash):

    - sudo systemctl start openvpn@server
    - sudo systemctl status openvpn@server
    - sudo journalctl -xeu openvpn@server.service

The issue was resolved by ensuring that the certificate and key files were correctly placed in the /etc/openvpn/ directory and that the server configuration file had the correct paths for these files.

2. **File Permissions:** Checked and updated file permissions on the certificates and keys to ensure the OpenVPN service had the required access.

---

4. Firewall Configuration

The objective of this section was to configure the firewall (UFW) to allow necessary traffic for the OpenVPN server to function, while also ensuring that SSH access remains available.

Commands run during firewall configuration:

1. **Allow OpenVPN traffic on UDP port 1194:** This command allows traffic on the UDP port where OpenVPN is listening. (bash)

    - sudo ufw allow 1194/udp

2. **Allow OpenSSH traffic:** To maintain SSH access while configuring the firewall. (bash)

    - sudo ufw allow OpenSSH

3. **Disable UFW (Firewall):** To reset and re-enable the firewall with the correct settings. (bash)

    - sudo ufw disable

4. **Enable UFW (Firewall):** Enable the firewall again with updated rules. (bash)
    - sudo ufw enable
5. **Check UFW Status:** Verify that the firewall is active and the correct rules are applied. (bash)
    - sudo ufw status

Issues encountered during firewall configuration:

There were no major issues encountered during the firewall configuration. The UFW rules were successfully applied, and the firewall was re-enabled.

Troubleshooting steps:

- After re-enabling UFW, confirmed that OpenVPN and SSH ports were allowed by running sudo ufw status to ensure that the correct ports were listed.

Once this was complete, the firewall was configured to allow OpenVPN traffic while keeping SSH access open for remote management.

---

5. Client Configuration

The objective of this section was to configure the OpenVPN client, generate the necessary client certificates and keys, and create a client configuration file to connect to the OpenVPN server.

Commands run during client configuration:

1. **Generate the client certificate and key:** This command generates the client's certificate and key files for secure communication with the OpenVPN server. (bash)
    - ./easyrsa gen-req ubuntu nopass
    - ./easyrsa sign-req client ubuntu
2. **Copy the client certificate and key to the OpenVPN directory:** The following commands were used to copy the client certificate and key files to the /etc/openvpn/ directory for easier access. (bash)
    - sudo cp /home/ubuntu/openvpn-ca/pki/issued/ubuntu.crt /etc/openvpn/
    - sudo cp /home/ubuntu/openvpn-ca/pki/private/ubuntu.key /etc/openvpn/
3. **Create and edit the client configuration file (ubuntu.ovpn):** Created the client configuration file, which contains the client certificate, key, and CA certificate required for the VPN connection. (bash)

- sudo nano /etc/openvpn/ubuntu.ovpn

Example content of the ubuntu.ovpn file:

bash

Copy code

```
client
dev tun
proto udp
remote [YourServerIP] 1194
resolv-retry infinite
nobind
persist-key
persist-tun
remote-cert-tls server
cipher AES-256-CBC
verb 3

<ca>
-----BEGIN CERTIFICATE-----
# Contents from /etc/openvpn/ca.crt
-----END CERTIFICATE-----
</ca>

<cert>
-----BEGIN CERTIFICATE-----
# Contents from /etc/openvpn/ubuntu.crt
-----END CERTIFICATE-----
</cert>

<key>
-----BEGIN PRIVATE KEY-----
```

Contents from /etc/openvpn/ubuntu.key

-----END PRIVATE KEY-----

</key>

4. **Retrieve the certificate and key files:** To fill in the client configuration file, used the following commands to get the contents of the CA certificate, client certificate, and private key (bash):
   - sudo cat /etc/openvpn/ca.crt
   - sudo cat /etc/openvpn/ubuntu.crt
   - sudo cat /home/ubuntu/openvpn-ca/pki/private/ubuntu.key
5. **Copy the client configuration file to the local machine:** Using SCP, transferred the generated ubuntu.ovpn file to the local machine for use by the OpenVPN client. (bash)
   - scp ubuntu@192.168.1.100:/etc/openvpn/ubuntu.ovpn /path/to/local/machine/

Issues encountered during client configuration:

- Initially encountered issues with resolving the server IP when copying the ubuntu.ovpn file using SCP. The issue was related to incorrect paths or undefined server IPs in the commands. This was resolved by correcting the hostname or IP address and using the correct destination path.

Troubleshooting steps:

- Ensured the correct server IP was specified in the ubuntu.ovpn file.
- Used hostname -I to check the server's IP address for transferring files using SCP.
- Verified the content of the ubuntu.ovpn file by checking the certificates and keys using the cat command before transferring it to the local machine.

Once the client configuration file was ready, it was transferred to the local machine, and the VPN connection was tested using the OpenVPN client.

---

6. Testing the VPN Connection

The objective of this section was to test the functionality of the OpenVPN server by connecting to it from a client machine (in this case, the host Windows machine) using the generated ubuntu.ovpn client configuration file. The purpose was to ensure that the VPN connection works properly and that the client can securely access the internal network through the VPN.

Commands run during VPN connection testing:

1. **Transfer the client configuration file (ubuntu.ovpn) to the Windows machine:** Used SCP to copy the ubuntu.ovpn file to the local Windows machine. The OpenVPN client on the Windows machine would then use this configuration file to connect to the server. (bash)
    - scp ubuntu@192.168.1.100:/etc/openvpn/ubuntu.ovpn /path/to/local/machine/
2. **Install OpenVPN on Windows:** On the Windows machine, downloaded and installed the OpenVPN client.
    - Download OpenVPN for Windows
3. **Import the client configuration file (ubuntu.ovpn):** After installing the OpenVPN client, imported the ubuntu.ovpn file into the OpenVPN GUI.
4. **Connect to the OpenVPN server:** Using the OpenVPN GUI, connected to the server. The OpenVPN tray icon was right-clicked, and "Connect" was selected to initiate the connection.

Issues encountered during testing:

- **Initial connection failure:**
  The initial connection failed due to an issue with the configuration file. The client was unable to load the inline certificate file. This was due to improperly pasted certificate contents in the ubuntu.ovpn file. The issue was resolved by ensuring the certificate and key files were correctly copied and pasted into the configuration.
- **Cipher mismatch warning:**
  The OpenVPN client on Windows showed a deprecation warning regarding the cipher configuration (AES-256-CBC), suggesting that the data-ciphers should include AES-256-GCM or AES-128-GCM. This warning did not block the connection, but it was noted for potential future adjustments.

Future Considerations: To resolve the cipher mismatch warning and comply with OpenVPN 2.6+ standards, update the OpenVPN configuration to use more modern ciphers.

1. Open the OpenVPN server configuration file (bash):
    - sudo nano /etc/openvpn/server.conf
2. Add or modify the following lines (yaml):
    - cipher AES-256-GCM
    - data-ciphers AES-256-GCM:AES-128-GCM
        - The data-ciphers directive allows OpenVPN to negotiate a stronger cipher between the client and server.
3. Save and restart the OpenVPN server (bash):
    - sudo systemctl restart openvpn@server

This can be implemented to strengthen the security of the VPN connection and ensure compatibility with the latest standards in future OpenVPN versions.

Troubleshooting steps:

1. **Resolve certificate loading issue:**
    - Reviewed and corrected the contents of the ubuntu.ovpn file to ensure the certificates and keys were properly pasted.
    - Verified the certificates using (bash):
        - sudo cat /etc/openvpn/ca.crt
        - sudo cat /etc/openvpn/ubuntu.crt
        - sudo cat /home/ubuntu/openvpn-ca/pki/private/ubuntu.key
2. **Check client log for connection issues:**
    - Examined the OpenVPN client logs on the Windows machine for any errors or warnings. This helped identify the issue with the inline certificate and the cipher mismatch warning.
3. **Test connectivity with the server:**
    - After connecting to the VPN, checked if the client was successfully assigned an IP address from the 10.8.0.0/24 network using the following command on the Windows machine (bash):
        - ipconfig
4. **Ping the server's internal IP:**
    - Once the client was connected, tested connectivity by pinging both the server's local network IP (192.168.1.100) and the VPN interface IP (10.8.0.1) from the Windows machine (bash):
        - ping 192.168.1.100
        - ping 10.8.0.1

Conclusion:

- **Successful VPN connection:**
  After resolving the certificate loading issue and importing the corrected ubuntu.ovpn file, the VPN connection was established successfully. The client received an IP address from the VPN network, and connectivity with the server was confirmed through successful ping tests.
- **Warnings:**
  The deprecation warning regarding the cipher was noted, but it did not prevent the VPN from

functioning. In the future, the cipher settings can be updated to use the recommended ciphers for OpenVPN 2.6+.

---

7. Troubleshooting Summary

The objective of this section is to summarize the key issues encountered during the setup and configuration of the OpenVPN server and client, as well as the steps taken to resolve them. This provides a comprehensive overview of the troubleshooting process and serves as a reference for future debugging.

Key Issues and Resolutions:

1. **Issue 1: OpenVPN service failed to start on the server**

   - **Description:** After configuring the OpenVPN server, the service failed to start due to missing or incorrect certificate and key files.
   - **Resolution:** The issue was resolved by copying the correct certificate and key files into the /etc/openvpn/ directory (bash):
     - sudo cp /home/ubuntu/openvpn-ca/pki/ca.crt /etc/openvpn/
     - sudo cp /home/ubuntu/openvpn-ca/pki/issued/server.crt /etc/openvpn/
     - sudo cp /home/ubuntu/openvpn-ca/pki/private/server.key /etc/openvpn/

Additionally, checked the OpenVPN logs using journalctl to identify the specific cause of the failure (bash):

- sudo journalctl -xeu openvpn@server.service

2. **Issue 2: Firewall rules not configured properly, blocking VPN traffic**

   - **Description:** After enabling the firewall (UFW), found that VPN connections were being blocked because the necessary firewall rules were not in place.
   - **Resolution:** The issue was resolved by allowing the required ports and services through the firewall (bash):
     - sudo ufw allow 1194/udp
     - sudo ufw allow OpenSSH
     - sudo ufw enable
     - sudo ufw status

3. **Issue 3: Client failed to connect due to certificate errors**

   - **Description:** When attempting to connect to the VPN from the Windows machine, the client failed to load the inline certificate file, resulting in a connection error.

- **Resolution:** Resolved this by ensuring that the contents of the CA certificate, client certificate, and client key were correctly pasted into the ubuntu.ovpn file. The following commands were used to retrieve the certificate and key contents (bash):
    - sudo cat /etc/openvpn/ca.crt
    - sudo cat /etc/openvpn/ubuntu.crt
    - sudo cat /home/ubuntu/openvpn-ca/pki/private/ubuntu.key

Once corrected, the ubuntu.ovpn file was transferred to the Windows machine and successfully imported into the OpenVPN client.

4. **Issue 4: Cipher mismatch warning on the client**
    - **Description:** During the connection process, the OpenVPN client displayed a deprecation warning about the AES-256-CBC cipher. The warning indicated that the cipher was not included in the --data-ciphers setting, and it suggested using more secure ciphers like AES-256-GCM or AES-128-GCM.
    - **Resolution:** This warning did not block the connection, so no immediate action was required. However, it is recommended to update the OpenVPN configuration to use modern ciphers in the future (bash):
        - cipher AES-256-GCM
5. **Issue 5: IP forwarding not enabled on the server**
    - **Description:** VPN traffic was not being routed properly because IP forwarding was not enabled on the server.
    - **Resolution:** The issue was resolved by enabling IP forwarding in the /etc/sysctl.conf file and applying the changes (bash):
        - sudo nano /etc/sysctl.conf
            1. net.ipv4.ip_forward = 1
            2. sudo sysctl -p

General Troubleshooting Approach:

- **Log Review:** One of the most effective troubleshooting methods used was reviewing logs, both on the server and the client. On the server, used journalctl and systemctl to review errors related to OpenVPN startup. On the client, the OpenVPN GUI provided logs that highlighted certificate issues.

- **Network Testing:** After establishing the VPN connection, tested connectivity by using ping to verify that the client could reach the server and vice versa. This helped confirm that the VPN tunnel was functioning properly.
- **Firewall Adjustments:** Ensuring that the firewall was properly configured to allow VPN traffic was crucial. I tested the rules using sudo ufw status and made adjustments as necessary.

---

8. Conclusion

The objective of this project was to configure a fully functional OpenVPN server to enable secure remote access to the internal network. The setup involved several key tasks, including the installation of OpenVPN and Easy-RSA, the generation of cryptographic certificates, the configuration of firewall rules, troubleshooting issues related to service startup, and finally, testing the VPN connection.

Throughout the process, addressed several challenges such as service failures, firewall rule misconfigurations, and certificate-related errors, ensuring a smooth and secure connection for remote users.

Key accomplishments:

1. **Secure VPN Server Setup**: Successfully installed and configured OpenVPN, ensuring that remote clients can securely connect to the internal network.
2. **Cryptographic Security**: Generated certificates and keys using Easy-RSA, ensuring secure communication between the VPN server and clients.
3. **Firewall Configuration**: Properly configured UFW to allow VPN traffic while maintaining security through OpenSSH access.
4. **Client Connection Testing**: Verified that the VPN client (running on a Windows machine) could successfully connect to the server and access the internal network.
5. **Troubleshooting and Resolution**: Identified and resolved issues related to OpenVPN service failures, client connection errors, and firewall misconfigurations.

By the end of the project, a secure and reliable OpenVPN server was established that allows remote clients to connect and access internal network resources. This VPN server is now fully functional and prepared for further use in the home lab environment, including integration with other services such as Graylog and Apache for remote logging and web access.

This setup provides a solid foundation for securing future remote connections and expanding the capabilities of the home lab for further cybersecurity testing and development.

Home Lab - OpenVPN Troubleshooting, Snapshots, and Backup Strategies

Primary Objective:

To outline the troubleshooting process undertaken to resolve connectivity issues that arose during the OpenVPN setup, caused by changes to the networking configuration. It details the steps taken to restore both OpenVPN and SSH functionality, while emphasizing the critical importance of system backups and recovery strategies to maintain system integrity and prevent future disruptions. This documentation serves as both a technical guide and a reflection on best practices for managing configuration changes and system recoverability.

Technical Context - Why Switch to Bridged Networking?

During my setup, the motivation behind switching to bridged networking (using dev tap0) was to simulate a scenario where VPN clients would appear as if they were part of the same local network as the server. This setup would allow VPN clients to access resources on the network as if they were physically connected, such as shared drives or printers. Bridged mode is ideal when:

- I need the VPN client to receive an IP address from the same subnet as the server (e.g., both on 192.168.1.100).
- I want full Layer 2 connectivity, allowing broadcast and multicast traffic to pass through, which might be required for certain services.

However, as I discovered, bridged networking adds more complexity and is often unnecessary unless these specific conditions are required. After experiencing network instability, I reverted to routed networking (dev tun), which is simpler and sufficient for most VPN use cases.

Changes Made to OpenVPN's Configuration:

1. **Switch from dev tun to dev tap0:**
    - Originally, OpenVPN was configured in **routed mode** using dev tun, which allows for IP routing within the VPN.
    - In an attempt to set up **bridged networking** for the VPN, the configuration was modified to use **dev tap0**, which is used for bridged mode (layer 2 tunneling). This change was intended to replicate a scenario where the VPN clients would appear as if they were on the same local network as the server.
2. **Specific Changes to server.conf:**
    - The dev tun line was replaced with dev tap0.

- The IP address configuration was changed from a routed IP block (10.8.0.0/24) to the bridge configuration (server-bridge 192.168.1.100 255.255.255.0 192.168.1.200 192.168.1.250). (bash)

    1. dev tap0
    2. server-bridge 192.168.1.100 255.255.255.0 192.168.1.200 192.168.1.250

3. **Effect on Network and VPN Connectivity:**

    1. **Network Conflict**:

        - The introduction of the network bridge (br0) in combination with tap0 led to conflicts with the existing network interface (enp0s8), which caused connectivity issues. Specifically, the bridge interface (br0) was inconsistently assigned IP addresses, leading to frequent changes in the server's IP.

    2. **Loss of VPN Functionality**:

        - The VPN clients were no longer able to connect due to the instability in IP assignments on the bridge interface. The connection logs showed errors related to TLS key negotiation and handshake failures, indicating a breakdown in the VPN connection process.

    3. **Loss of SSH Connectivity**:

        - The IP address changes also affected SSH access to the server. Since the bridge IP (br0) kept fluctuating, SSH connections were dropped, and the server became unreachable over the network.

Changes Made to the Netplan Configuration to Introduce the Network Bridge (br0):

To implement bridged networking for OpenVPN, I edited the **Netplan configuration** to introduce the bridge interface br0 and associate it with the physical interface enp0s8 and the virtual interface tap0.

1. **Original Netplan Configuration:** Before introducing the bridge, the configuration for the primary interface enp0s8 was set to obtain an IP address using DHCP (yaml):

```
network:
  version: 2
  renderer: networkd
  ethernets:
    enp0s8:
      dhcp4: yes
```

2. **Modified Netplan Configuration (for Bridged Networking):** To create the bridge and include the virtual interface for the VPN, the Netplan file was modified as follows (yaml):

```
network:
 version: 2
 renderer: networkd
 ethernets:
  enp0s8:
   dhcp4: no
 bridges:
  br0:
   interfaces: [enp0s8, tap0]
   dhcp4: yes
```

- **enp0s8**: The physical network interface was no longer assigned an IP directly; instead, it was attached to the bridge.
- **tap0**: This virtual interface was added to the bridge to enable VPN clients to be part of the same network as the server.
- **br0**: The bridge interface was configured to use DHCP, allowing it to receive an IP address dynamically.

Effect of These Changes:

- The server began experiencing **frequent IP changes** on the bridge interface br0, which caused issues with both VPN and SSH connectivity.
- The bridge would get an IP address, but this was inconsistent, and OpenVPN's bridged mode (tap0) added complexity, further destabilizing the network.

Reverting the Changes to Restore Routed Networking:

After identifying the instability caused by the bridge, the decision was made to revert the Netplan configuration back to the original state, removing the bridge and returning to **routed mode** using dev tun in OpenVPN.

1. **Reverted Netplan Configuration:** The bridge (br0) was removed, and the interface enp0s8 was restored to use DHCP directly (yaml):

```
network:
 version: 2
 renderer: networkd
```

```
ethernets:
  enp0s8:
    dhcp4: yes
```

2. **Applying the Changes:** After editing the Netplan configuration, the following command was used to apply the changes (bash):
    - sudo netplan apply
3. **Manually Removing the Bridge:** Although the bridge was removed from the Netplan configuration, it persisted in the system, to address this, manually removed the bridge with the following commands (bash):
    - sudo ip link set br0 down
    - sudo brctl delbr br0

These commands effectively removed the bridge interface from the system, restoring proper network connectivity to enp0s8.

Firewall Settings (Before and After the Changes):

Before and after the changes, worked with both **UFW (Uncomplicated Firewall)** and **iptables** to ensure proper routing and access for OpenVPN and SSH. Here's a breakdown of the firewall settings:

Before the Changes:

1. **UFW Settings (Prior to Bridged Networking)**:
    - OpenVPN (UDP on port 1194) and OpenSSH (TCP on port 22) were allowed through the firewall. (bash)
    - sudo ufw allow 1194/udp
    - sudo ufw allow OpenSSH

UFW Status Output (bash):

- sudo ufw status

```
1194/udp        ALLOW     Anywhere
OpenSSH         ALLOW     Anywhere
```

2. **iptables Settings**:
    - The relevant iptables rules were configured to accept OpenVPN traffic on port 1194 and SSH on port 22. Before the switch to bridged networking, iptables was correctly handling traffic for these services. (bash)
        - sudo iptables -A INPUT -p udp --dport 1194 -j ACCEPT

- sudo iptables -A INPUT -p tcp --dport 22 -j ACCEPT

After the Changes (Post-Reversion to Routed Mode):

1. **UFW After Reverting Back to dev tun**:
   - Once the network setup was reverted back to routed mode (dev tun), verified that UFW was still allowing OpenVPN and SSH traffic. UFW rules were not significantly changed during troubleshooting, confirmed that they were correct after reverting the changes.

UFW Allow Rules for OpenVPN and SSH (bash):

- sudo ufw allow 1194/udp
- sudo ufw allow OpenSSH

UFW Status Output (After Reversion) (bash):

- sudo ufw status

This confirmed that both OpenVPN and SSH traffic were allowed as expected.

2. **iptables After Reverting**:
   - After removing the bridge and reverting to routed mode, the iptables rules were verified to ensure that OpenVPN and SSH traffic was still being handled correctly. No major changes were required here, but the rules were confirmed to be in place.

Verifying iptables Rules (bash):

- sudo iptables -L

Rules Verified:

- UDP traffic on port 1194 for OpenVPN.
- TCP traffic on port 22 for SSH.

OpenVPN Service Configuration (Before and After the Changes):

1. **Before Changes (When Switching to dev tap0)**:
   - In the initial configuration switch to **bridged mode** with tap0, I updated the OpenVPN configuration file /etc/openvpn/server.conf to reflect the new bridge interface and network settings.

OpenVPN Config (Before) (bash):

- dev tap0
- server-bridge 192.168.1.100 255.255.255.0 192.168.1.200 192.168.1.250

This bridged configuration caused conflicts with the network, leading to the loss of connectivity.

2. **After Reversion to Routed Mode (dev tun)**:

    - To restore functionality, reverted back to the original **routed mode (dev tun)** configuration for OpenVPN.

OpenVPN Config (After Reversion) (bash):

- dev tun
- server 10.8.0.0 255.255.255.0

- The network setup was simplified, and the **server 10.8.0.0** block was restored to handle routing between the VPN clients and the server.

Restarting OpenVPN: After making the changes, restarted the OpenVPN service to apply the new configuration (bash):

- sudo systemctl restart openvpn

Verifying OpenVPN Service:

- Checked that OpenVPN was listening on the correct port and functioning as expected (bash):
    - sudo netstat -tuln | grep 1194

Steps Taken to Test SSH and OpenVPN Connectivity:

After reverting the OpenVPN and network configuration back to routed mode (dev tun) and ensuring the firewall rules were correct, performed several steps to verify that SSH and OpenVPN were working as expected.

1. SSH Connectivity Testing:

Once the IP address for the primary interface (enp0s8) was stable and assigned correctly (after removing the bridge), tested SSH connectivity from the host machine to the server.

Steps to Test SSH:

- **Ping the Server**: Started by pinging the server's IP address (192.168.1.100) to ensure it was reachable over the network. (bash)
    - ping 192.168.1.100

Result: Successful response from the server with no packet loss, confirming that the server was reachable on the network.

- **SSH into the Server**: Once the server responded to pings, used SSH to log into the server from the host machine (bash):

- ssh -i ~/.ssh/id_rsa ubuntu@192.168.1.100

Result: The SSH login prompt appeared, and successfully authenticated and accessed the server. This confirmed that SSH connectivity was restored.

---

2. OpenVPN Connectivity Testing:

After ensuring that SSH was functional, proceeded to test the OpenVPN connection from the host machine to the VPN server.

Steps to Test OpenVPN Connectivity:

- **Connect to OpenVPN**: Using the OpenVPN client on the host machine, initiated a connection to the VPN server. This involved running the OpenVPN client configuration file (bash):
  - sudo openvpn --config /path/to/client.ovpn

Result: Successful connection to the VPN server, with the log showing that the client successfully negotiated a TLS handshake and established a connection.

Common Logs for a Successful Connection:

Initialization Sequence Completed

- **Check IP Address**: Once connected, verified that the VPN client received an IP address from the **10.8.0.0/24** range (as configured in server.conf) (bash):
  - ifconfig

Result: The VPN client returned the IP address 10.8.0.6, confirming that the OpenVPN server was assigning IPs correctly.

- **Check Routing**: Also checked if traffic was being routed through the VPN (bash):
  - curl ifconfig.me

Result: The external IP address was 100.34.236.235, the public IP of the VPN server (the home network), indicating that the client's traffic was correctly routed through the VPN.

---

3. Verifying Logs and Network Settings:

To further identify and resolve issues, examined various logs and network settings to ensure that both OpenVPN and SSH were working properly.

- **Check OpenVPN Logs**: Used journalctl to inspect the OpenVPN logs and look for any errors during the connection process (bash):
  - sudo journalctl -u openvpn

During troubleshooting, identified **TLS key negotiation errors** and **TLS handshake failures**. These issues occurred when the network bridge was active and the OpenVPN configuration was unstable.

After reverting to dev tun, these errors were no longer present in the logs.

- **Check Network Interface Configuration**: Used the ifconfig command to ensure that the network interfaces (enp0s8 and tun0) were correctly assigned IP addresses.

Result: enp0s8 has the local IP, 192.168.1.100, and tun0 (the OpenVPN interface) has the IP 10.8.0.1/24.

- **Command (bash)**:
  - ifconfig
- **Check OpenVPN Service Status**: To ensure that OpenVPN was actively running, checked its status using systemctl (bash):
  - sudo systemctl status openvpn

Result: The service was listed as active (running) with no errors in the output.

Cause of the Dynamic IP Address Changes on the Bridge Interface (br0):

The dynamic IP address changes on the bridge interface were caused by the way the **bridge (br0)** was set up in the Netplan configuration. The bridge was configured to use **DHCP** to obtain an IP address automatically. Here's a breakdown of what happened:

1. **Bridge (br0) with DHCP Enabled**:
   - The Netplan configuration for the bridge was set as follows (yaml):

```
bridges:
 br0:
  interfaces: [enp0s8, tap0]
  dhcp4: yes
```

This meant that the bridge interface (br0) would attempt to dynamically acquire an IP address from the DHCP server on the network. However, because it was tied to both a physical interface (enp0s8) and a virtual interface (tap0), the IP address assignment became unstable.

2. **Network Conflict Between Interfaces**:
   - The physical interface (enp0s8) and the bridge (br0) were both competing for an IP address, causing conflicting requests to the DHCP server. This resulted in frequent **IP address changes** for the bridge, which disrupted connectivity.
   - The bridge would acquire an IP address, but it would change frequently as both the bridge and the underlying interface (enp0s8) tried to obtain their own IPs, creating instability.

Index

Resolution to the Dynamic IP Issue:

To resolve this, it was necessary to simplify the network configuration by **removing the bridge** and reverting back to a stable, static setup using **routed networking (dev tun)**. Here's how this was handled:

1. **Revert Netplan Configuration**:

   - Removed the bridge entirely and reverted the network settings to assign a static IP to the physical interface (enp0s8) using DHCP, without involving the bridge or virtual interfaces.

Reverted Netplan Configuration (yaml):

```
network:
  version: 2
  renderer: networkd
  ethernets:
    enp0s8:
      dhcp4: yes
```

2. **Manual Removal of the Bridge**:

   - Since simply editing the Netplan file wouldn't automatically remove the bridge, manually brought down and deleted the bridge interface (br0).
   - **Commands Used (bash)**:
     - sudo ip link set br0 down
     - sudo brctl delbr br0

These commands disabled and removed the bridge interface, resolving the dynamic IP conflicts caused by the bridge.

3. **Verify Network Stability**: After removing the bridge, verified that the server was now receiving a stable IP address on the enp0s8 interface via DHCP.

Command to Check IP Assignment (bash):

- ifconfig

Result: enp0s8 had a stable IP address, 192.168.1.100, assigned via DHCP.

Summary:

The dynamic IP address changes were caused by the use of DHCP on the bridge interface (br0), which conflicted with the underlying physical interface. By removing the bridge and reverting to a simplified network configuration, network stability was restored.

Final Changes to Restore Full OpenVPN Functionality:

After identifying and resolving the core network issues, several final changes were made to ensure OpenVPN was fully operational and stable. Here's a breakdown of the final steps:

1. Reverting OpenVPN to Routed Mode (dev tun):

The OpenVPN configuration was reverted from **bridged mode** (dev tap0) back to **routed mode** (dev tun). This was done to restore the VPN's original functionality and avoid the instability caused by the bridged network.

Final OpenVPN Configuration (bash):

- sudo nano /etc/openvpn/server.conf
  - The dev tap0 line was reverted back to dev tun (bash):
    - dev tun
  - The bridged network IP settings (server-bridge) was replaced with the routed network configuration (bash):
    - server 10.8.0.0 255.255.255.0
  - Additional settings such as **persist-key**, **persist-tun**, and the **DHCP options** for DNS were retained to ensure proper VPN client behavior (bash):
    - push "dhcp-option DNS 8.8.8.8"
    - push "dhcp-option DNS 8.8.4.4"
  - **Restarted the OpenVPN service** to apply the changes (bash):
    - sudo systemctl restart openvpn

---

2. Confirming Network Interface Stability:

- After removing the network bridge (br0) and restoring the enp0s8 interface to DHCP, confirmed that the server's network interface was now stable and assigned a consistent IP address (e.g., 192.168.1.100).

Command to Verify Interface Stability (bash):

- ifconfig

Result: enp0s8 now has a stable IP address, and no dynamic changes should occur. The VPN interface (tun0) has an IP address of **10.8.0.1/24**.

---

3. Firewall and Routing Configuration:

- The firewall settings were verified to ensure that traffic for OpenVPN (UDP port 1194) and SSH (TCP port 22) was correctly allowed through the system firewall.

Firewall Check (bash):

- sudo ufw status

Result: UDP traffic on port 1194 (OpenVPN) and TCP traffic on port 22 (SSH) is allowed.

---

4. Testing and Validating OpenVPN Connectivity:

Once the configuration changes were made, validated that OpenVPN was working correctly by running the following tests:

- **OpenVPN Client Connection**:
    - Initiated an OpenVPN connection from the host machine using the OpenVPN client configuration file.
        - sudo openvpn --config /path/to/client.ovpn

Result: The client successfully connected to the OpenVPN server, with log messages confirming a successful TLS handshake and connection.

- **Check Assigned VPN IP**:
    - After the client connected to the server, checked the IP address assigned to the VPN client.
- **Command to Check IP (bash)**:
    - ifconfig

Result: The VPN client has an IP address, **10.8.0.6**, indicating that it is properly routed through the VPN.

- **Verify Routing through the VPN**:
    - Ran a check to ensure the VPN was routing traffic correctly. By querying an external service, confirmed that the traffic was being routed through the VPN.
- **Command to Check External IP**:
    - curl ifconfig.me

Result: The output showed the public IP address of the VPN server, indicating that traffic is being routed through the VPN connection.

---

5. Confirming SSH Access:

- After confirming that OpenVPN was working, verified that **SSH access** to the server was restored by logging in from the host machine.
- **SSH Command (bash)**:
  - ssh -i ~/.ssh/id_rsa ubuntu@192.168.1.100

Result: Successful SSH login, confirming that the network and VPN were stable and properly configured.

6. Snapshot Created:

Once the system was confirmed to be working, a **snapshot** of the current configuration was created to preserve the state of the system for future recovery if needed.

Learning Moment:

After completing the troubleshooting and successfully restoring both OpenVPN and SSH services, this experience highlighted the **importance of system backups** and having a solid recovery strategy in place. Here's how it reinforced key lessons:

1. Importance of System Backups:

During the troubleshooting process, changes to the network configuration (like introducing a bridge and switching OpenVPN to bridged mode) caused unexpected issues. If I had not been able to recover the system configuration through snapshots, this could have resulted in a more time-consuming and complex recovery process. Here's what I learned:

- **Prevention**: Taking snapshots of critical system configurations before making major changes is essential to ensure quick recovery in case of failure.
- **Recovery Efficiency**: With a snapshot in place, I would have been able to quickly revert to a known working state and test out fixes without fear of losing progress. This would have saved time.
- **Long-term Sustainability**: System snapshots and backups allow for a more sustainable system management process. In future projects, I plan to incorporate regular backup routines to ensure that I can restore functionality quickly in the event of unexpected changes or failures.

2. Bridging the Gap Between Learning and Practice:

This scenario also provided a **practical learning experience** in how the network and VPN configuration changes can have wide-reaching effects. The main takeaways were:

- **Hands-On Troubleshooting**: The hands-on troubleshooting experience helped me gain deeper insights into network configuration management, particularly the differences between routed (dev tun) and bridged (dev tap0) VPN setups.

- **Complexity of Bridged Networks**: The difficulties encountered while trying to use bridged networking (tap0), especially with fluctuating IP addresses and network instability, highlighted that more complex network setups require careful planning which may not always be necessary in simpler environments.

3. Future Use of Snapshots and Backups:

- From this experience, I learned that **having a snapshot or backup** before making significant changes is critical, especially when working with complex systems like OpenVPN and network configurations.
- Moving forward, I will always make sure to create snapshots at key points in my projects, particularly before introducing major changes. This way, I can easily revert if things go wrong and continue experimenting without worrying about losing my progress.

Conclusion:

Through this experience, I learned the importance of taking snapshots and creating backups **before** making major changes. Since I hadn’t taken a snapshot prior to making the network and VPN configuration changes, I had to manually revert to a working state. This process reinforced the value of snapshots for:

- **Saving Time**: Having a snapshot in place would have allowed me to revert quickly, instead of spending time manually undoing the changes.
- **Preventing Disruption**: Without a snapshot, there was a higher risk of prolonged downtime and potential configuration errors during the manual recovery process.

Beyond backups, I also learned valuable lessons about network management and system configuration. Switching to bridged mode (dev tap0) introduced unnecessary complexity and caused network instability. Reverting to routed mode (dev tun) simplified the setup and restored stable connectivity. The experience emphasized that proper network configuration is critical for consistent IP assignments and overall system stability.

In the end, this situation taught me that backups and recovery strategies are not just theoretical practices, but essential for maintaining system integrity. By successfully restoring both OpenVPN and SSH functionality, I gained a clearer understanding of how to troubleshoot, revert changes, and preserve system states. These lessons will be invaluable in future projects, allowing me to manage configurations more efficiently and reduce downtime in the event of disruptions.

Future Steps – Expanding IT Infrastructure Knowledge and SOC Development

---

Overview

This document serves as a roadmap for the next phase of my learning and development in IT infrastructure, server management, automation, and security. Building upon the foundation established in my home lab project, the goal is to further refine existing skills, explore advanced configurations, and set up new systems to simulate real-world IT and cybersecurity environments. Each section outlines a specific area of focus, tools to be used, and clear milestones for success, prioritized in a logical sequence for growth.

---

1. Advancing Knowledge of Current Tools

Objective: Gain a deeper, more advanced understanding of the tools already deployed in my home lab, focusing on their more intricate and powerful features. This section involves maximizing the potential of **Wireshark**, **Nmap**, **Graylog**, **Splunk**, and **OpenVPN**, while refining configurations for better performance and more secure implementations.

Tools:

- **Wireshark/Tshark**:
    - **Objective**: Use advanced filters, deep packet inspection, and network flow analysis to monitor and troubleshoot more complex protocols.
    - **Learning Focus**: Advanced packet analysis, custom filters, troubleshooting real-time traffic issues, using capture and replay for incident analysis.
    - **Milestones**:
        - Implement filters for complex protocols.
        - Analyze SSH tunneling traffic and detect anomalies.
        - Simulate attacks and study traffic signatures.
- **Nmap**:
    - **Objective**: Explore advanced scanning techniques, including stealth scans, firewall evasion, and version detection.
    - **Learning Focus**: OS detection, vulnerability detection, advanced network scanning techniques.
    - **Milestones**:
        - Perform OS detection scans.

        - Configure vulnerability detection scripts.
        - Conduct stealth scans to test firewall evasion.

- **Graylog and Splunk**:
    - **Objective**: Enhance log processing and visualization for security monitoring.
    - **Learning Focus**: Set up more advanced pipelines in **Graylog** to automate log parsing and create **Splunk** dashboards for real-time event correlation.
    - **Milestones**:
        - Create custom pipelines for log parsing and threat detection.
        - Build custom **Splunk** dashboards for visualizing key security metrics.
        - Integrate alerts for abnormal traffic patterns and incidents.
- **OpenVPN**:
    - **Objective**: Strengthen security by improving the configuration of encryption settings and multi-client management.
    - **Learning Focus**: Implement stronger encryption algorithms, manage VPN user permissions, and enforce multi-factor authentication (MFA).
    - **Milestones**:
        - Configure advanced encryption settings (e.g., AES-256).
        - Set up MFA for OpenVPN clients.
        - Create detailed logging of VPN activity.

---

2. Security Hardening and Auditing

Objective: Strengthen server security by hardening configurations and conducting regular audits using tools like **Lynis**.

Tools:

- **Lynis**, **UFW**, **iptables**, **fail2ban**:
    - **Learning Focus**: Implement best practices for server security, including firewalls, SSH hardening, and intrusion detection.
    - **Milestones**:
        - Harden server configurations (e.g., SSH, firewall settings).
        - Conduct security audits with **Lynis**.
        - Implement **fail2ban** to prevent brute-force attacks.

---

3. Server Monitoring and Performance Tuning

Objective: Monitor server health and performance using **Prometheus** or **Nagios** to track metrics such as CPU usage, memory, and network traffic, and implement performance optimization.

Tools:

- **Prometheus**, **Nagios**:
  - **Learning Focus**: Implement server monitoring and track system metrics in real-time, with a focus on identifying and resolving performance bottlenecks.
  - **Milestones**:
    - Configure monitoring dashboards for server health.
    - Set alerts for critical server resource usage.
    - Optimize server configurations based on performance metrics.

---

4. Automating Configuration with Ansible/Puppet/Chef

Objective: Automate server configurations and updates across multiple machines using tools like **Ansible**, **Puppet**, or **Chef**.

Tools:

- **Ansible**, **Puppet**, **Chef**:
  - **Learning Focus**: Use configuration management tools to automate repetitive tasks, ensuring consistency across server environments.
  - **Milestones**:
    - Automate server setup and configuration.
    - Create reusable playbooks and manifests for provisioning.
    - Manage updates and configurations across multiple servers.

---

5. Advanced Web Server Management

Objective: Set up a cluster of web servers with **HAProxy** or **Nginx** load balancing to efficiently manage traffic distribution and ensure high availability.

Tools:

- **HAProxy** or **Nginx**:
  - **Learning Focus**: Understand how load balancers distribute traffic across multiple web servers, ensuring fault tolerance and seamless failover in case of server failure.

- **Milestones**:
    - Set up and configure **HAProxy** or **Nginx** as a load balancer.
    - Implement traffic distribution to multiple servers.
    - Simulate server failover and test the high-availability configuration.

---

6. Hypervisor Management and Virtualization

Objective: Deepen my understanding of virtualization by using **Proxmox** or **VMware ESXi** to create and manage a fully virtualized infrastructure, including managing virtual networks and snapshots.

Tools:

- **Proxmox** or **VMware ESXi**:
    - **Learning Focus**: Explore advanced virtual machine management, including resource allocation, virtual networking, and system snapshots.
    - **Milestones**:
        - Set up and manage multiple VMs using **Proxmox** or **ESXi**.
        - Create and manage virtual networks to simulate different environments.
        - Implement snapshot management for VM state rollback and recovery.

---

7. Service Management Beyond Web Hosting

Objective: Expand service management by deploying and managing advanced Linux services like **Mail** (Postfix), **DNS** (BIND), and **FTP** (vsftpd).

Tools:

- **Postfix**, **BIND**, **vsftpd**:
    - **Learning Focus**: Learn to configure and secure essential services beyond web hosting.
    - **Milestones**:
        - Set up and secure a mail server using **Postfix**.
        - Deploy a **DNS** server using **BIND**.
        - Manage file transfers with **vsftpd**, ensuring secure access and FTP logging.

---

8. Backup Strategies and Disaster Recovery

Objective: Implement automated, incremental backup strategies using **rsync** or **Bacula**, ensuring secure data backups and disaster recovery procedures.

Tools:

- **rsync**, **Bacula**, **cron jobs**:
    - **Learning Focus**: Automate data backup and implement disaster recovery strategies to protect data integrity.
    - **Milestones**:
        - Automate incremental backups using **rsync** and schedule them with **cron**.
        - Set up enterprise-level backups with **Bacula**.
        - Test recovery procedures to ensure data integrity.

---

9. Building a Security Operations Center (SOC)

Objective: Set up a Security Operations Center (SOC) using tools like **Graylog**, **Splunk**, **Suricata**, and **Snort** to centralize monitoring, threat detection, and incident response.

Tools:

- **Graylog**, **Splunk**, **Suricata**, **Snort**:
    - **Learning Focus**: Implement a full SOC environment with real-time monitoring, intrusion detection, and automated incident response workflows.
    - **Milestones**:
        - Configure centralized logging for security monitoring.
        - Set up **Suricata** or **Snort** for intrusion detection.
        - Implement playbooks for incident response automation.

---

10. Cloud Integration and Hybrid Networks

Objective: Gain hands-on experience with cloud technologies by integrating services like **AWS** or **Azure** to simulate hybrid cloud environments and explore cloud security.

Tools:

- **AWS**, **Azure**:
    - **Learning Focus**: Learn cloud-based server management, storage, networking, and security best practices.
    - **Milestones**:
        - Set up virtual networks and instances in **AWS** or **Azure**.

- Explore **AWS IAM** and **Azure Active Directory** for cloud security management.
- Integrate on-premise and cloud services for hybrid network simulation.

Home Lab – Final Thoughts

This home lab project successfully demonstrates the ability to design, deploy, and manage a complex IT infrastructure while focusing on cybersecurity, networking, and system administration. Over the course of this project, several key skills were developed and honed, each contributing to the overall success of the lab:

- **Infrastructure and Virtualization Mastery**: By partitioning a 1TB drive into three dedicated partitions (E, F, and G) and configuring **VirtualBox** (version 7.0.20) to manage multiple virtual machines, the project highlighted the importance of resource management and scalability. This allowed the virtual machines to run services, security tools, and monitoring systems in isolated environments, simulating enterprise-level IT environments with clear segregation between different tools and services.
- **Networking Expertise**: The setup and configuration of multiple networking modes (**Host-Only**, **NAT**, and **Bridged Networking**) showcased the ability to implement and understand various network architectures. Troubleshooting Bridged Networking, particularly ensuring the correct Wi-Fi interface (**wlp3s0**) was used, was crucial to enabling external access to services like **Graylog**. These hands-on experiences allowed for testing network isolation, internal communication, and secure external access, essential for managing both local and wide-area networks in real-world environments.
- **Security Tools Deployment**: The successful installation and configuration of security tools like **Wireshark**, **Nmap**, and **Metasploit** demonstrated proficiency in monitoring and securing networks. Using **Wireshark** and **Tshark** to analyze protocols like **mDNS**, **SSDP**, and **SSH traffic** provided deep insights into network behavior, while **Metasploit** was used to simulate real-world vulnerability assessments, including testing and exploiting weak points in the network infrastructure.
- **Log Management and SIEM Setup**: Configuring **Graylog** and **Splunk** for centralized log aggregation and monitoring was a critical aspect of this project. Logs from multiple sources, including SSH login attempts and network protocols, were aggregated and analyzed in real-time to detect potential security incidents. **Splunk's** powerful dashboarding and **Graylog's** real-time log processing created a robust foundation for advanced Security Information and Event Management (SIEM) capabilities. The ability to visualize, filter, and analyze logs provided invaluable insights into system events and potential threats.
- **Database and Web Service Management**: The deployment of **PostgreSQL** as a primary database engine for storing large volumes of log data from **Splunk** reinforced the importance of backend data management. Additionally, the setup of **Apache** as a web server to host applications and services demonstrated competence in managing key web infrastructure components. These services were configured to ensure that data integrity and availability were prioritized, mimicking enterprise-level web service and database management.
- **Containerization and Service Deployment**: The installation and use of **Docker** for containerization allowed for the rapid deployment and isolation of services. Containers

were deployed to simulate real-world applications, enabling efficient testing and management. This skill is increasingly relevant in DevOps environments, where containerized applications are a cornerstone for scalable and reliable deployments.

- **Remote Access and VPN Security**: Setting up **OpenVPN** to provide secure remote access to the internal network ensured that external connections were safely encrypted, protecting sensitive data during transmission. The firewall configurations using **UFW** (Uncomplicated Firewall) ensured that both VPN and SSH traffic were securely allowed while minimizing exposure to potential external threats.

Key Learnings

Throughout the execution of this home lab project, several key insights and practical skills were developed:

- **Effective Resource Management**: Partitioning drives and allocating system resources for virtual machines taught the importance of planning and optimizing storage for scalable environments. This reinforced how to efficiently manage resources in both physical and virtual settings, ensuring that each service had the necessary CPU, RAM, and disk space.
- **Adapting to Networking Challenges**: Troubleshooting network configurations, especially **Bridged Networking** and ensuring correct adapter usage (such as **wlp3s0** for Wi-Fi), provided invaluable experience. Configuring and testing **Host-Only**, **NAT**, and **Bridged Networking** in **VirtualBox** showcased a deeper understanding of how different network architectures impact security, communication, and access across environments.
- **Security Best Practices**: Implementing tools like **Wireshark**, **Nmap**, and **Metasploit** helped deepen my understanding of network security monitoring and vulnerability management. Real-world testing through **Metasploit** simulated actual penetration testing processes, while using **Wireshark** to analyze SSH traffic and service discovery protocols reinforced the role of monitoring in defending against security threats.
- **Log Management and Data Aggregation**: Setting up centralized log management systems with **Graylog** and **Splunk** emphasized the importance of monitoring and analyzing logs for real-time threat detection and operational insights. I learned how to configure pipelines in **Graylog** to process logs efficiently and how to build custom dashboards in **Splunk** to visualize critical system and security data.
- **Service Deployment and Automation**: Learning to deploy services using **Docker** and configure web servers like **Apache** taught me the value of automation and containerization in modern IT environments. These tools streamline service deployment and testing in isolated environments, reducing the complexity of managing multiple services.
- **Secure Remote Access**: Setting up and securing **OpenVPN** connections provided practical experience in creating and managing secure remote access for real-world applications. Configuring firewall rules via **UFW** to allow secure traffic while preventing unauthorized access emphasized the importance of layered security when managing remote systems.

Index

Final Thoughts

This home lab project not only demonstrates technical proficiency across multiple domains but also showcases problem-solving, adaptability, and a structured approach to system and network management. From initial setup to the deployment of sophisticated tools, every aspect of the project contributed to a deeper understanding of how to configure, manage, and secure IT environments.

The combination of networking, security, database management, and virtualization provided a holistic view of IT infrastructure, positioning this project as an essential demonstration of skills needed in roles such as system administration, network engineering, and cybersecurity analysis. These practical skills, coupled with the troubleshooting and optimization encountered along the way, highlight readiness for complex IT challenges in professional settings.