Contents

PROJECT-1 BUILD A COMPLETE CYBER LAB ENVIRONMENT

CHAPTER 1: SET UP AND BIOS PROCESS

GOALS:

1.1 VM Setup and Hardware Understanding

1.2 BIOS/UEFI & Boot Process

1.1 VM SETUP AND HARDWARE UNDERSTANDING

- **Purpose**
  This self-paced project focuses on setting up virtual machines using VirtualBox or VMware,
  installing Windows 10/11, and understanding the roles of hardware components in virtualization. The goal is to practice creating and managing VMs for learning and experimentation

- **Setup**
  2.1 Requirements
  • Software:
  – VirtualBox (version 7.0 or later) or VMware Workstation Player (version 17 or later).
  – ISO file for Windows 10/11 (e.g., Win11_24H2_English_x64.iso).
  • System Requirements:
  – Host OS: Windows 10/11, macOS, or Linux.
  – Minimum 8GB RAM, 4-core CPU, 100GB free storage (SSD recommended).
  • Dependencies:
  – Virtualization enabled in BIOS (VT-x/AMD-V)

  2.2 Installation Steps

  1. Install Virtualization Software:

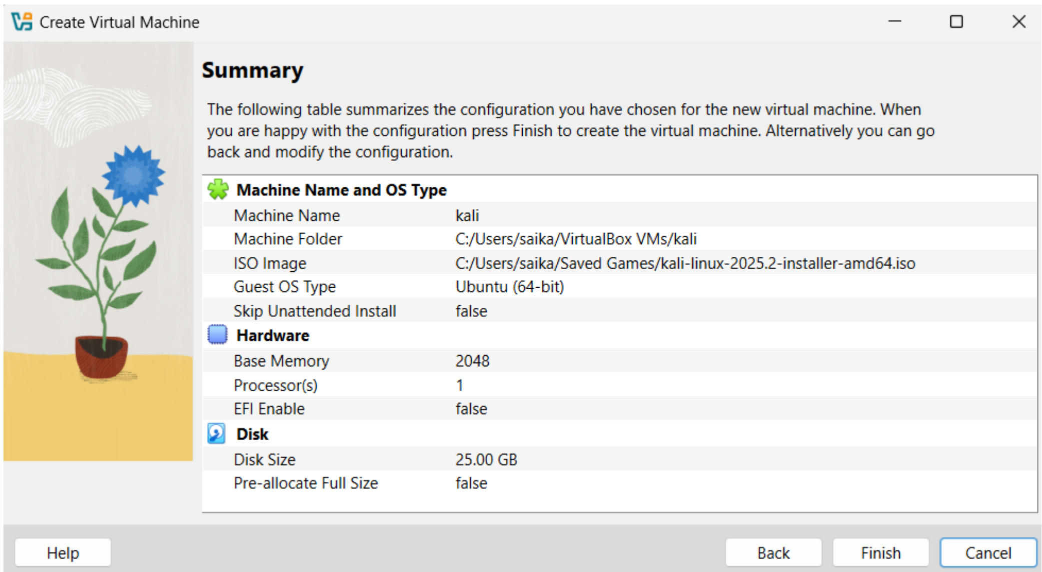
     • Download VirtualBox from https://www.virtualbox.org

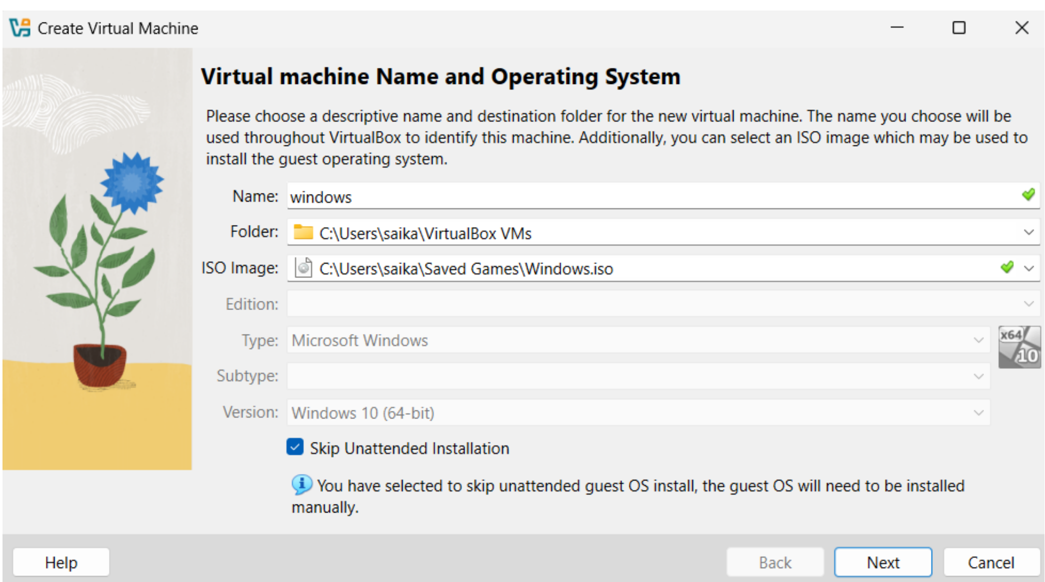
     • Follow installer prompts to complete setup.

  2. Create Virtual Machine:
     • Open VirtualBox and select “New VM.”
     • Configure VM specs:
       – Windows 10/11 VM: 2GB RAM, 2 CPU cores, 50GB storage (dynamic allocation).
     • Assign Windows ISO file.
       - Kali Linux VM: 2GB RAM, 2 CPU cores, 25GB storage (dynamic allocation).

• Assign Kali-linux-2025.2 ISO file.

3. Install Windows 10/11:
   • Boot VM, enter product key (or skip for trial), select “Custom Install.”
   • Follow prompts to partition disk and complete setup.
4. Verify Setup:
   • Ensure the VM boots correctly and network is configured (e.g., NAT for internet access)





3.Hardware Component Roles

• CPU:
  – Powers VM processing, such as running the OS and applications.

– More cores allow smoother performance, especially with multiple VMs.

• RAM:

– Provides memory for the VM; 2GB minimum ensures Windows runs efficiently.

– Insufficient RAM may cause lag or crashes.

• GPU:

– Handles graphical rendering for Windows desktop and applications.

– Enable 3D acceleration in VM settings for improved visuals.

• SSD/HDD:

– Stores VM disk images and OS files.

– SSDs offer faster boot times and responsiveness compared to HDDs.

Note:

• Challenges:

-VirtualBox was pretty new and made few mistakes while configuring my VM

-Wrong configuration lead to os crash

•lessons:

-I was able to learn configuration within the VM box to particular os

Future Improvements

• Create additional VMs with different OS versions (e.g., Windows Server).

• Experiment with bridged networking for VM-to-host communication.

• Test resource-intensive applications to study hardware limits.

• Hoping this Cyber lab Environment

1.2 BIOS/UEFI and Boot Process Documentation

1 Purpose

This self-paced project explores BIOS/UEFI settings and the boot process in virtual machines using VirtualBox or VMware. The objectives are to access and modify BIOS/UEFI settings, compare Legacy and UEFI boot modes, document the complete boot sequence from power-on to desktop, and practice Safe Mode and recovery boot options for learning purposes.

2 Accessing BIOS/UEFI Settings

1. VirtualBox:
   - Select the Windows VM and click “Settings” > “System” > “Motherboard.”
   - Enable “Enable EFI (special OSes only)” for UEFI mode; disable for Legacy BIOS.
   - Start the VM and press F2 repeatedly during boot to enter BIOS/UEFI setup

2.Modifying Boot Order and Comparing Legacy vs. UEFI

- Modifying Boot Order:
  – In BIOS/UEFI, navigate to the “Boot” menu.
  – Adjust boot priority (e.g., prioritize virtual hard disk over CD/DVD).
  – Save changes (usually F10) and reboot to test.
- Legacy vs. UEFI Comparison:
  – Legacy BIOS: Uses MBR partitioning, simpler interface, limited to 2TB disks, no Secure Boot.
  – UEFI: Supports GPT partitioning, graphical interface, larger disks, and Secure Boot for enhanced security.
  – Tested both modes in VM settings; UEFI required GPT disk setup during Windows installation.

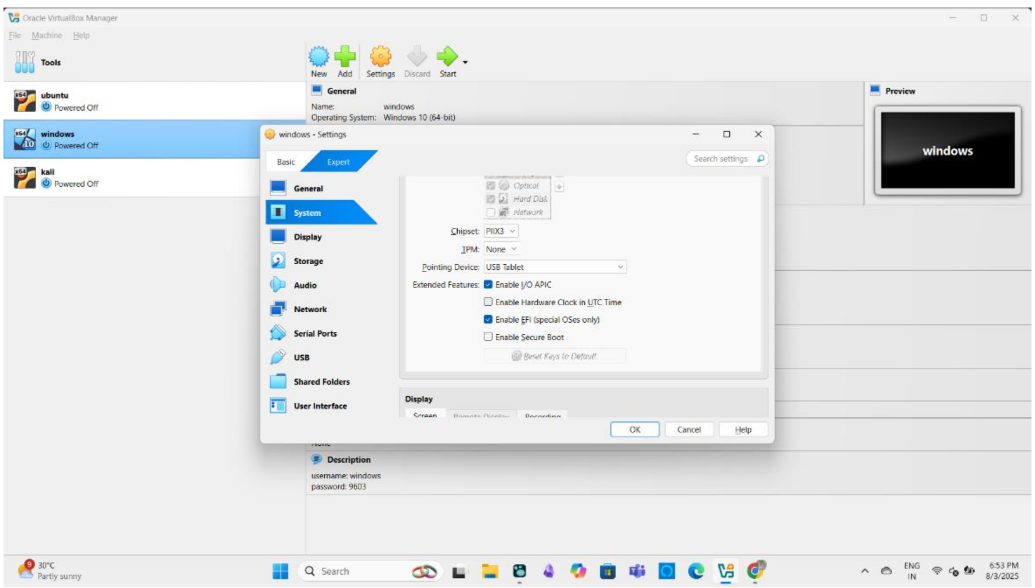
3 Boot Sequence

The complete boot sequence from power-on to desktop in a Windows 10/11 VM:

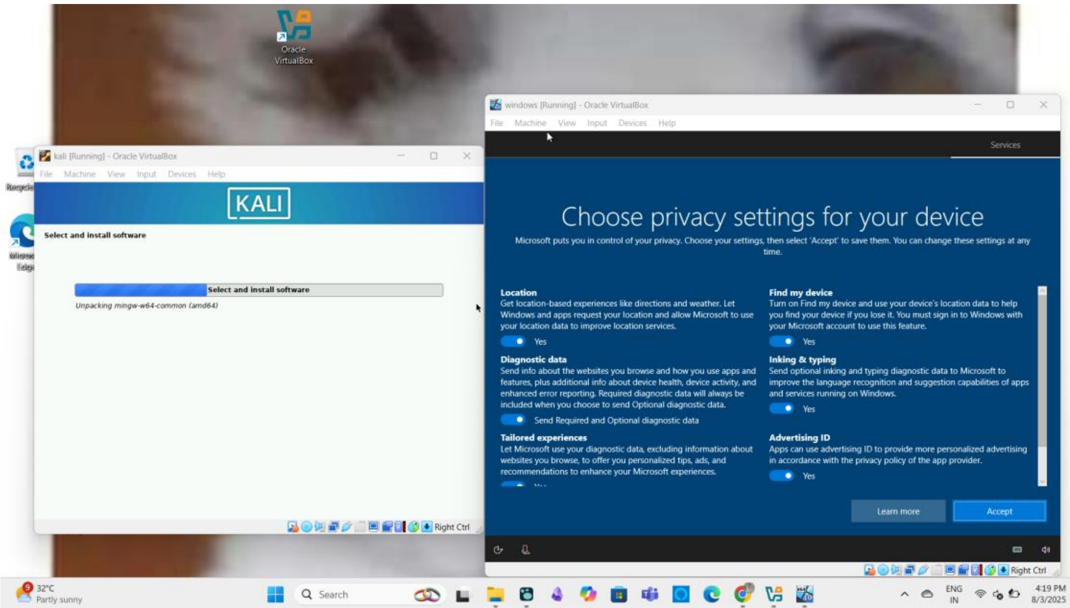
1. Power-On: VM initializes virtual hardware (CPU, RAM, etc.).
2. POST (Power-On Self-Test): BIOS/UEFI checks virtual hardware integrity.
3. Bootloader: Loads Windows Boot Manager (UEFI) or NTLDR (Legacy).
4. Kernel Loading: Windows kernel initializes, drivers load.
5. User Interface: Login screen appears, followed by desktop after user authentication.

Observation: UEFI boot is faster due to streamlined firmware; Legacy boot showed slight delay

PROJECT-1 BUILD A COMPLETE CYBER LAB ENVIRONMENT



PROJECT-1 BUILD A COMPLETE CYBER LAB ENVIRONMENT



4.Safe Mode and

Recovery Boot Options

1. Safe Mode:

- Restart VM, press F8 (or Shift + F8 in UEFI) during boot.
- Select "Safe Mode" from Advanced Boot Options.
- Result: Boots with minimal drivers; used to test system stability.

2. Recovery Options:

- Access via Settings > System > Recovery > "Restart now" under Advanced Startup.
- Alternatively, interrupt boot three times to trigger Automatic Repair.
- Options tested: System Restore, Startup Repair, Command Prompt.

Note:

- **Challenges:**
  - ⇨ Enabling EFI Without 2nd boot in VM requests for bootable OSI image even able proper boot
  - ⇨ Using precise timing with f2 for boot is kind of tricky
- **Lessons Learned:** UEFI's Secure Boot enhances security but requires compatible OS configuration

Future Improvements

• Test boot process with other OSes (e.g., Linux distributions).

• Explore Secure Boot configuration in UEFI.

• Simulate hardware failures to study recovery options in depth

---

CHAPTER 2: OPERATING SYSTEM DEEP DRIVE

2.1 OS Architecture Exploration

2.11 Compare GUI vs CLI interfaces

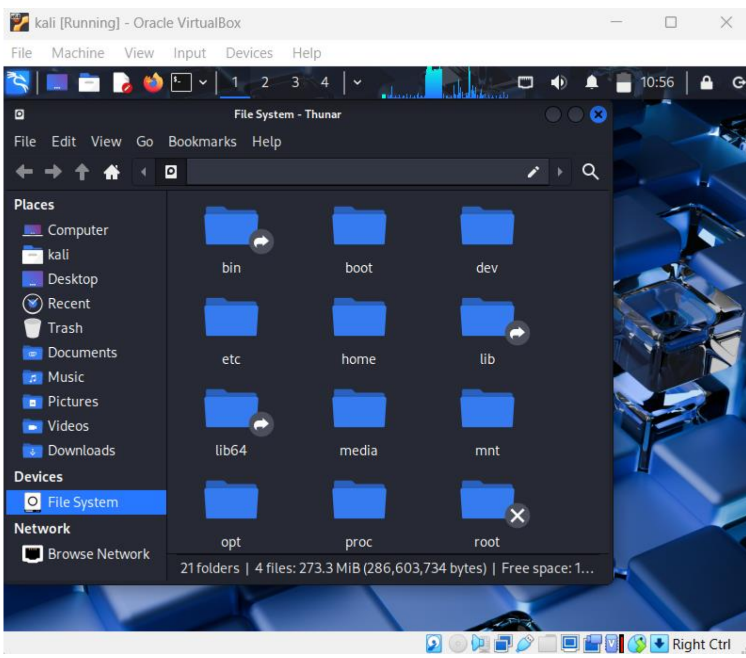
Feature	**GUI**	**CLI**
Ease of Use	Beginner-friendly and intuitive	Requires memorization and practice
Speed (Execution)	Slower for advanced tasks (requires clicks)	Faster for experienced users (one-liners)
Resources Used	High (needs CPU, RAM, GPU for graphics)	Low (minimal system resource usage)
Automation	Difficult (limited scripting support)	Excellent (supports scripting & automation)
Flexibility	Limited to what's shown in menus	Very flexible, can combine commands (piping, redirection)
Learning Curve	Shallow (easy for new users)	Steep (hard at first, powerful once learned)
Remote Access	Needs remote desktop tools	Easily accessible via SSH

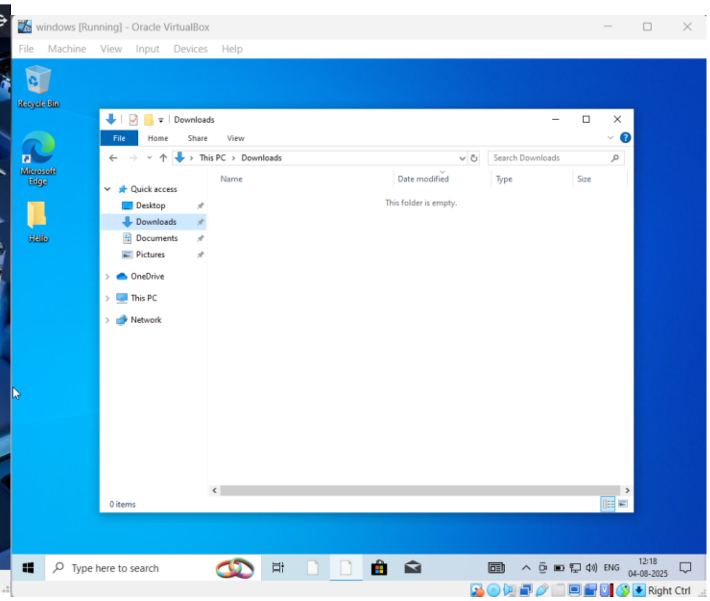
PROJECT-1 BUILD A COMPLETE CYBER LAB ENVIRONMENT

Feature	**GUI**	**CLI**
Error Feedback	Visual indicators (alerts, popups)	Text output or error codes

Practical Part

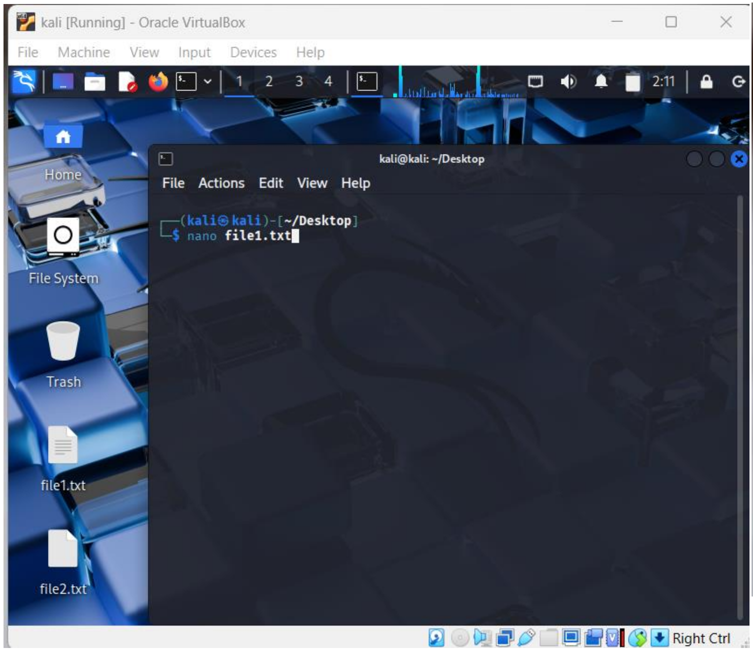
⇨ **File explore in (Windows,Linux) using GUI**

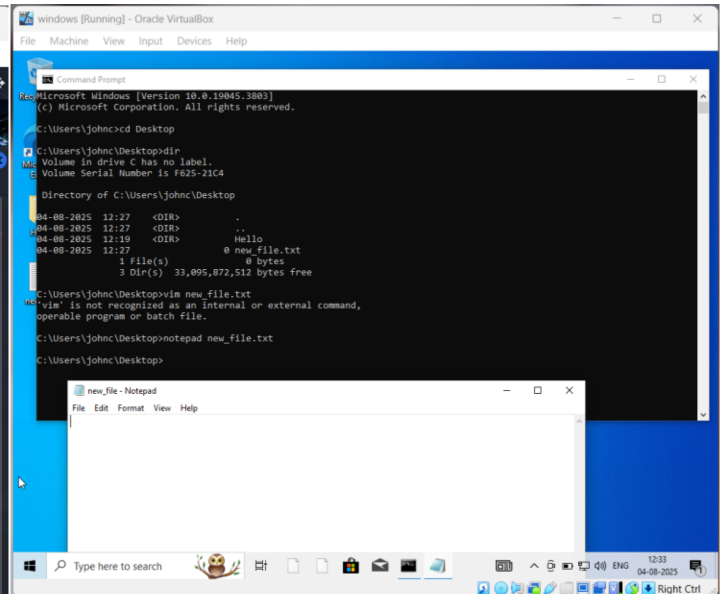




PROJECT-1 BUILD A COMPLETE CYBER LAB ENVIRONMENT

⇨ **CLI based file editor**





- Similarly all the other practical images are in images/GUIvsCLI/ folder

2) **Kernal vs user space**

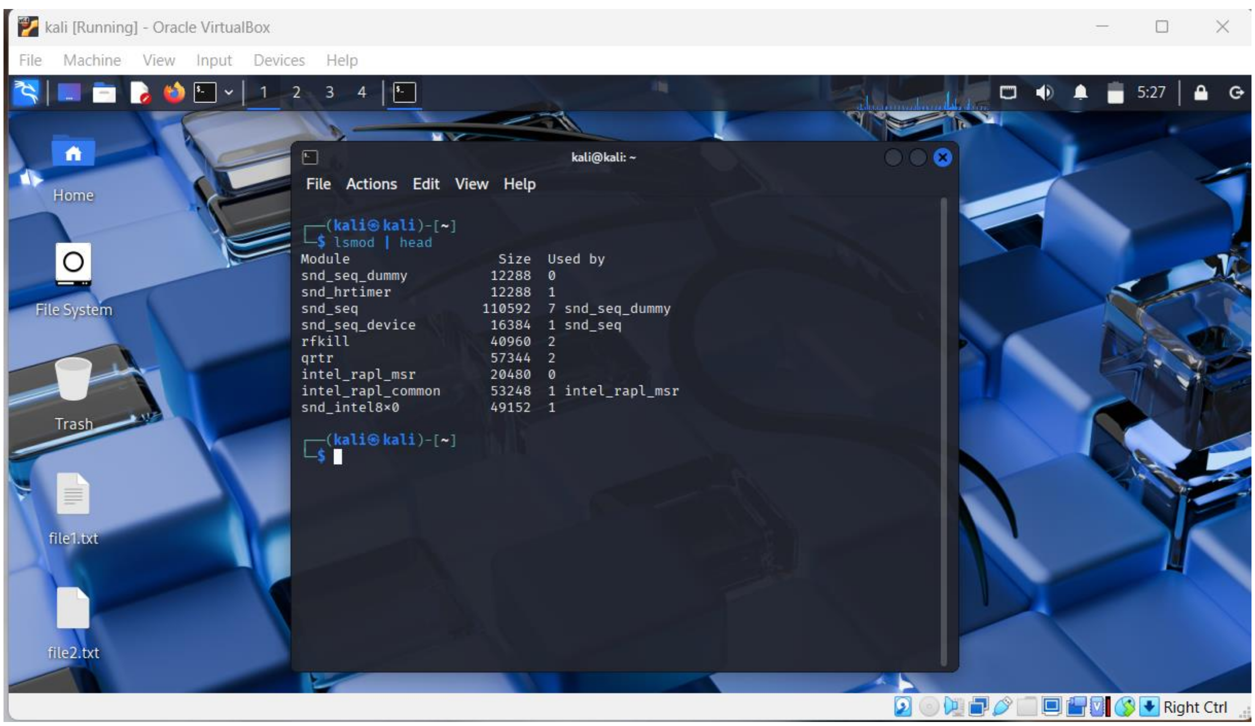
Aspect	**Kernel Space**	**User Space**
Access Level	**Full access to hardware and system resources**	**Limited access; must use system calls to communicate with kernel**
Security	**Highly privileged**	**Less privileged — sandboxed from direct hardware access**
Crash Impact	**Kernel crash = entire system crash (kernel panic)**	**User app crash = only that app fails**
Examples	**Device drivers, process scheduler, memory manager**	**Web browser, text editor, terminal**
Code Execution	**Runs in Ring 0 (privileged mode)**	**Runs in Ring 3 (non-privileged mode)**

PROJECT-1 BUILD A COMPLETE CYBER LAB ENVIRONMENT

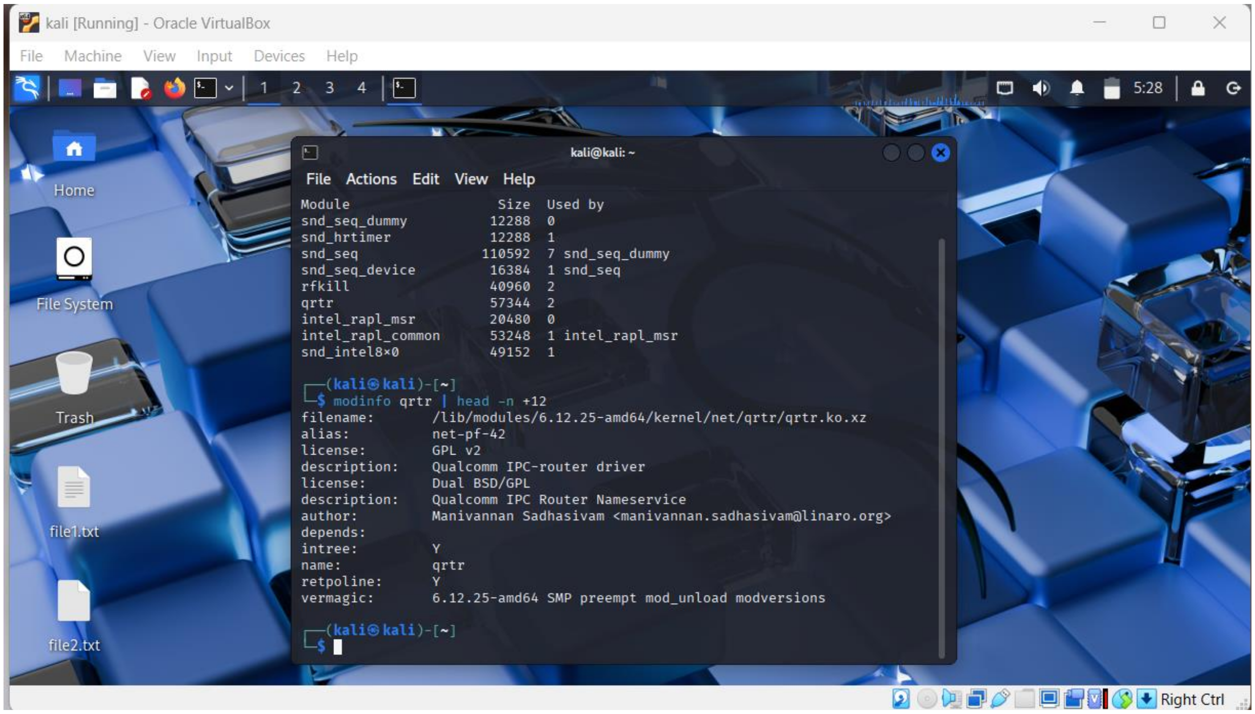
Aspect	Kernel Space	User Space
Memory Protection	Kernel protects itself from user space	User space is isolated from the kernel
Performance	Handles critical tasks quickly	May be slower due to context-switching when calling kernel services

- **Practical on Kernel and User space**

⇨ List of kernel modules and description of each modules

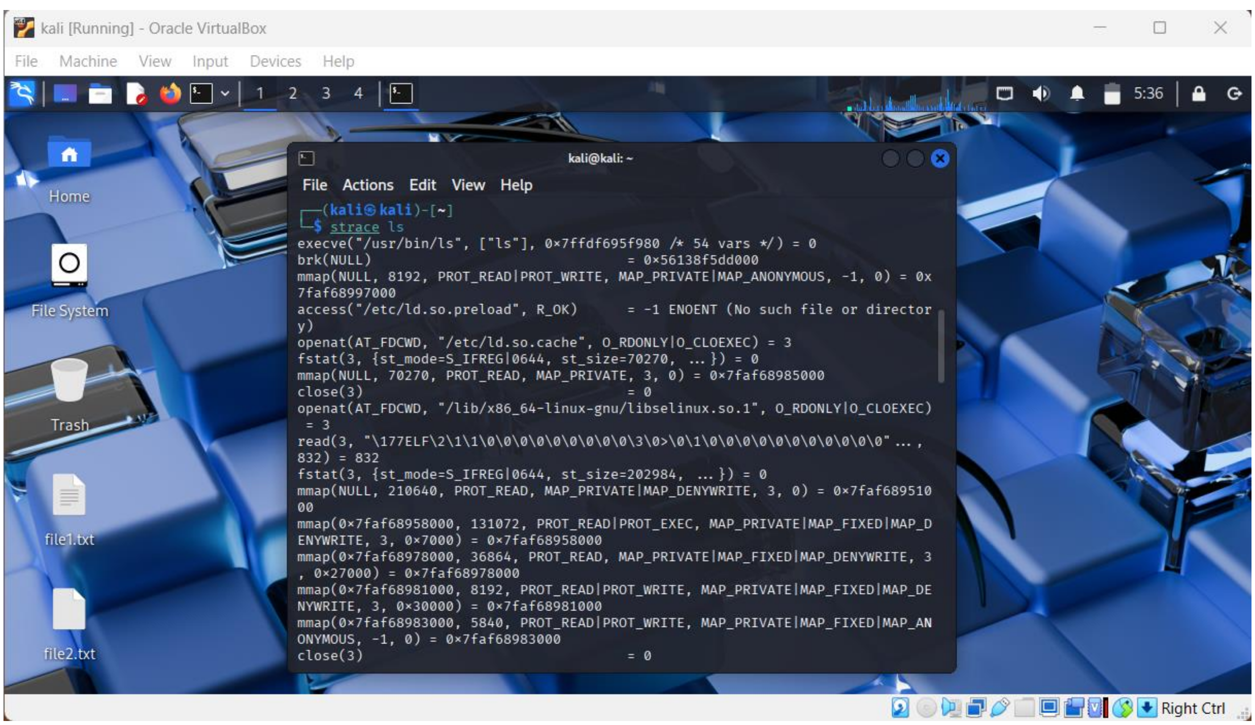


PROJECT-1 BUILD A COMPLETE CYBER LAB ENVIRONMENT



- ⇨ tracing user space process converting into kernel module using `strace <process/command> `

  ex: strace ls



- ⇨ Similarly also performed list of user space using “ps -aux” , getting logs of kernel buffer using “dmesg”

⇨ Also performed in windows rest of images are in /images/KernelVsUser/ folder

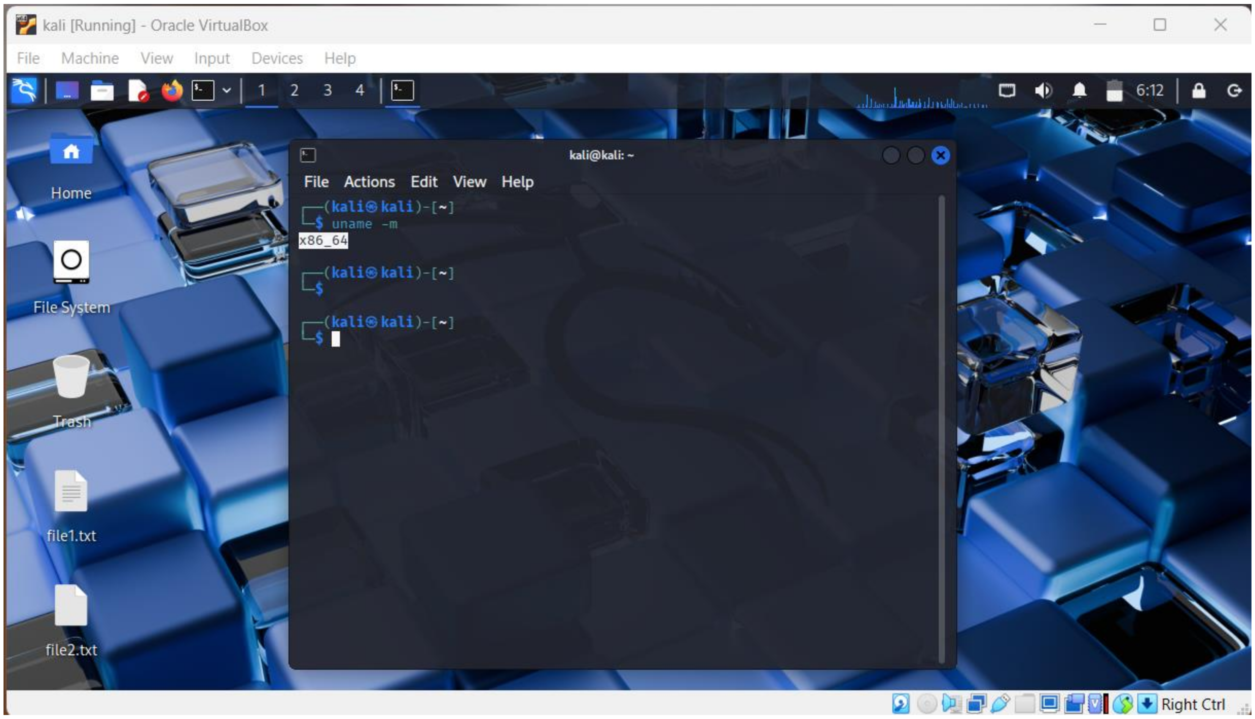
---

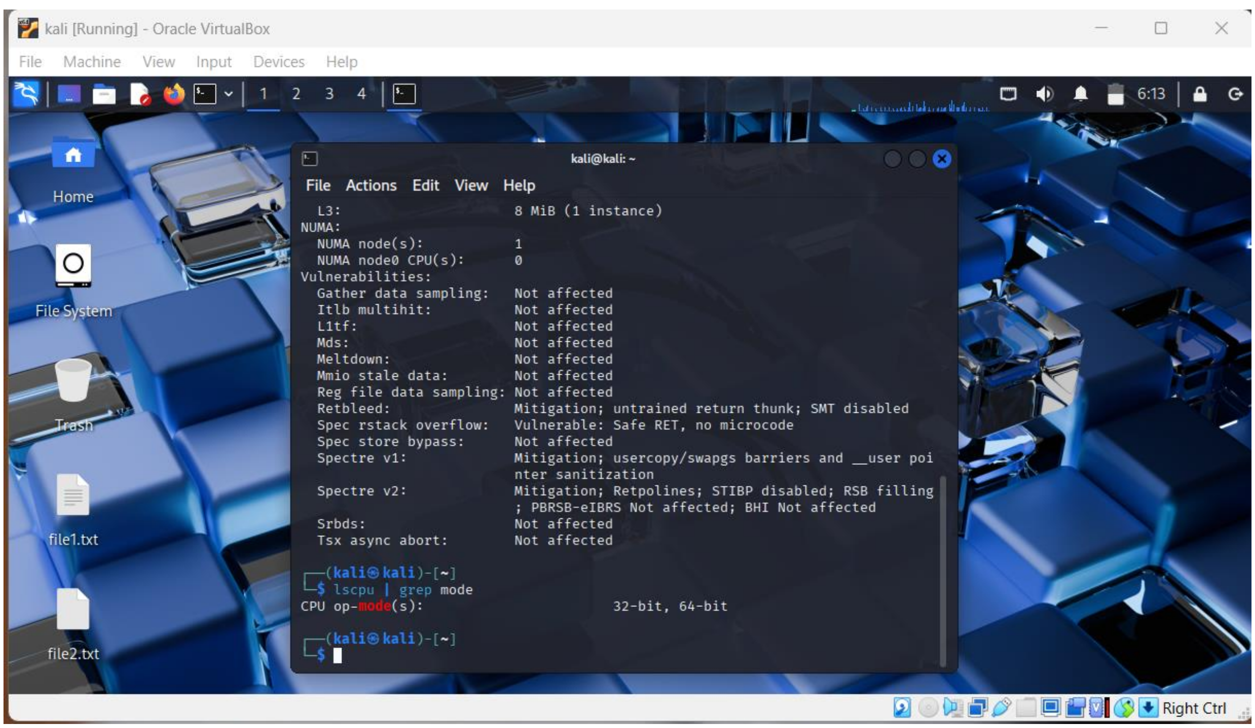
3) 32-bit vs 64 bits

Criteria	**32-bit**	**64-bit**
Max RAM	~4 GB	128 GB+
Software Support	Only 32-bit	32-bit & 64-bit
OS Compatibility	32-bit only	Both
Performance	Limited	Better, scalable
Security	Basic	Advanced (more secure)

- Practical Part
  - ⇨ uname -m
    i686, i386 → 32-bit
    x86_64 → 64-bit in Linux
  - ⇨ systeminfo | findstr /C:“System Type” in windows

PROJECT-1 BUILD A COMPLETE CYBER LAB ENVIRONMENT





Note: All these images are also mentioned in Images/32vs64 bits/

PROJECT-1 BUILD A COMPLETE CYBER LAB ENVIRONMENT

2.2 Process and Service Management

=> Using Systemctl for creation , removing services, creating custom service

- **Systemctl :** Using Systemctl for creation , removing services, creating custom service

---

Creating & Deleting a Custom systemd Service on Linux

Part 1: Create a Custom Service

Step 1: Create the Script

Create the executable script that will run when the service starts.

sudo nano /usr/local/bin/custom_script.sh

Paste this content:

#!/bin/bash

echo "✅ Custom service ran on boot!" >> /var/log/custom_log.log

Make it executable:

sudo chmod +x /usr/local/bin/custom_script.sh

Step 2: Create the Service File

Create a new service file in /etc/systemd/system/:

sudo nano /etc/systemd/system/custom-boot.service

Paste this configuration:

```
[Unit]
Description=✅ My Custom Boot Service
After=network.target

[Service]
```

```
ExecStart=/usr/local/bin/custom_script.sh
Type=oneshot

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

Step 3: Reload systemd and Enable the Service

sudo systemctl daemon-reload

sudo systemctl enable custom-boot.service

sudo systemctl start custom-boot.service

✅ Verify log:

cat /var/log/custom_log.log

Part 2: Delete the Custom Service

▼ Step 1: Stop & Disable

sudo systemctl stop custom-boot.service

sudo systemctl disable custom-boot.service

Step 2: Remove Files

sudo rm /etc/systemd/system/custom-boot.service

sudo rm /usr/local/bin/custom_script.sh

sudo rm /var/log/custom_log.log # Optional

Step 3: Reload systemd

sudo systemctl daemon-reload

Confirm removal:

systemctl status custom-boot.service

Expected:
Unit custom-boot.service could not be found.

Process vs Thread:

Process

- An independent unit of execution with its own memory and resources.
- Heavyweight – creating and managing processes is resource-intensive.
- Crash in one process doesn't affect others.
- Used for isolated tasks, like running separate programs.

Thread

- A lightweight unit within a process; shares memory/resources with other threads in the same process.
- Faster to create and switch.
- Crash in one thread can affect the whole process.
- Used for parallelism within the same application.

Chapter 3: Network Configuration

3.1 Configure different network mode

1. **NAT (Network Address Translation)**
   Default & safest for internet access
   What it does:
- VM shares host's internet using virtual NAT router.
- VM can access the internet.

- Other machines cannot access the VM.

Use Case:

- Safe internet access for updates or downloads.
- Good for penetration testing tools that require outbound connections.

Configure:

- Go to **VM Settings → Network → Attached to: NAT**
- No extra setup required.

2. Bridged Networking

VM acts like a real device on your LAN

What it does:

- VM connects directly to your physical network (LAN).
- Gets an IP address from your router, just like your host PC.
- Other devices on the same LAN can reach the VM.

Use Case:

- Testing network tools like Wireshark, Nmap, FTP servers.
- Simulate real device behavior.
- Use VM as a server visible to other devices.

Configure:

- Go to **VM Settings → Network → Attached to: Bridged Adapter**
- Select correct physical network interface (Wi-Fi/Ethernet).

3. Host-Only Adapter

VM talks **only with host**

What it does:

- VM is **isolated from internet & other networks**.
- VM **can communicate with the host** via a virtual adapter.
- IPs are from a private range (e.g., 192.168.56.0/24).

Use Case:

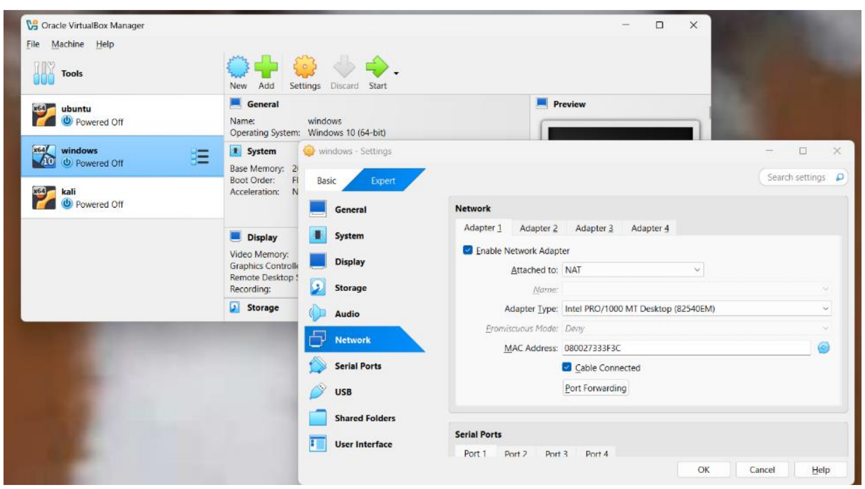
- Lab environments where **host needs to send files to VM**.

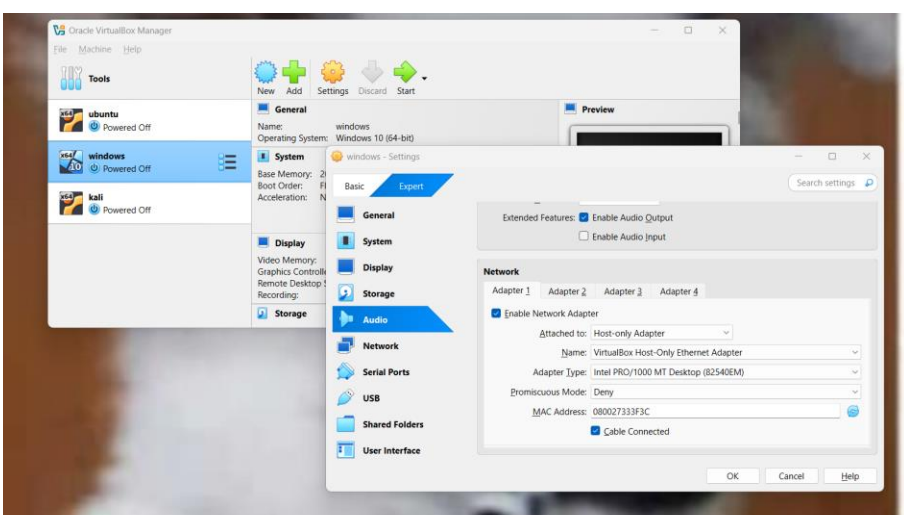
PROJECT-1 BUILD A COMPLETE CYBER LAB ENVIRONMENT

- Secure setups with **no internet** access.
- Testing or capturing traffic between host and VM.

  Configure:
- Go to **VM Settings → Network → Attached to: Host-Only Adapter**
- Ensure **VirtualBox Host-Only Network Adapter** is installed.

Practical Images



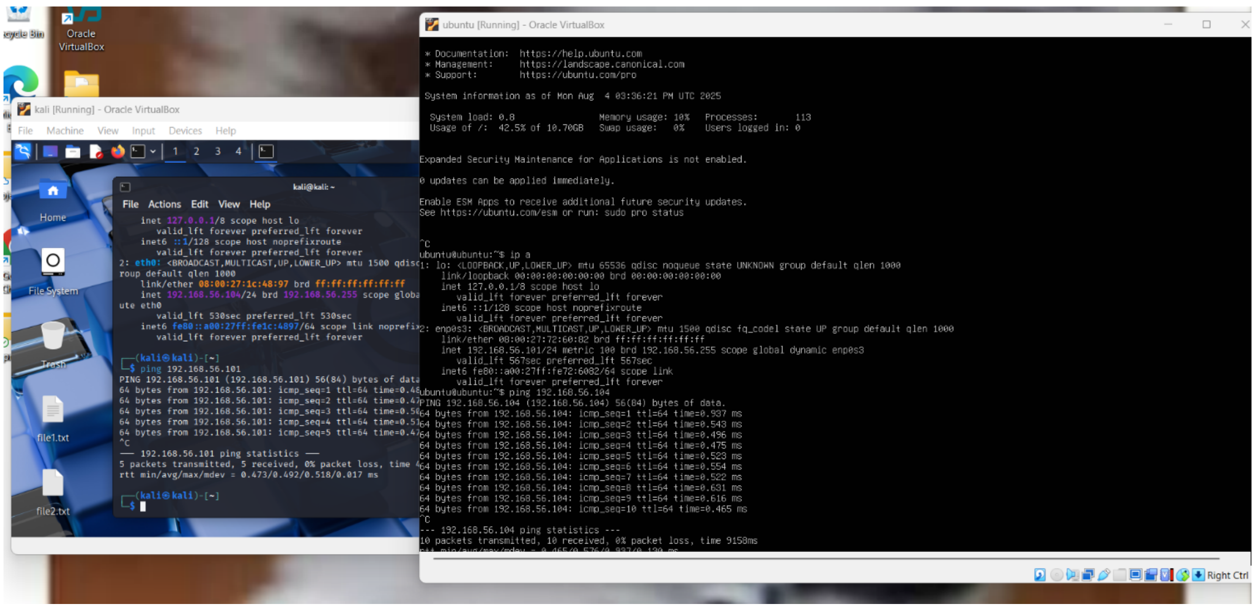


3.2 Set up Communication between VMs

⇨ After configuration of Network mode

   - Use “ ip a” in Linux to know IP address of Linux VM
   - Use “ip a” in Ubuntu to Know IP address of Ubuntu VM
  - ⇨ Use “ping < Ubuntu IP address> through Linux VM
  - ⇨ Use “ping <Linux IP address> through Ubuntu VM
- If successful “n packets transferred, n received, 0% loss, time km/s

Practical Images



Note: Windows VM doesn’t support HOST-only adapter

3.3 Few network troubleshoot like ping , traceroute (tracert), netstat (ss)

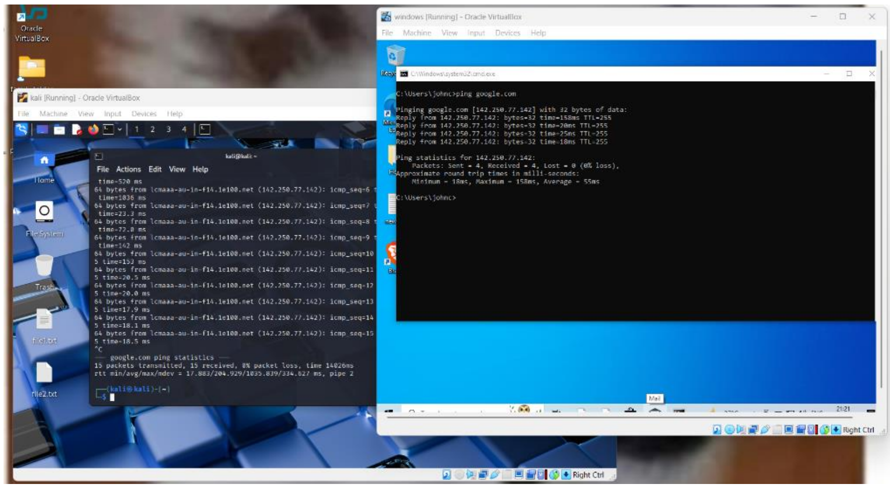
=> ping: Tests connectivity between your system and another network host by sending ICMP echo requests**.**

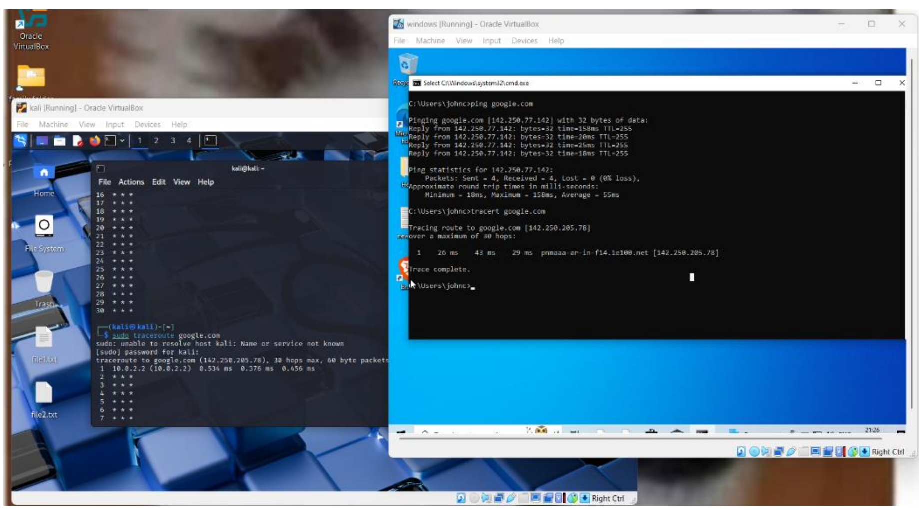
=> traceroute / tracert: Traces the path packets take to reach a destination by showing each hop along the route.

=> netstat / ss: Displays active network connections, listening ports, and socket statistics on your system.

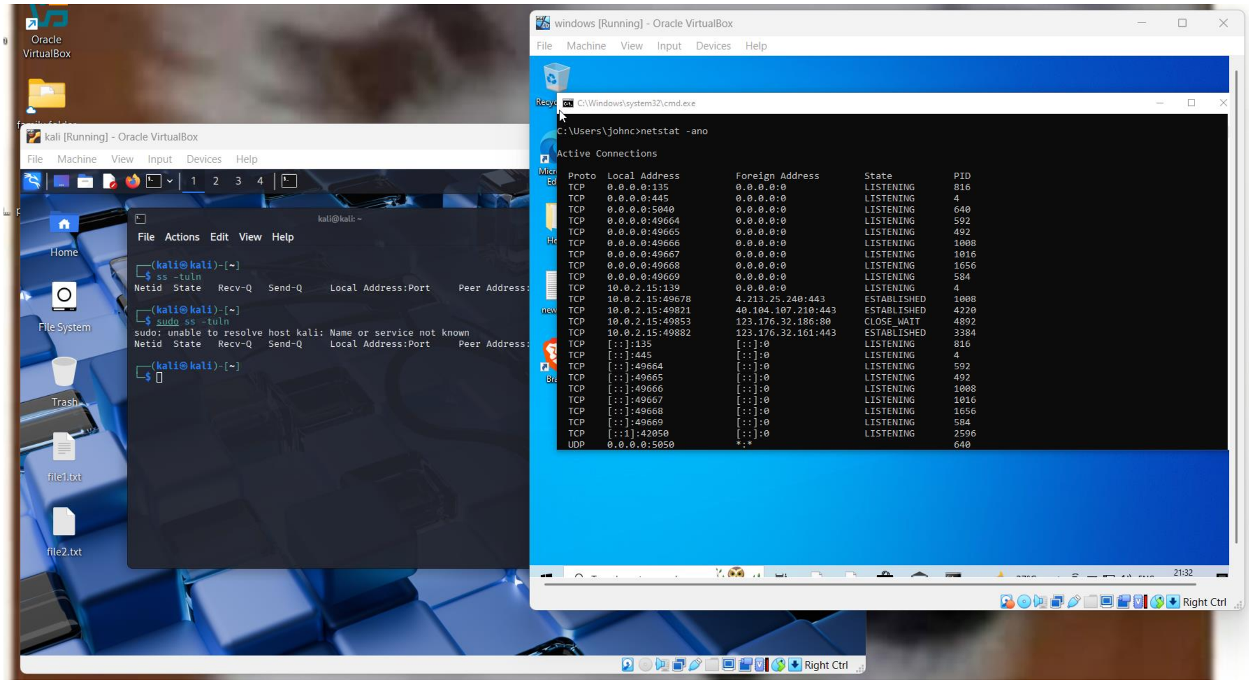
PROJECT-1 BUILD A COMPLETE CYBER LAB ENVIRONMENT

Practical Images:





PROJECT-1 BUILD A COMPLETE CYBER LAB ENVIRONMENT



Lincense:

This project is for personal learning and not licensed for distribution