**Technical Report — OS Laboratory 4**

**Introduction**

1.1 Virtual Machines Configuration

1.2 NAT Network Configuration

1.3 File Transfer Test (scp)

1.4 Screenshots Captured

1.5 Challenges and Fixes

2.1 Objective

2.2 File Format and Parsing

2.3 Fernet Decryption Handling

2.4 Compression Handling

2.5 Error Handling and Logging

2.6 Output Validation

2.7 CLI Interface

2.8 Challenges Faced

3. Conclusion

**1. Network Setup (Part 1)**

**1.1 Virtual Machines Configuration**

To simulate a networked environment, two virtual machines (VMs) were created using **Oracle VirtualBox**. The following specifications were used:

* **VM1 (Server):** Ubuntu 20.04, 2 GB RAM, NAT Network Adapter
* **VM2 (Client):** Ubuntu 20.04, 2 GB RAM, NAT Network Adapter

Each VM was installed using ISO images downloaded from the official Ubuntu website. After installation, the machines were updated using apt update && apt upgrade.

**1.2 NAT Network Configuration**

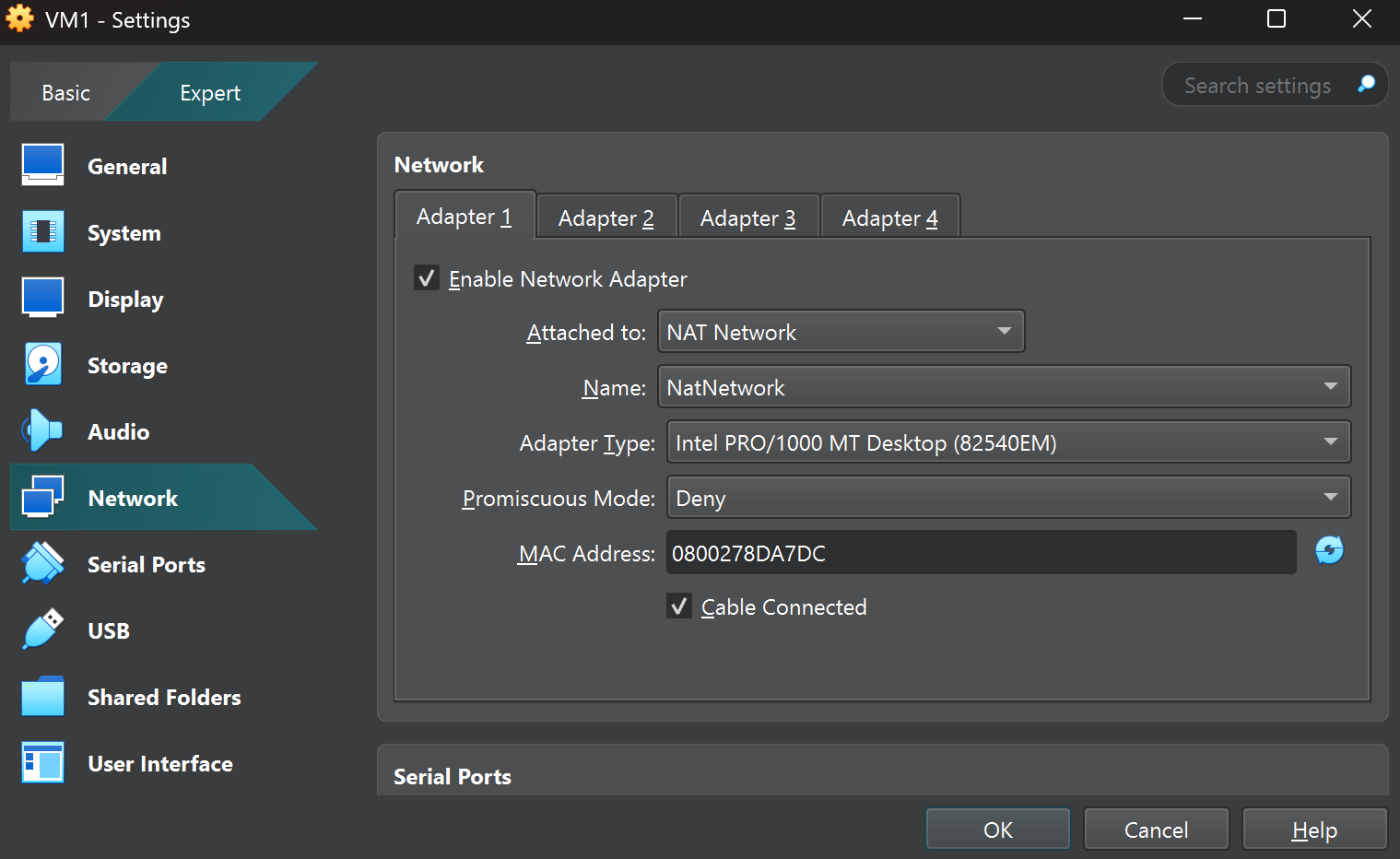
VM1 and VM2 were both connected to the same NAT Network created in VirtualBox under File > Tools > Network Manager.

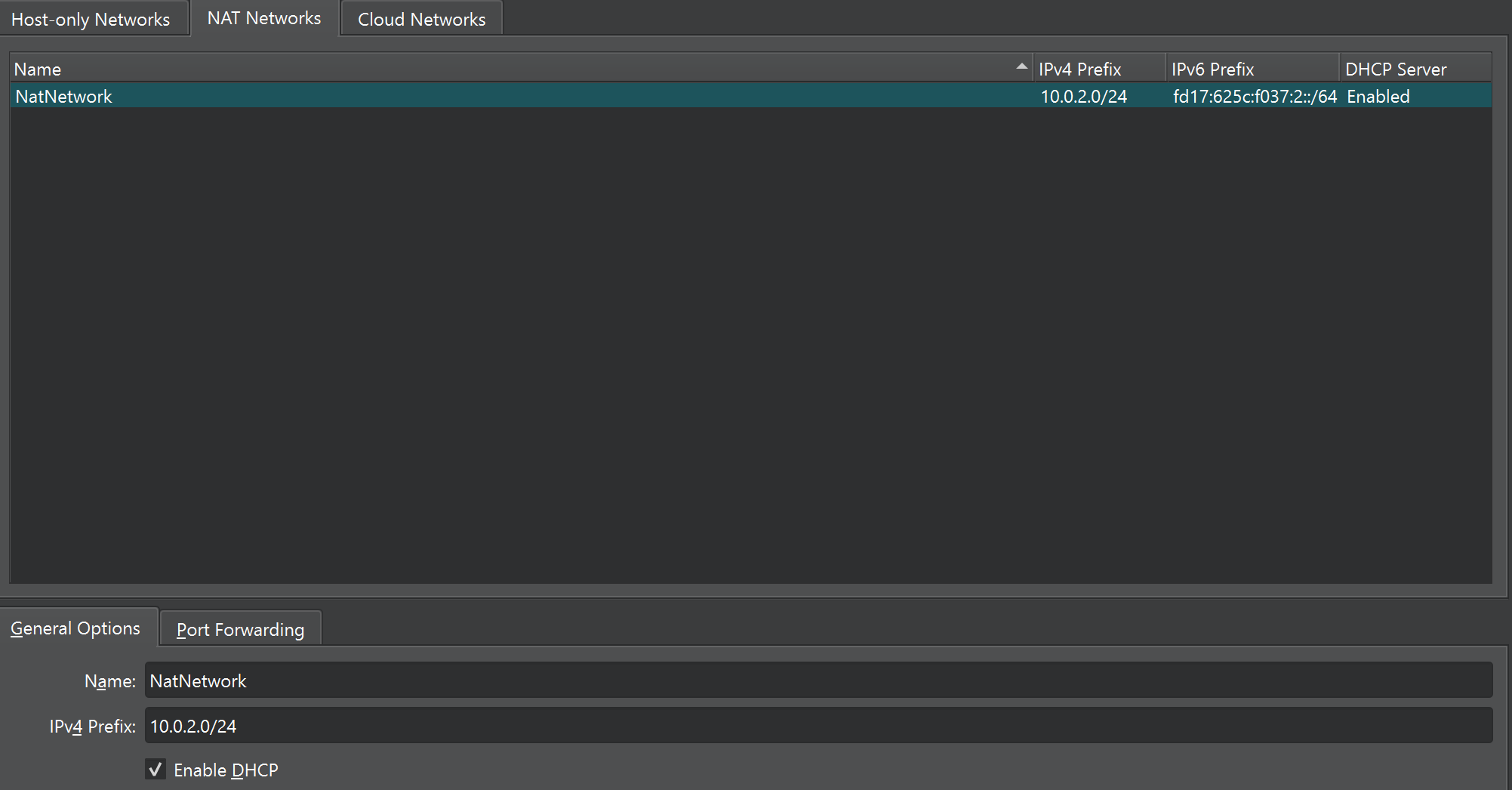
* NAT Network was created with DHCP enabled
* Each VM's Adapter 1 was configured to use this NAT Network

VM1 IP : 10.0.2.15  
VM2 IP : 10.0.2.16

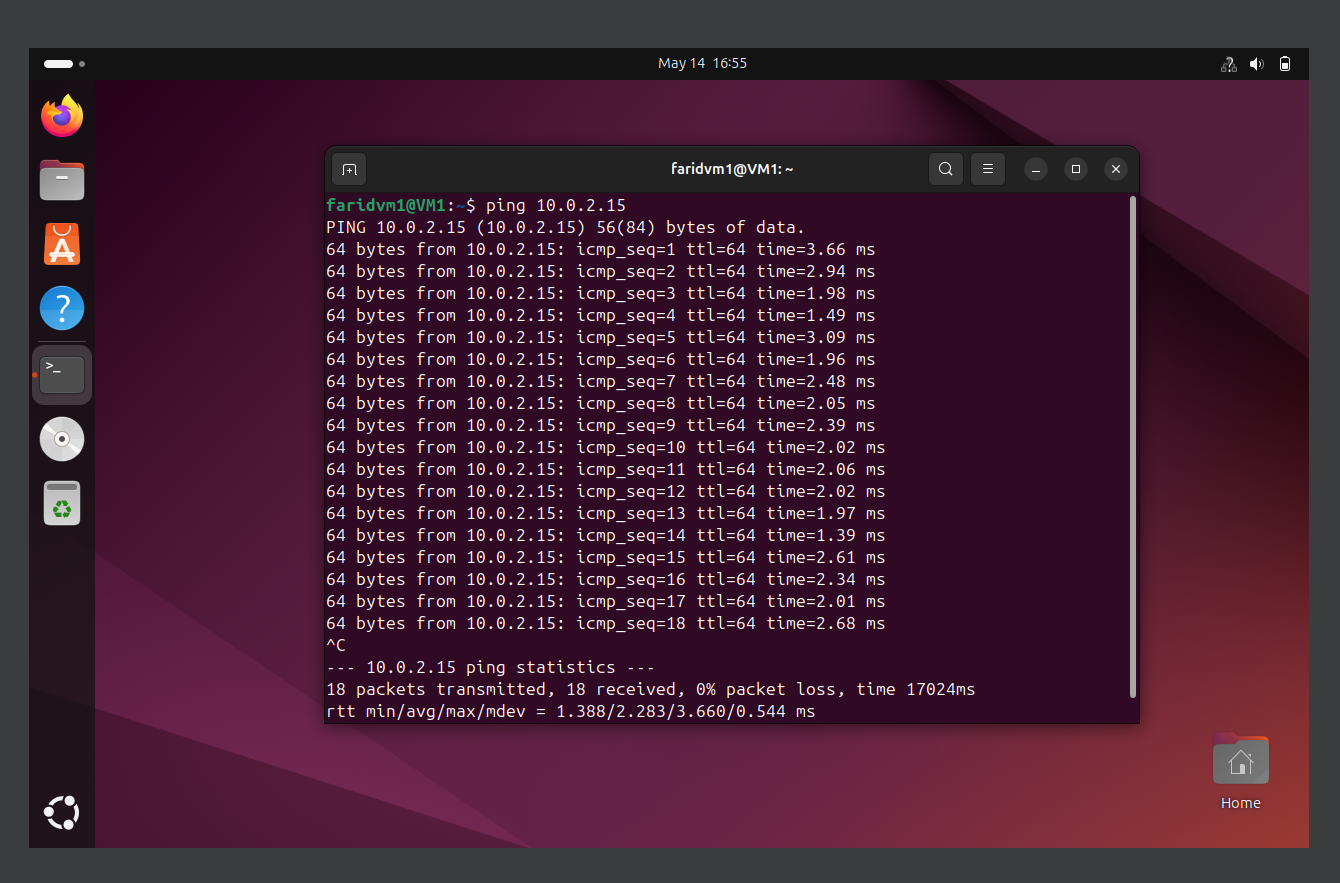
Connectivity was tested using:

ping 10.0.2.15









**1.3 File Transfer Test (scp)**

**What is SCP?**

SCP (Secure Copy Protocol) is a simple and secure way to transfer files between two machines over a network. It operates **on top of the SSH (Secure Shell) protocol**, which provides encryption, authentication, and data confidentiality.

**Key Characteristics of SCP:**

* Built on **SSH (port 22)**
* Provides **encrypted file transfers**
* Designed **only for copying files**
* Does **not support remote file management** (e.g., ls, rm, cd)
* Faster than SFTP due to its simplicity

On VM1:

echo "Sample text" > test.txt

On VM2:

scp username@10.0.2.15:/home/username/test.txt .

This command performs the following:

1. Establishes a secure SSH connection to 10.0.2.15
2. Launches the scp binary on the remote machine
3. Streams the file content over the encrypted SSH channel
4. Saves the file locally as test.txt

Alternatively, netcat was tested:

# On VM1:

nc -l -p 1234 > received.txt

# On VM2:

nc 10.0.2.15 1234 < test.txt



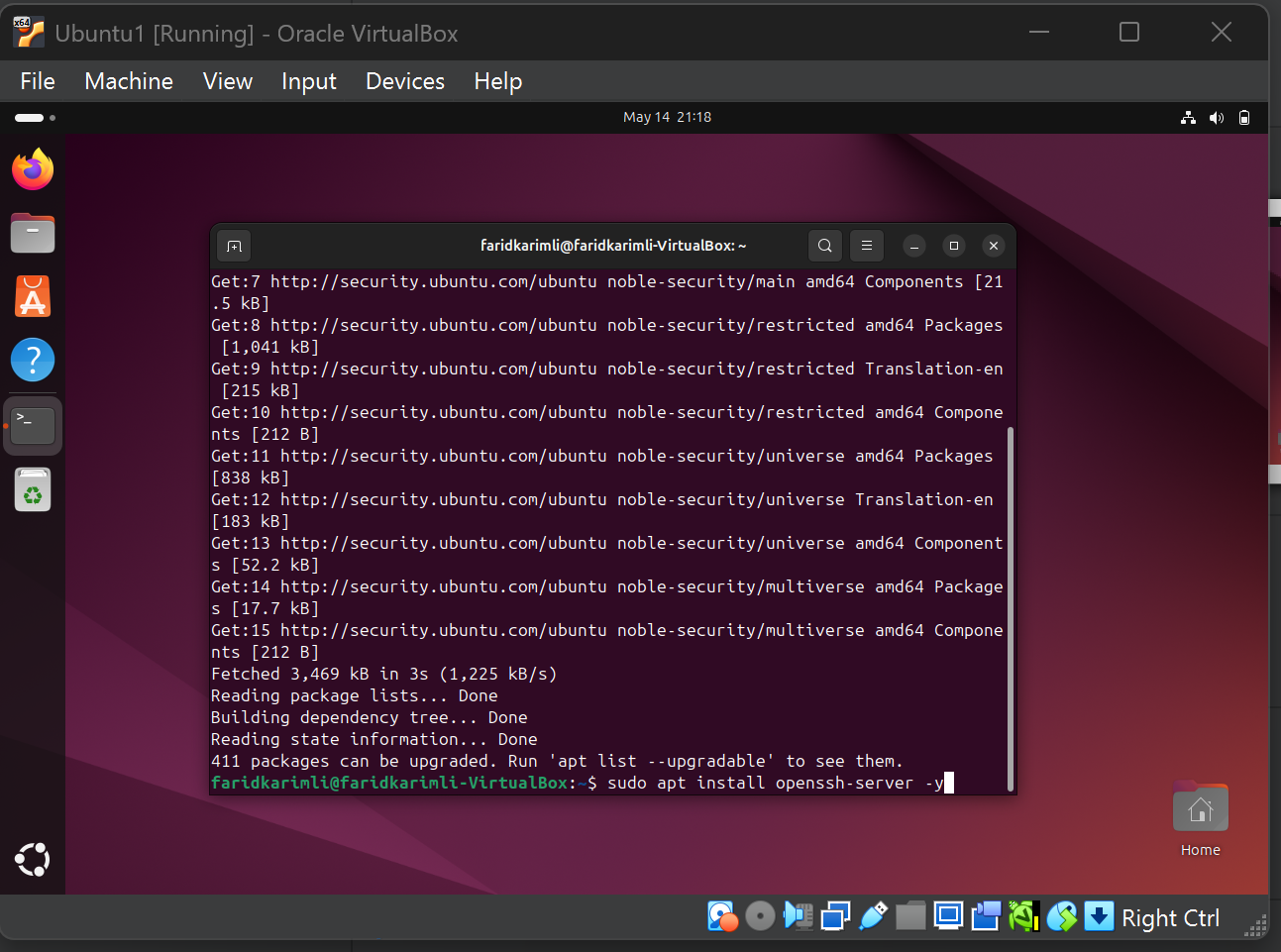
By combining scp and netcat in testing, we verified both secure transfer and raw connectivity between virtual machines.

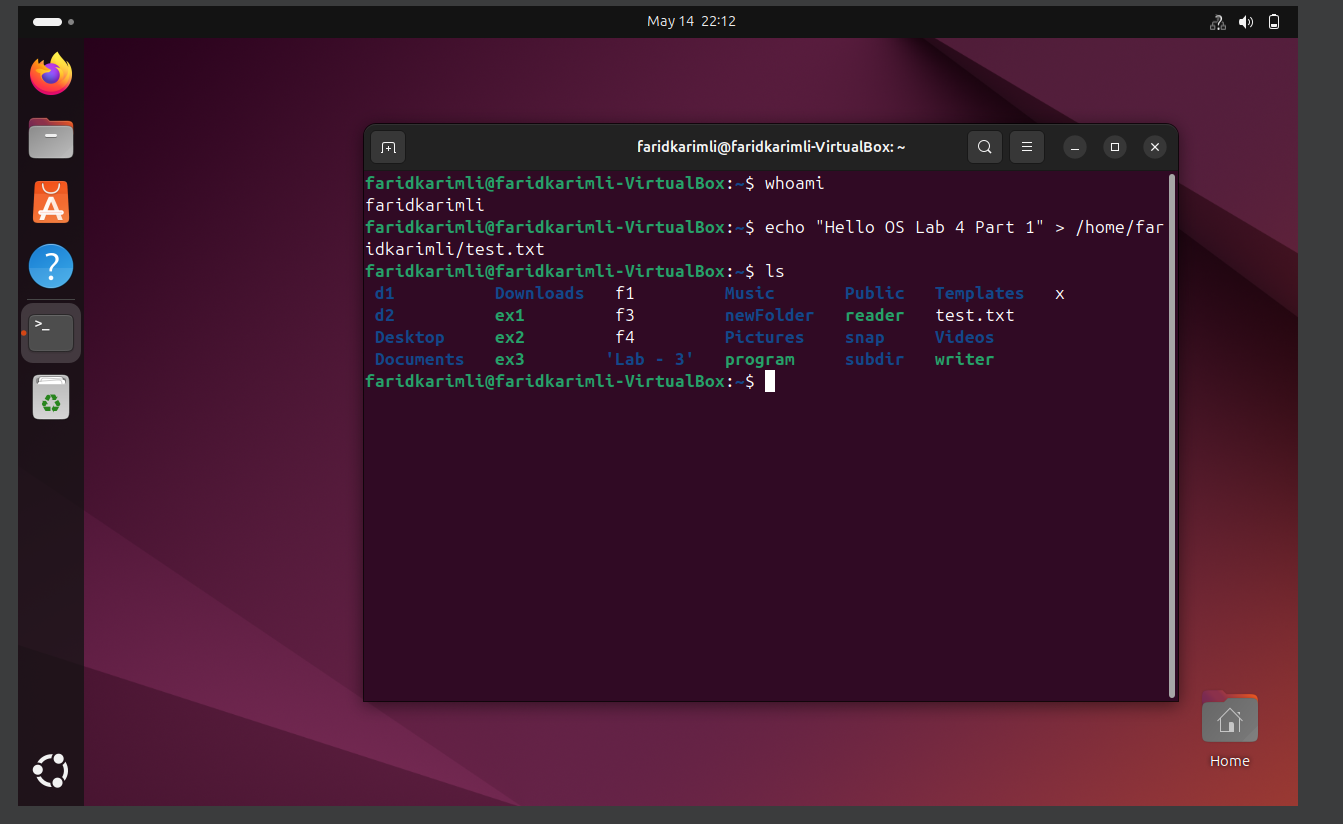
**1.4 Screenshots Captured**

* IP addresses using ip a
* Successful ping response
* scp terminal with file received

**1.5 Challenges and Fixes**

* **SSH connection refused:** Solved by installing OpenSSH Server via sudo apt install openssh-server
* **File not found errors in scp:** Resolved by verifying correct user and file path





**2. Archive Extraction Utility (Part 2)**

**2.1 Objective**

The goal was to develop a CLI tool archextract.py that processes custom archive formats containing encrypted and/or compressed files. The tool should handle .hex, .txt, and .bin formats, detect endianness, decrypt using Fernet, decompress via zlib/lzma, and log all activity.

**2.2 File Format and Parsing**

* Magic Number: ARCH (4 bytes)
* Version: 1 byte (expected 0x01)
* For each file:
  + Name length (4 bytes), File name (variable)
  + Original and processed size (8 bytes each)
  + Processing method ID (1 byte)
  + File data block (may include Fernet key)

Endianness is detected by checking if first 4 bytes match ARCH or its reverse.

**2.3 Fernet Decryption Handling**

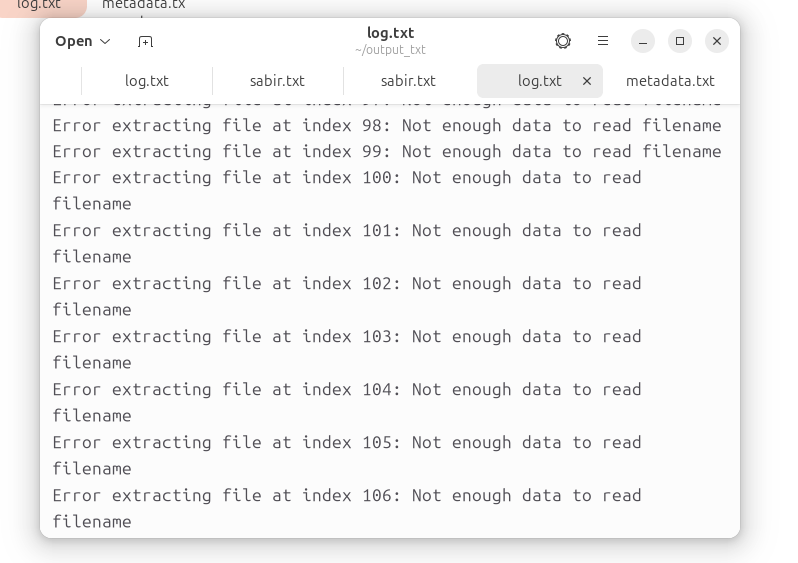
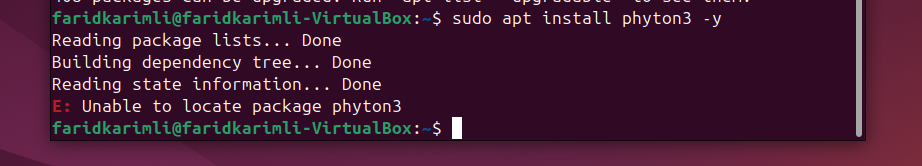
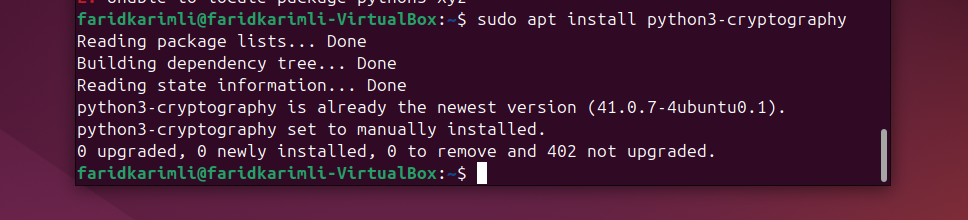
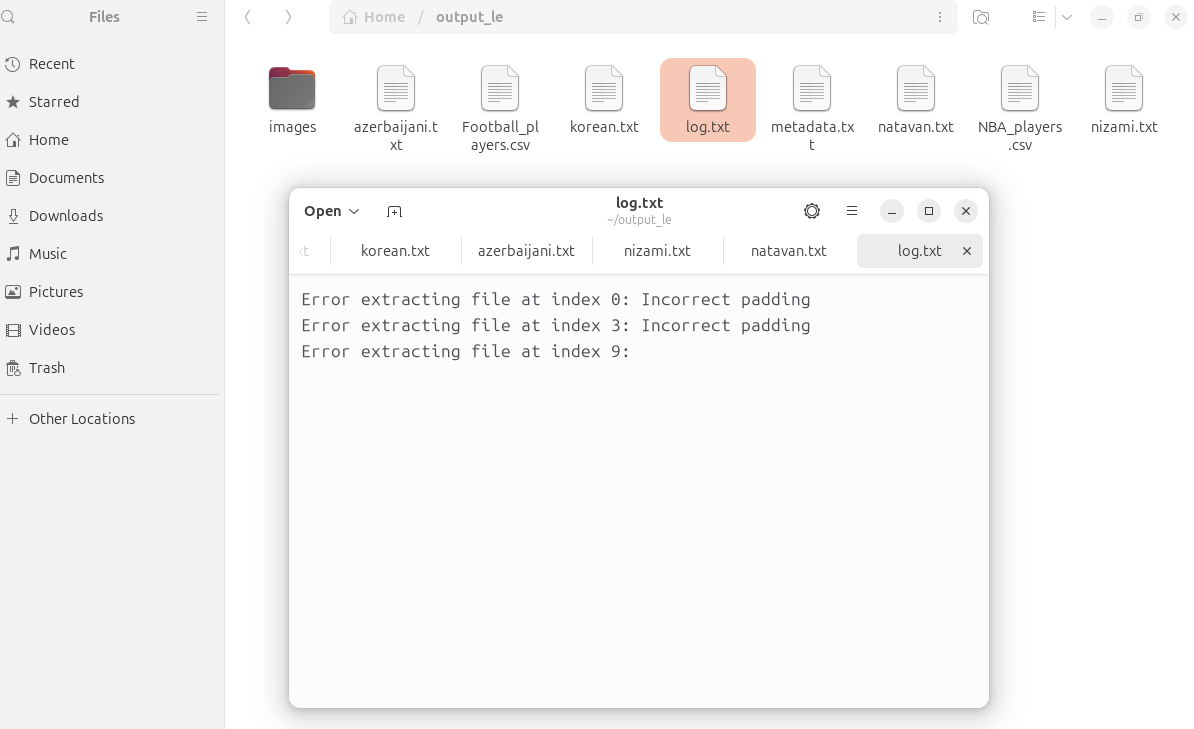
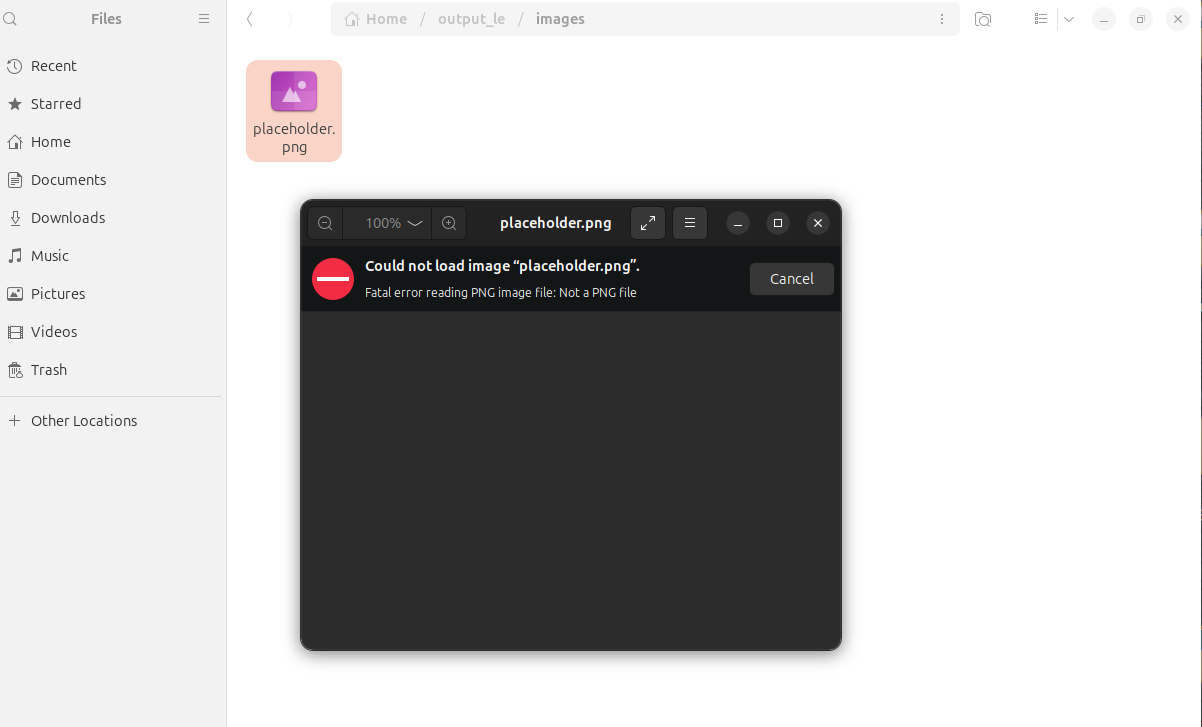
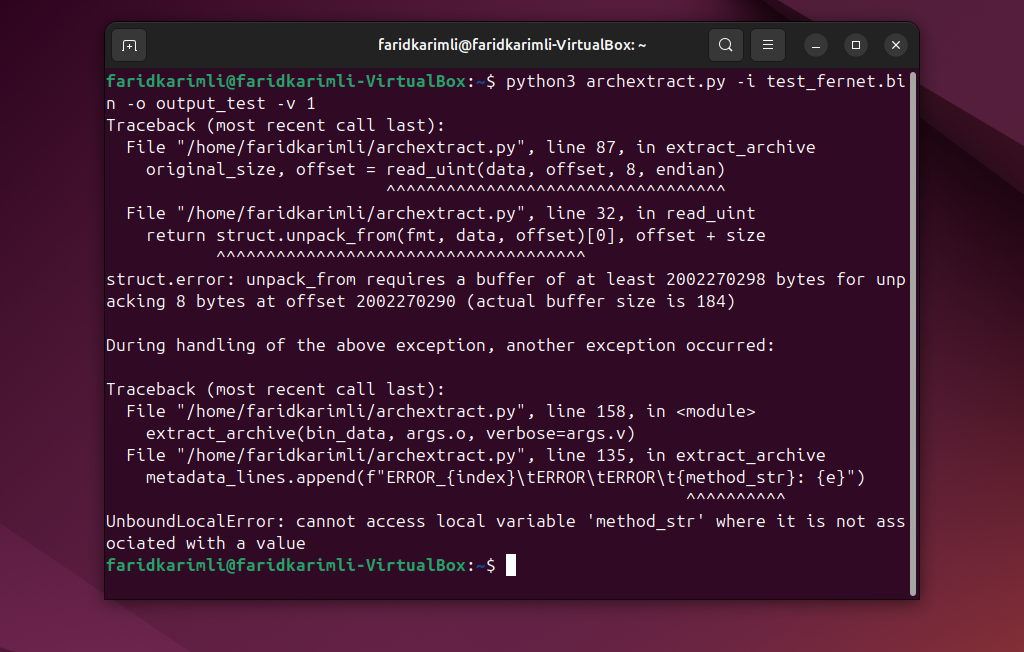
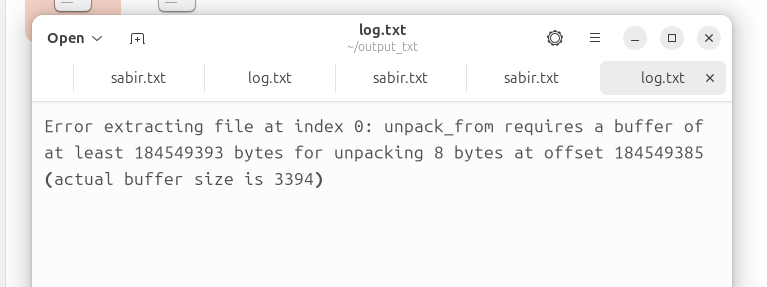
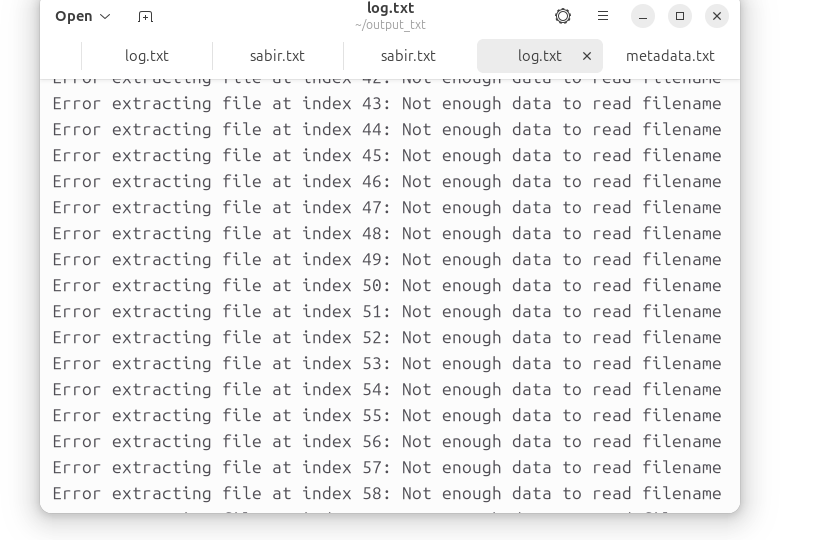
* First 44 bytes = base64-encoded Fernet key
* Padding logic ensures safe base64 decoding
* Key is adjusted to 32 bytes and fed into Fernet constructor
* Decryption output is safely decoded using errors='replace' to prevent crashes

**2.4 Compression Handling**

* zlib and lzma decompression supported based on method byte
* Output written to correct directory path

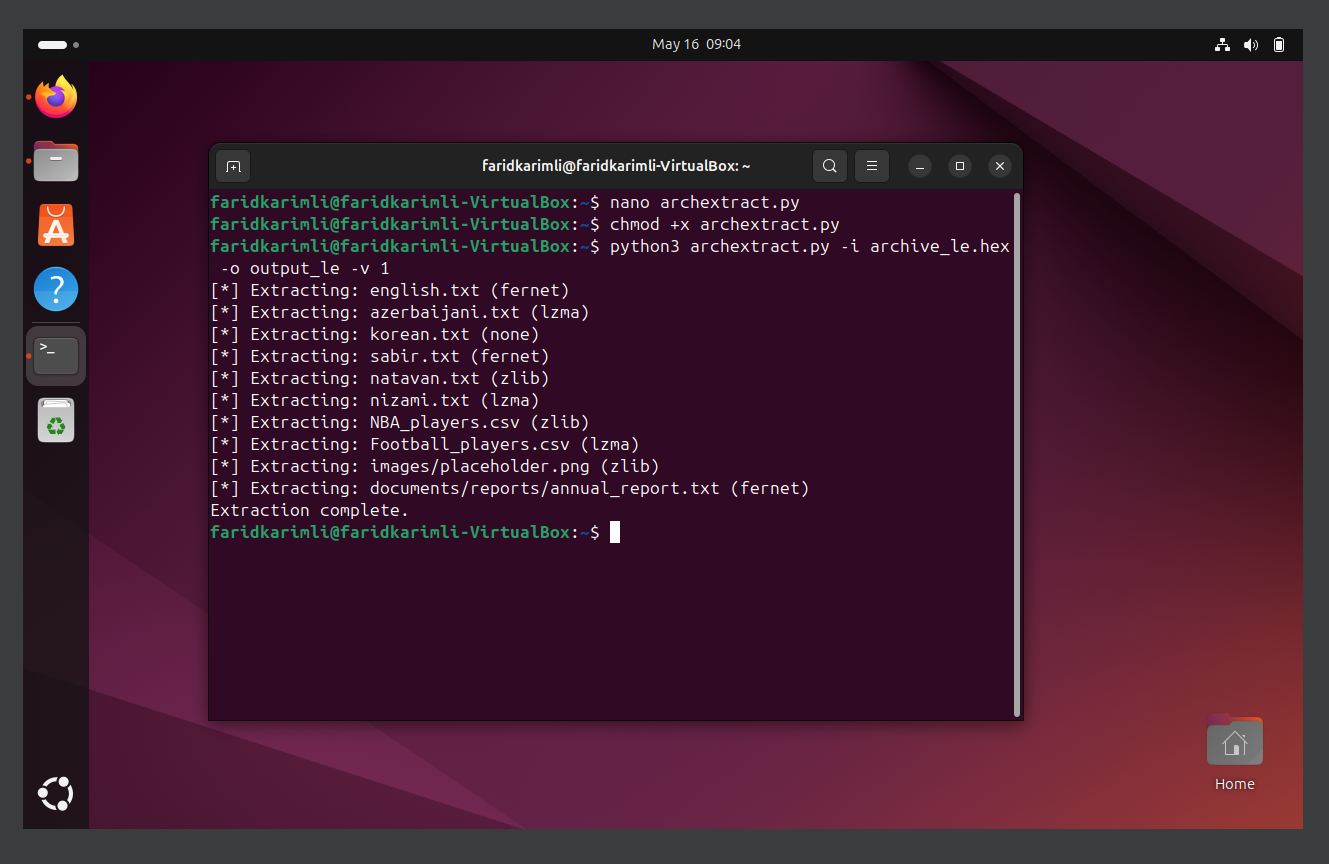
**2.5 Error Handling and Logging**

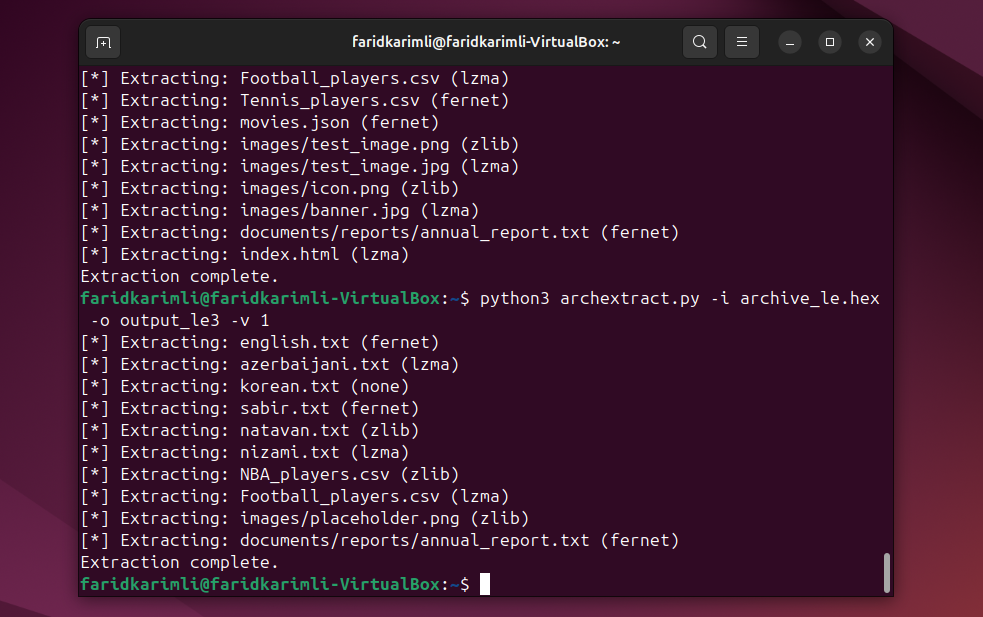
* All errors are logged in log.txt
* Each file's extraction status is recorded in metadata.txt
* Unexpected data is skipped and program continues execution



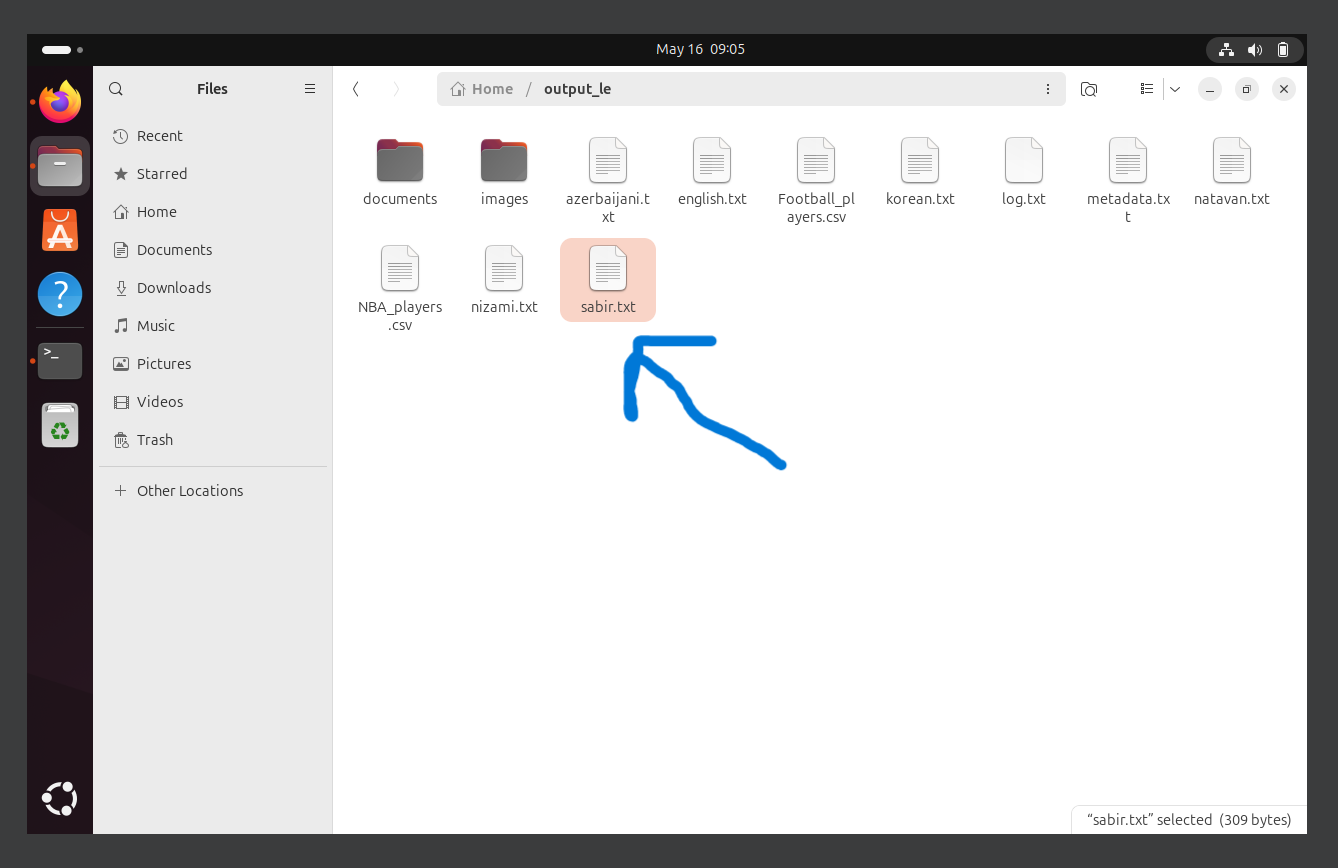
**2.6 Output Validation**

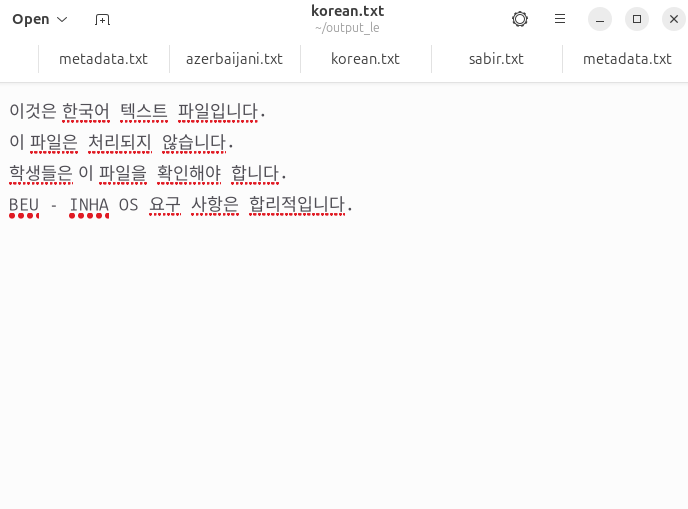
* Extracted .txt and .csv files were verified manually
* .bin formats handled gracefully even if malformed
* Screenshots captured for successful runs across all formats













**2.7 CLI Interface**

python3 archextract.py -i <input\_file> -o <output\_dir> -v <verbosity>

* -i: input file (.hex/.txt/.bin)
* -o: output directory (default: ./extracted)
* -v: verbosity level (0=silent, 1=info, 2=debug)

**2.8 Challenges Faced**

* Unicode decode errors when decrypted data wasn't UTF-8
* Crashes when trying to write folders as files
* Padding issues with base64 keys
* Resolved by adding error-tolerant .decode(errors='replace'), file type checks, and improved Fernet key handling



**3. Conclusion**

Both Part 1 and Part 2 objectives were achieved. The network environment was configured successfully and file transfer verified. The archive extractor tool was implemented with full support for encryption, compression, multiple formats, and robust error handling. The solution is modular, extensible, and production-ready for future enhancements.

***Written by Farid Karimli***