UNIVERSITY OF SCIENCE - HCM NATIONAL UNIVERSITY
Faculty of Information Technology

REPORT
Project Socket

Course: Computer Network

Conducted by:

24127254 - Hồ Đình Trí
24127027 - Lê Minh Duy
24127382 - Trần Nhật Hoàng

Project Supervisor: Nguyễn Thanh Quân

Contents

10th August 2025

1 System Overview

1.1 Introduction

This project simulates a secure file transfer workflow where files are scanned for viruses before being uploaded to the server. Using socket programming, we establish communication between components, implement the FTP protocol, and integrate the ClamAV antivirus engine for real-time security scanning.

The system is composed of three main components working together to ensure safe file uploads:

1.2 Components

1.2.1 FTP Client

A custom-developed client program that serves as the main interface for the user. It accepts FTP-like commands, categorized into three functional groups:

File and Directory Operations Manage files and directories on the FTP server:

- `ls` – List files and directories on the server.
- `cd` – Change the working directory.
- `pwd` – Show the current directory path.
- `mkdir` / `rmdir` – Create or remove directories.
- `delete` – Delete a file from the server.
- `rename` – Rename a file on the server.

Upload and Download Handle file transfers between client and server, including antivirus scanning for uploads:

- `put` – Upload a single file (scanned before upload).
- `mput` – Upload multiple files (all scanned before upload).
- `get` / `recv` – Download a single file.
- `mget` – Download multiple files.
- `prompt` – Toggle confirmation for multiple transfers.

Session Management Control FTP connection settings and session status:

- `ascii` / `binary` – Set file transfer mode.
- `status` – Show the current session status.
- `passive` – Toggle passive FTP mode.
- `open` / `close` – Connect or disconnect from the server.

- `quit` / `bye` – Exit the client application.
- `help` / `?` – Show available commands.

Before any upload, the FTP Client sends the file to the ClamAVAgent for scanning.

- If the result is `OK` → proceed with upload to the FTP Server.
- If the result is `INFECTED` → abort the upload.

1.2.2 ClamAVAgent (ClamAV Server)

A server program running on a separate machine or port. It receives files from the FTP Client via socket communication, scans them using the ClamAV antivirus engine (`clamscan`), and sends back the result (`OK` or `INFECTED`).

1.2.3 FTP Server

An existing FTP server software (in our case, **FileZilla Server**) that stores uploaded files and supports standard FTP operations such as listing, navigation, uploading, and downloading.

1.3 Overall Workflow

1. The FTP Client establishes a command channel with the FTP Server for session control and metadata exchange.
2. Before uploading, the client sends the file to the ClamAVAgent for scanning.
3. If clean, the client proceeds with the upload via the FTP data channel; otherwise, the upload is canceled.
4. Downloads and directory listings occur directly via the FTP Server without antivirus scanning.

2 Diagram(Architectural)

2.1 Diagram

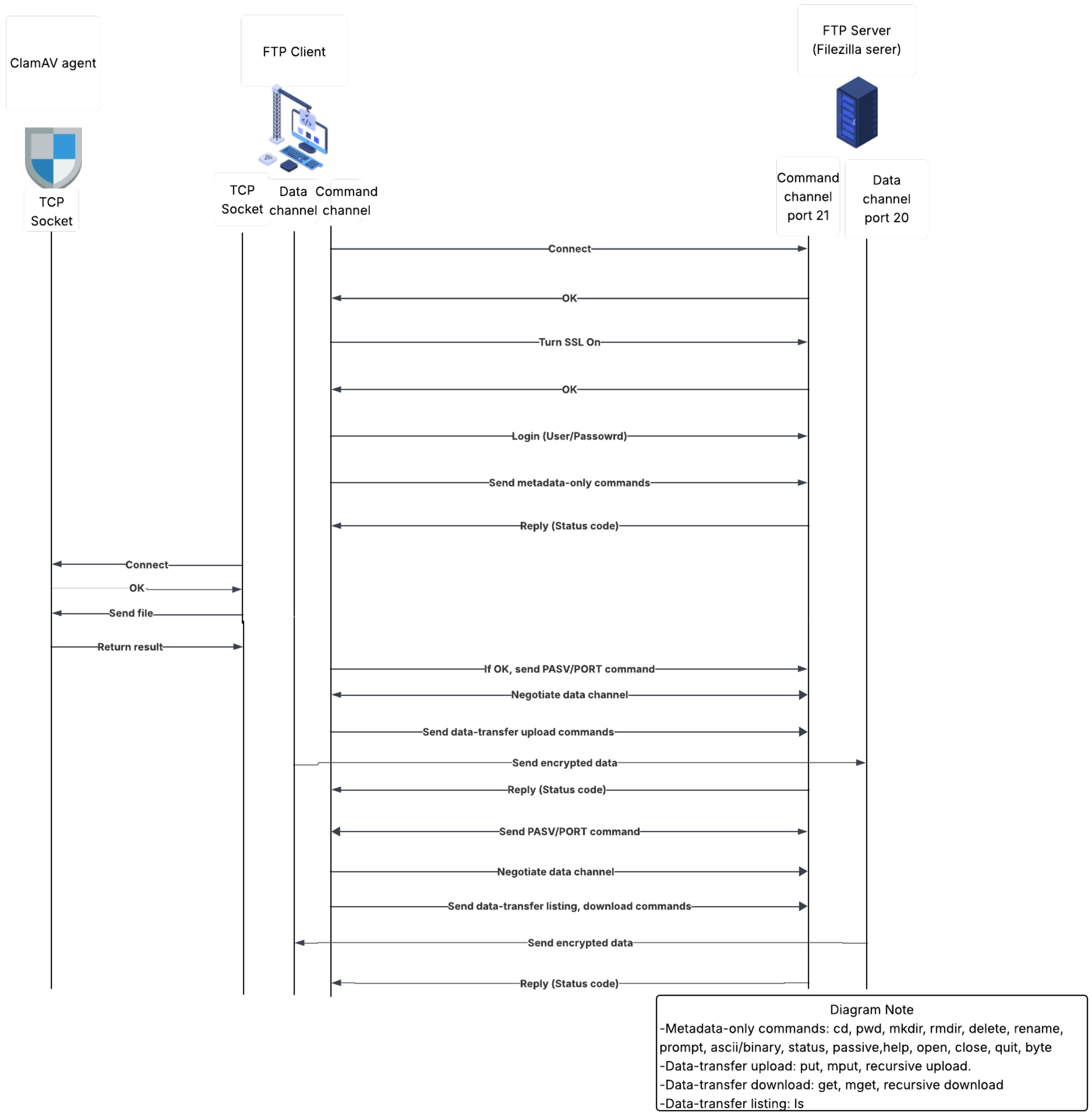

ClamAV agent
FTP Client
FTP Server (Filezilla serer)
TCP Socket
TCP Socket
Data channel
Command channel
Command channel port 21
Data channel port 20
Connect
OK
Turn SSL On
OK
Login (User/Passowrd)
Send metadata-only commands
Reply (Status code)
Connect
OK
Send file
Return result
If OK, send PASV/PORT command
Negotiate data channel
Send data-transfer upload commands
Send encrypted data
Reply (Status code)
Send PASV/PORT command
Negotiate data channel
Send data-transfer listing, download commands
Send encrypted data
Reply (Status code)
Diagram Note
-Metadata-only commands: cd, pwd, mkdir, rmdir, delete, rename, prompt, ascii/binary, status, passive,help, open, close, quit, byte
-Data-transfer upload: put, mput, recursive upload.
-Data-transfer download: get, mget, recursive download
-Data-transfer listing: ls

Figure 1: FTP architectural diagram with ClamAV scanning before upload.

2.2 Description

FTP Architectural Overview

This architecture illustrates the interaction between an FTP Client, an FTP Server (FileZilla), and a ClamAV agent for security scanning. The client and server communicate using two distinct channels:

- **Command channel (Port 21 for Server and Random port for client)** — A persistent TCP connection used to exchange FTP commands (e.g., login, metadata-only commands) and receive status replies.
- **Data channel (Port 20 for Server and Negotiated port for client)** — A temporary TCP connection created for each data transfer operation (upload, download, listing), then closed after completion.

Workflow(in detailed):

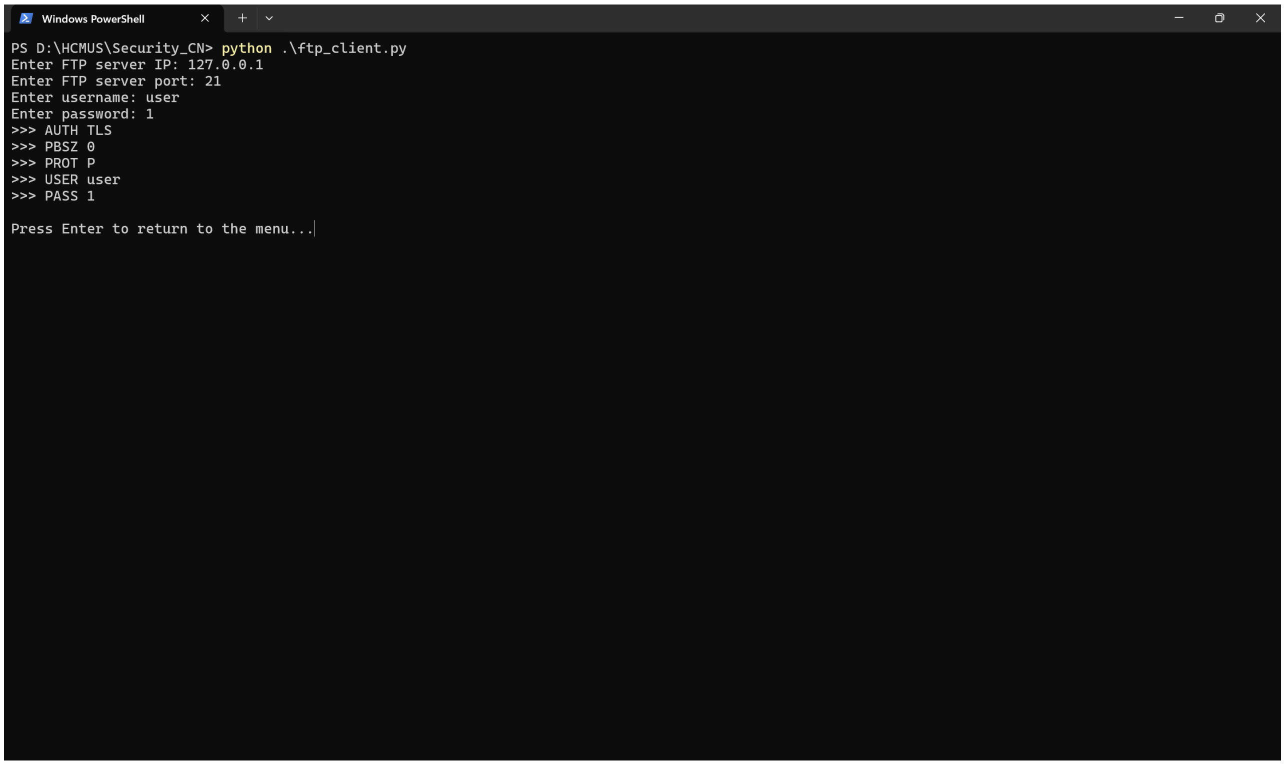
1. The FTP client establishes the command channel to the FTP server, enabling authentication and metadata-only commands (such as directory navigation or file management) without transferring file contents.
2. For uploads, before initiating the FTP data channel, the client sends the file to a ClamAV agent via a separate TCP socket for virus scanning. Only if the scan result is clean does the client proceed with the PASV/PORT negotiation to establish the FTP data channel for upload.
3. For downloads and directory listings, the client directly negotiates the data channel with the server after issuing the appropriate data-transfer commands.
4. All data transfers (upload, download, listing) occur over temporary data channels and can be encrypted if required.
5. Status codes and confirmations are always returned over the command channel.

This separation of concerns ensures that control operations and data transfers are independent, allows for per-transfer security scanning, and keeps the architecture compatible with both active and passive FTP modes.

3 Screenshots of a Successful Session

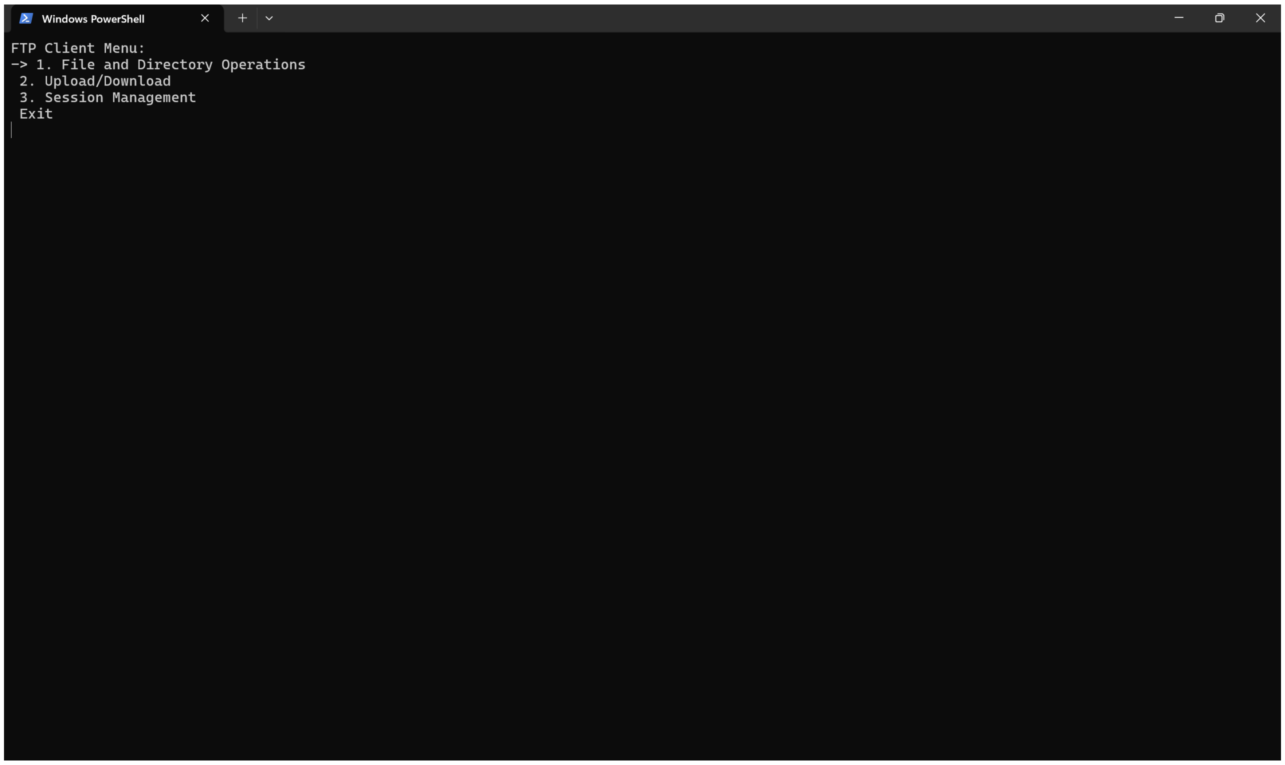
This section presents screenshots of a complete and successful FTP client session with ClamAV antivirus scanning. The session demonstrates all supported commands, including file and directory operations, upload/download, and session management.

1. File and Directory Operations



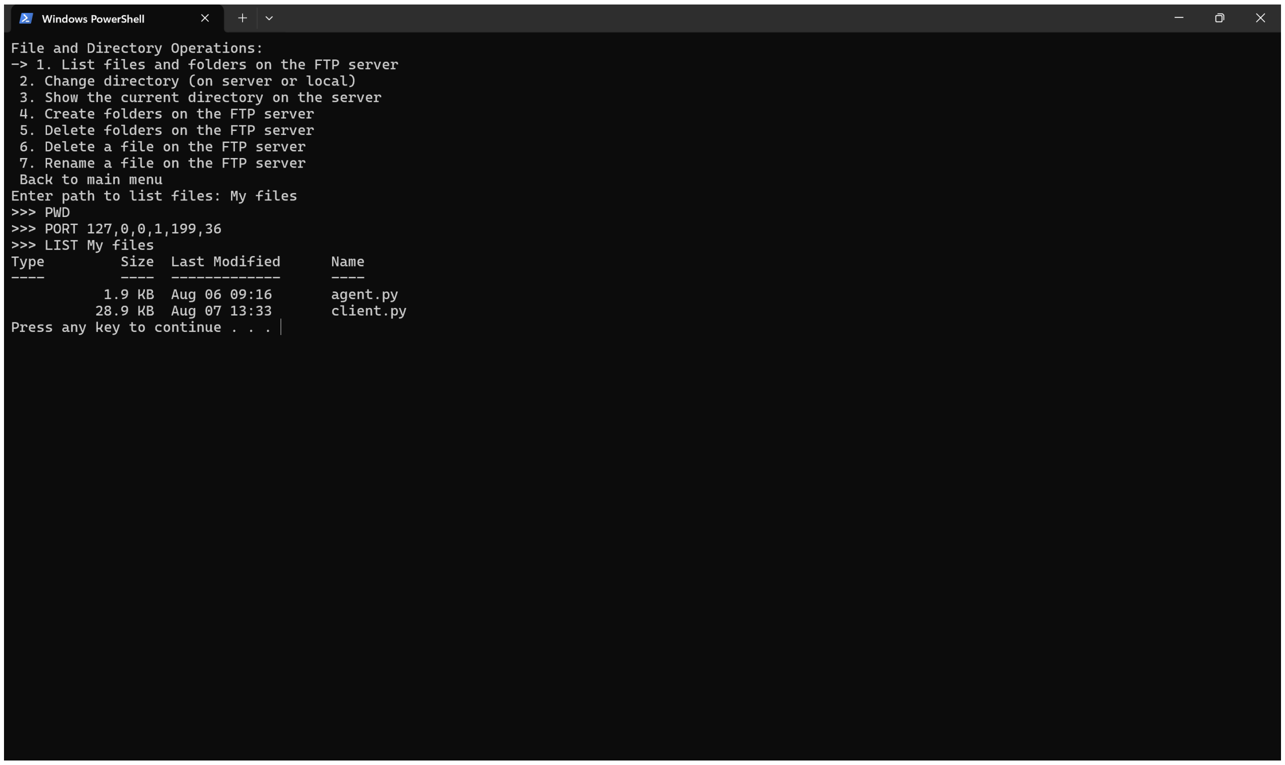
```
PS D:\HCMUS\Security_CN> python .\ftp_client.py
Enter FTP server IP: 127.0.0.1
Enter FTP server port: 21
Enter username: user
Enter password: 1
>>> AUTH TLS
>>> PBSZ 0
>>> PROT P
>>> USER user
>>> PASS 1

Press Enter to return to the menu...
```

Figure 1: Input the ip, port, user, password to connect and login.



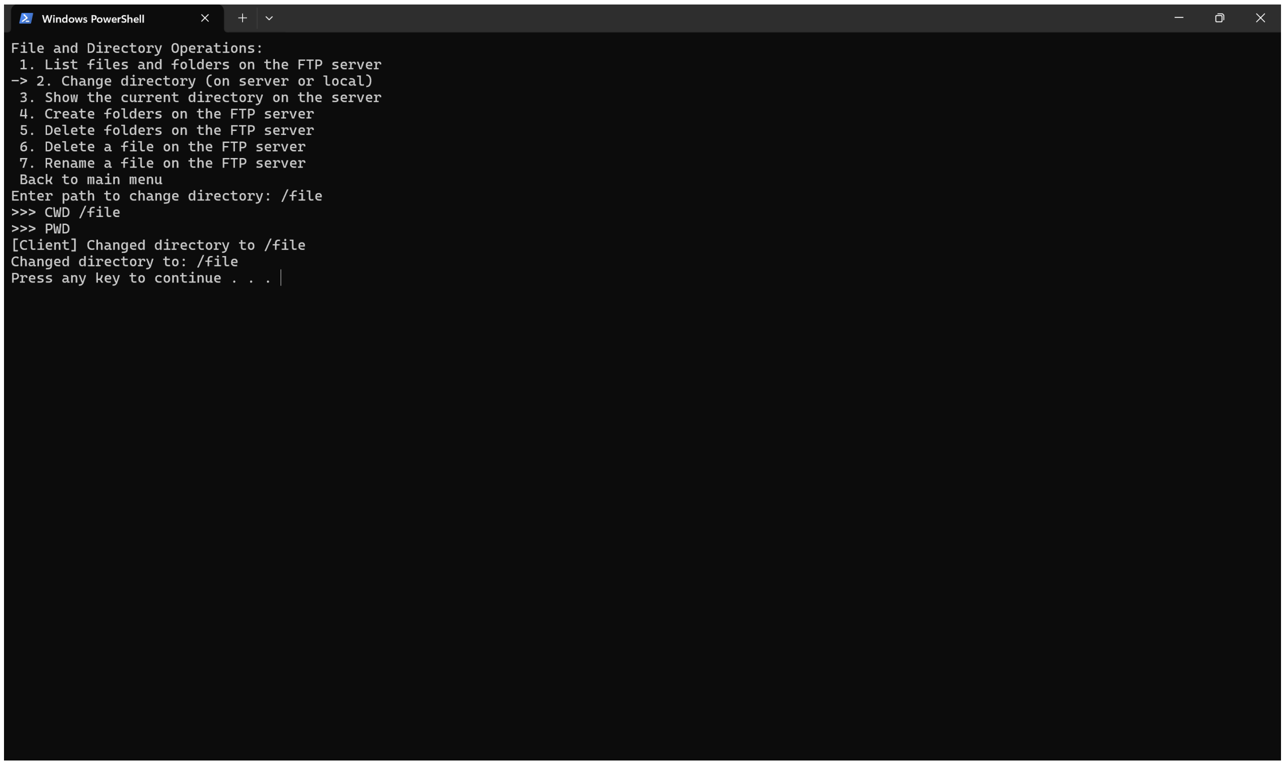
```
FTP Client Menu:
-> 1. File and Directory Operations
 2. Upload/Download
 3. Session Management
 Exit
```

Figure 2: Displaying the menu for user to choose



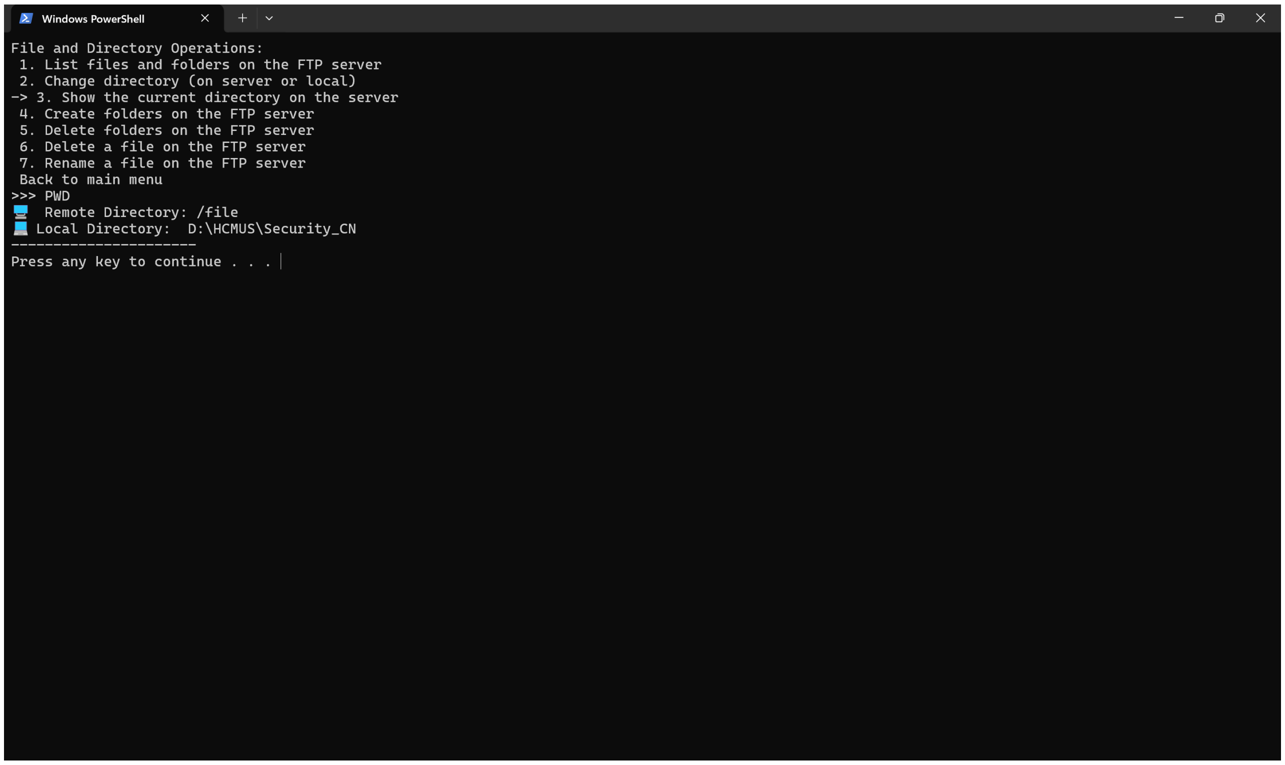
```
File and Directory Operations:
-> 1. List files and folders on the FTP server
 2. Change directory (on server or local)
 3. Show the current directory on the server
 4. Create folders on the FTP server
 5. Delete folders on the FTP server
 6. Delete a file on the FTP server
 7. Rename a file on the FTP server
 Back to main menu
Enter path to list files: My files
>>> PWD
>>> PORT 127,0,0,1,199,36
>>> LIST My files
Type         Size  Last Modified      Name
----         ----  -------------      ----
           1.9 KB  Aug 06 09:16       agent.py
          28.9 KB  Aug 07 13:33       client.py
Press any key to continue . . .
```

Figure 3: Using the `ls` command to list files and folders on the FTP server.



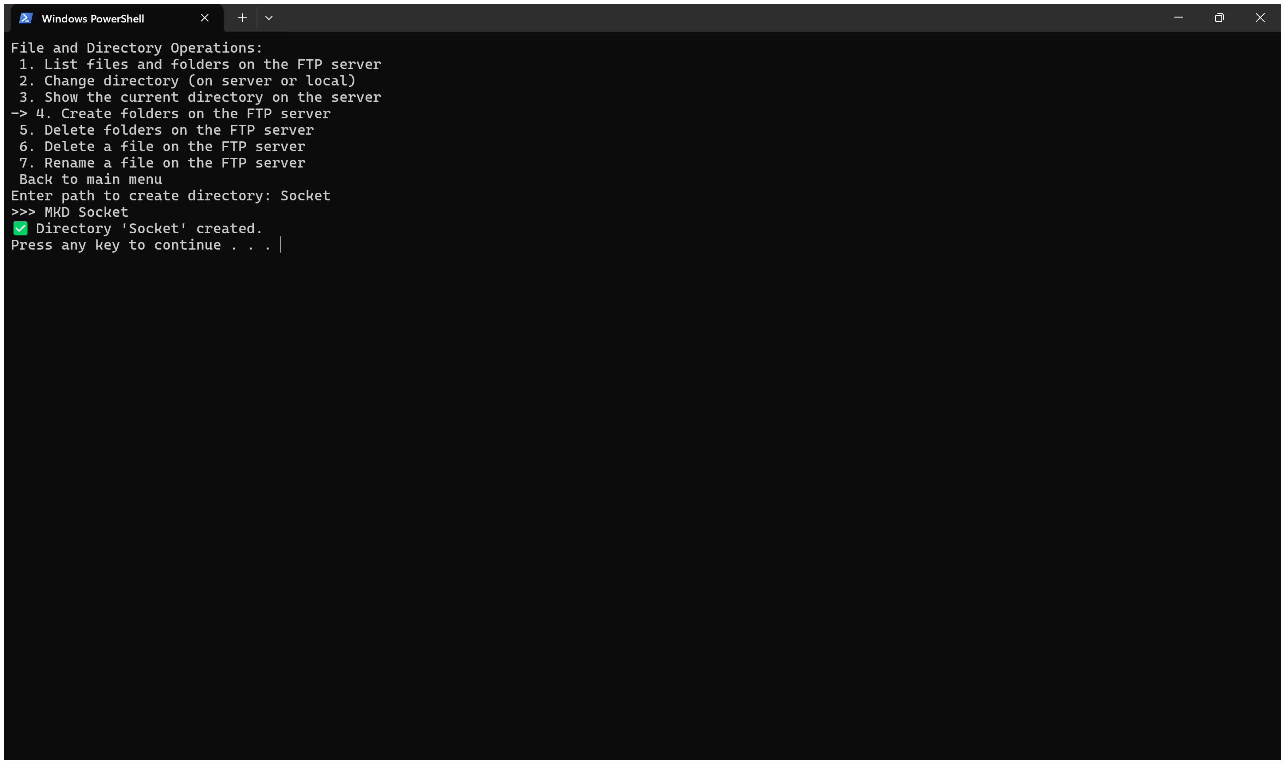
```
File and Directory Operations:
 1. List files and folders on the FTP server
-> 2. Change directory (on server or local)
 3. Show the current directory on the server
 4. Create folders on the FTP server
 5. Delete folders on the FTP server
 6. Delete a file on the FTP server
 7. Rename a file on the FTP server
 Back to main menu
Enter path to change directory: /file
>>> CWD /file
>>> PWD
[Client] Changed directory to /file
Changed directory to: /file
Press any key to continue . . .
```

Figure 4: Changing directory on the FTP server with the `cd` command.



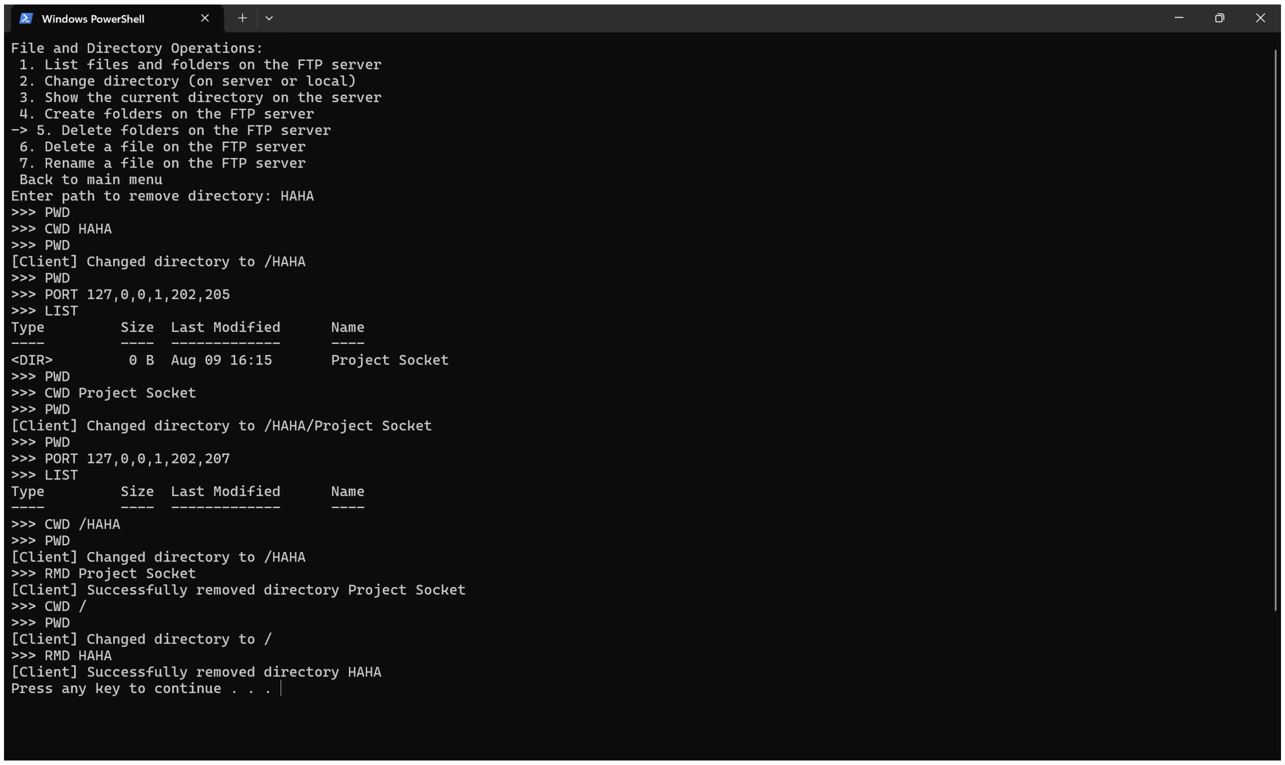
```
File and Directory Operations:
 1. List files and folders on the FTP server
 2. Change directory (on server or local)
-> 3. Show the current directory on the server
 4. Create folders on the FTP server
 5. Delete folders on the FTP server
 6. Delete a file on the FTP server
 7. Rename a file on the FTP server
 Back to main menu
>>> PWD
🖥  Remote Directory: /file
💻 Local Directory:  D:\HCMUS\Security_CN
----------------------
Press any key to continue . . .
```

Figure 5: Displaying the current working directory on the FTP server using `pwd`.



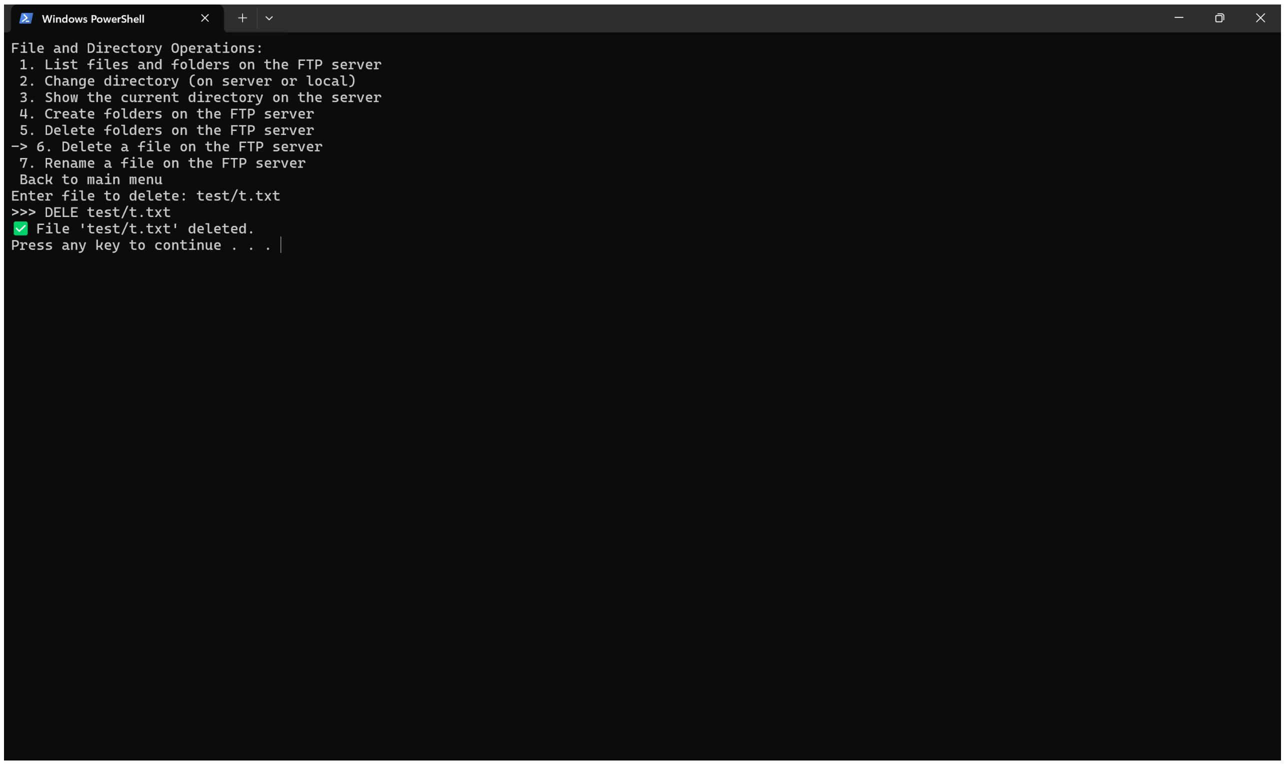
```
File and Directory Operations:
 1. List files and folders on the FTP server
 2. Change directory (on server or local)
 3. Show the current directory on the server
-> 4. Create folders on the FTP server
 5. Delete folders on the FTP server
 6. Delete a file on the FTP server
 7. Rename a file on the FTP server
 Back to main menu
Enter path to create directory: Socket
>>> MKD Socket
✅ Directory 'Socket' created.
Press any key to continue . . .
```

Figure 6: Creating a directory on the FTP server with `mkdir`.



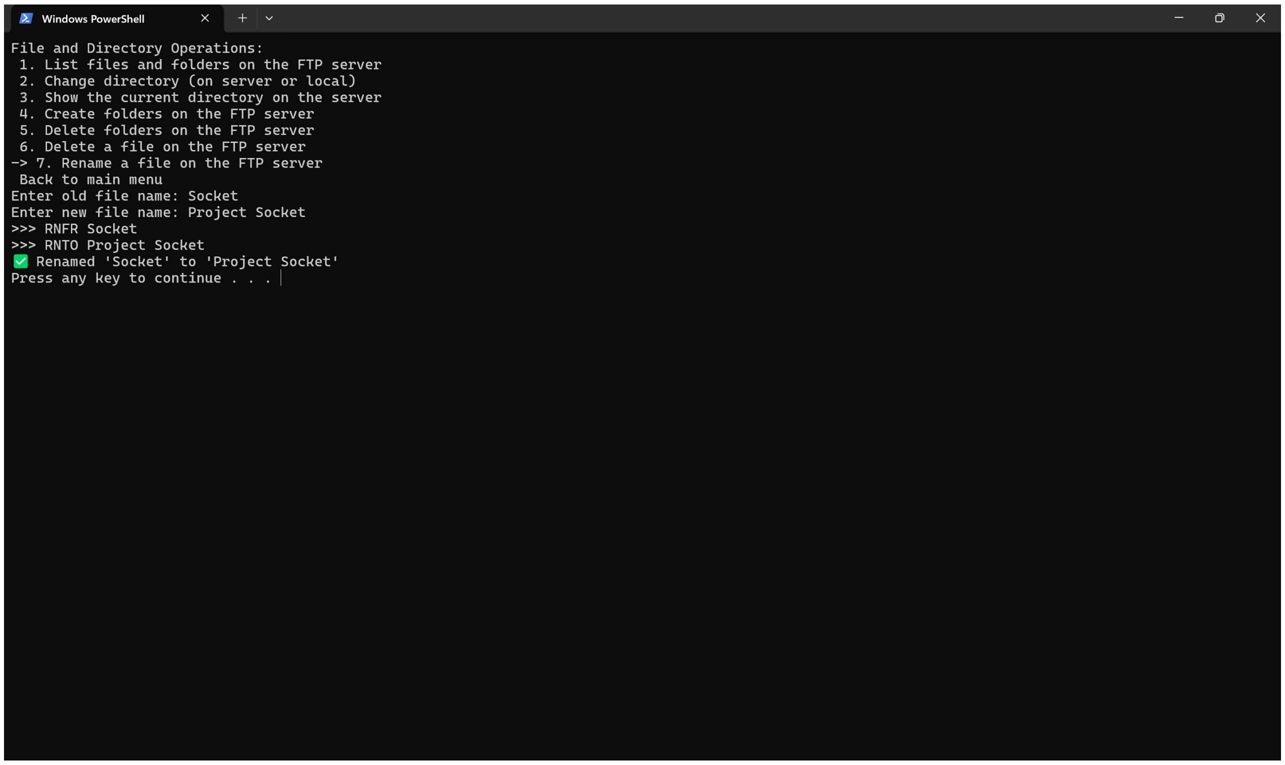
```
File and Directory Operations:
 1. List files and folders on the FTP server
 2. Change directory (on server or local)
 3. Show the current directory on the server
 4. Create folders on the FTP server
-> 5. Delete folders on the FTP server
 6. Delete a file on the FTP server
 7. Rename a file on the FTP server
 Back to main menu
Enter path to remove directory: HAHA
>>> PWD
>>> CWD HAHA
>>> PWD
[Client] Changed directory to /HAHA
>>> PWD
>>> PORT 127,0,0,1,202,205
>>> LIST
Type         Size  Last Modified      Name
----         ----  -------------      ----
<DIR>         0 B  Aug 09 16:15       Project Socket
>>> PWD
>>> CWD Project Socket
>>> PWD
[Client] Changed directory to /HAHA/Project Socket
>>> PWD
>>> PORT 127,0,0,1,202,207
>>> LIST
Type         Size  Last Modified      Name
----         ----  -------------      ----
>>> CWD /HAHA
>>> PWD
[Client] Changed directory to /HAHA
>>> RMD Project Socket
[Client] Successfully removed directory Project Socket
>>> CWD /
>>> PWD
[Client] Changed directory to /
>>> RMD HAHA
[Client] Successfully removed directory HAHA
Press any key to continue . . .
```

Figure 7: Removing a directory on the FTP server with `rmdir`.



```
File and Directory Operations:
 1. List files and folders on the FTP server
 2. Change directory (on server or local)
 3. Show the current directory on the server
 4. Create folders on the FTP server
 5. Delete folders on the FTP server
-> 6. Delete a file on the FTP server
 7. Rename a file on the FTP server
 Back to main menu
Enter file to delete: test/t.txt
>>> DELE test/t.txt
✅ File 'test/t.txt' deleted.
Press any key to continue . . .
```

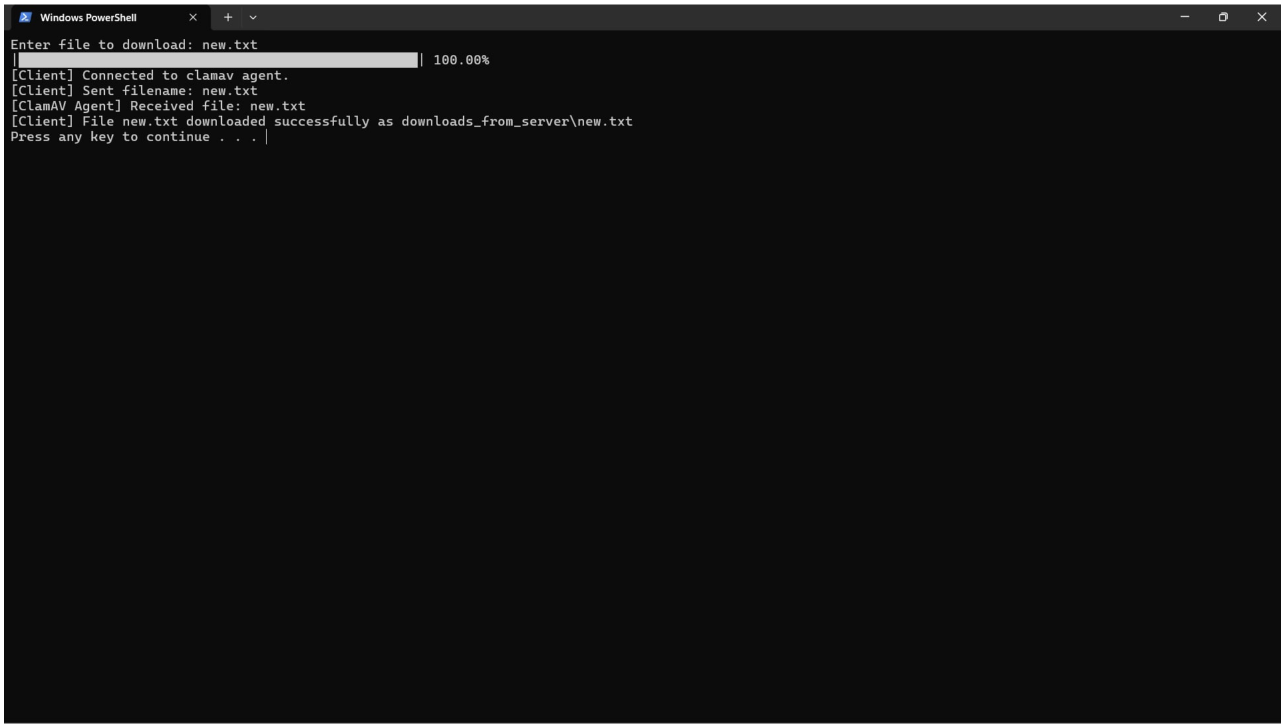
Figure 8: Deleting a file from the FTP server using the `delete` command.



```
File and Directory Operations:
 1. List files and folders on the FTP server
 2. Change directory (on server or local)
 3. Show the current directory on the server
 4. Create folders on the FTP server
 5. Delete folders on the FTP server
 6. Delete a file on the FTP server
-> 7. Rename a file on the FTP server
 Back to main menu
Enter old file name: Socket
Enter new file name: Project Socket
>>> RNFR Socket
>>> RNTO Project Socket
✅ Renamed 'Socket' to 'Project Socket'
Press any key to continue . . .
```

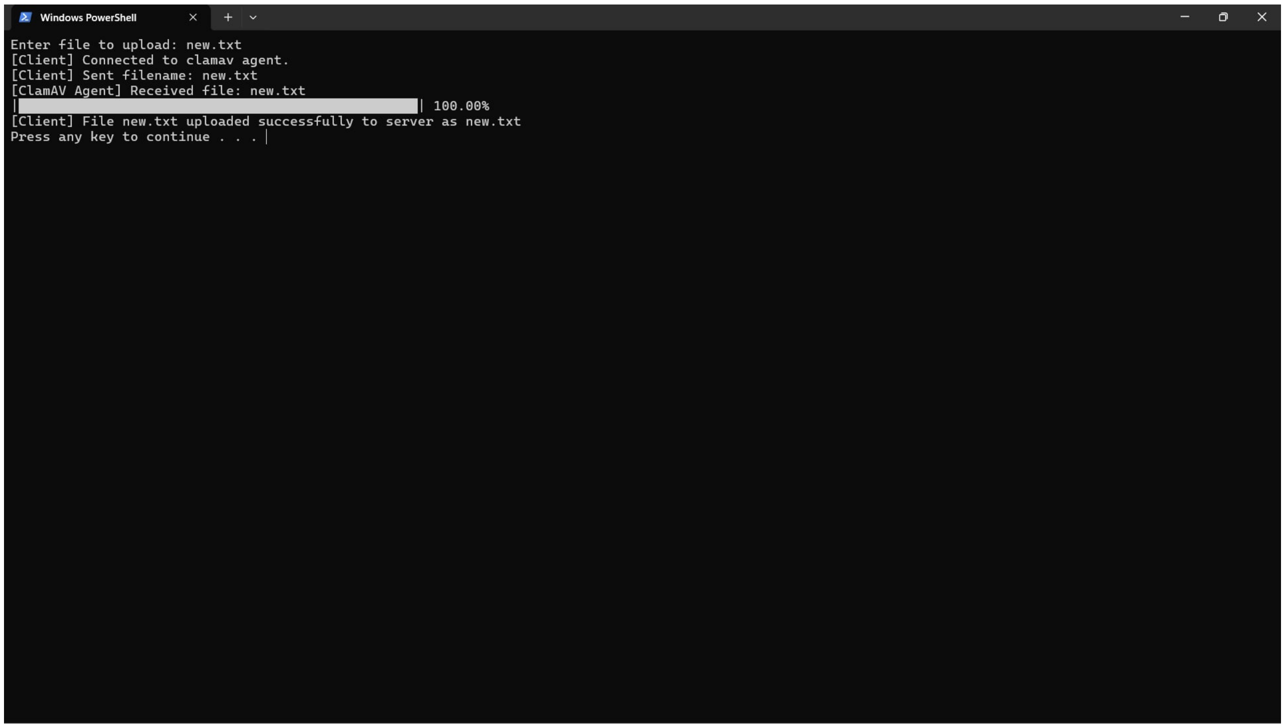
Figure 9: Renaming a file on the FTP server using the `rename` command.

2. Upload and Download



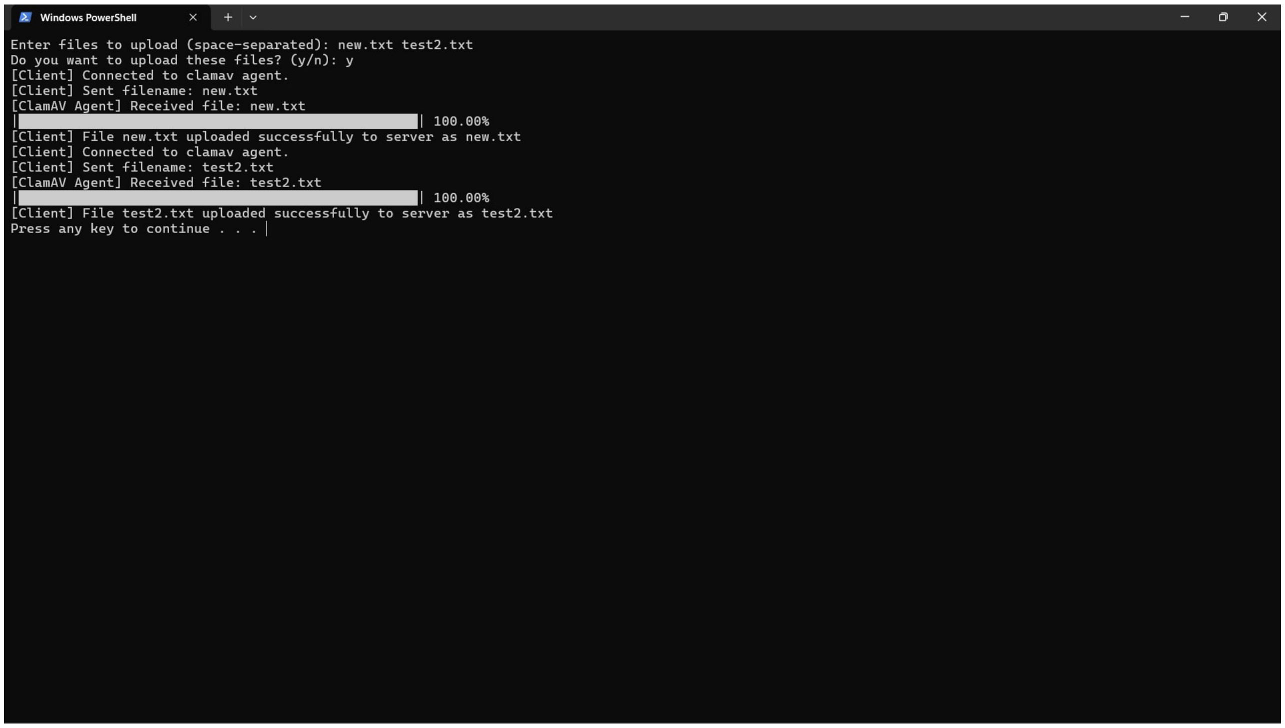
```
Enter file to download: new.txt
|██████████████████████████████████████████████| 100.00%
[Client] Connected to clamav agent.
[Client] Sent filename: new.txt
[ClamAV Agent] Received file: new.txt
[Client] File new.txt downloaded successfully as downloads_from_server\new.txt
Press any key to continue . . .
```

Figure 10: Downloading a file from the FTP server using `get` / `recv`.



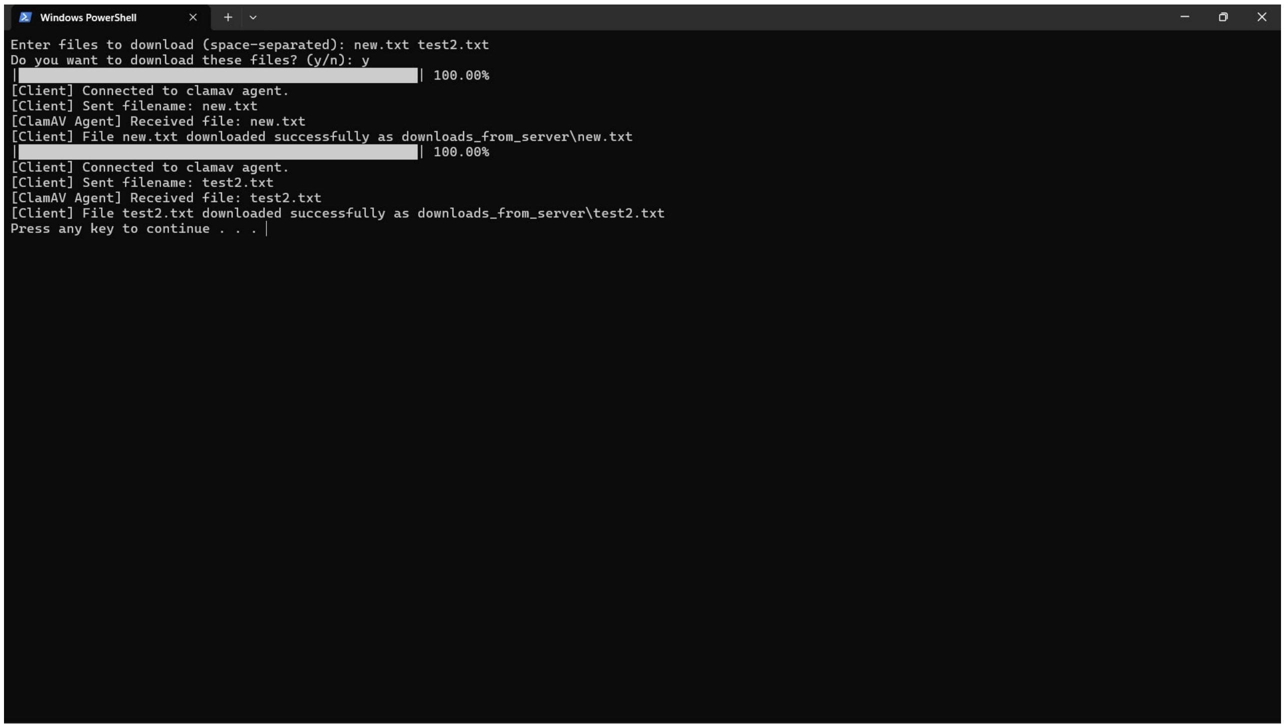
```
Enter file to upload: new.txt
[Client] Connected to clamav agent.
[Client] Sent filename: new.txt
[ClamAV Agent] Received file: new.txt
|██████████████████████████████████████████████████| 100.00%
[Client] File new.txt uploaded successfully to server as new.txt
Press any key to continue . . .
```

Figure 11: Uploading a single file after ClamAV scan using `put`.



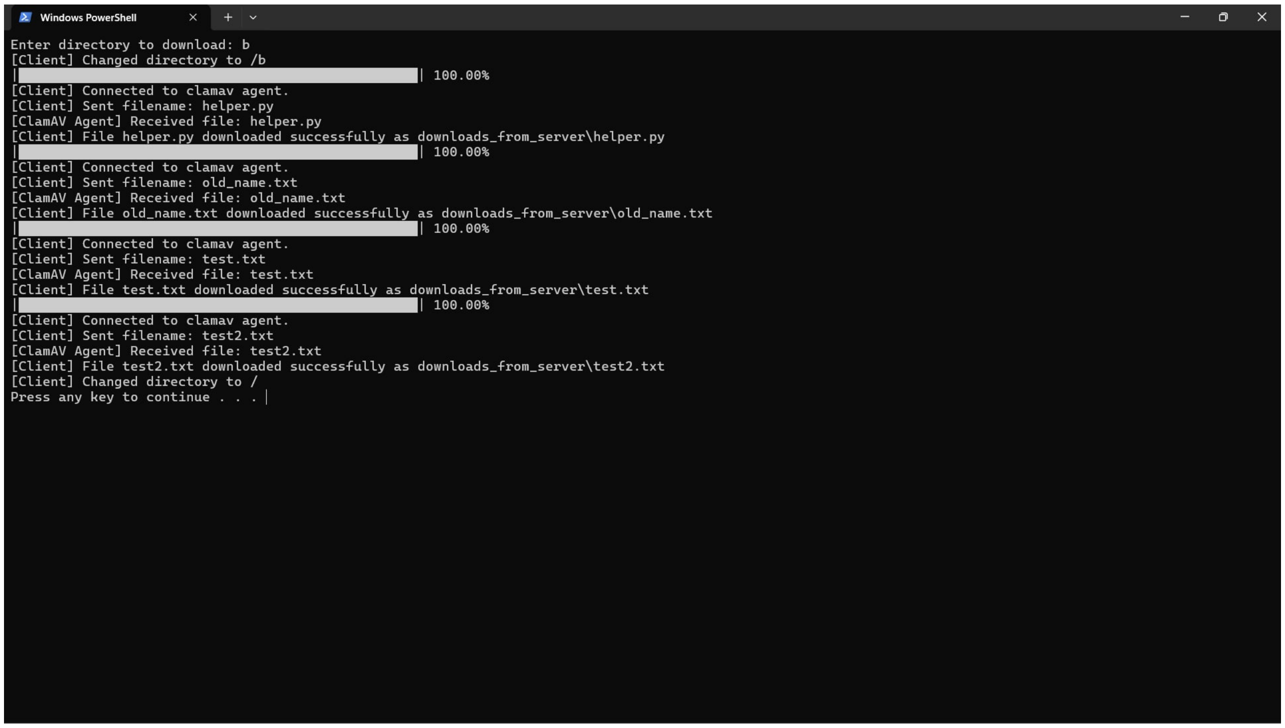
```
Enter files to upload (space-separated): new.txt test2.txt
Do you want to upload these files? (y/n): y
[Client] Connected to clamav agent.
[Client] Sent filename: new.txt
[ClamAV Agent] Received file: new.txt
|██████████████████████████████████████████████████| 100.00%
[Client] File new.txt uploaded successfully to server as new.txt
[Client] Connected to clamav agent.
[Client] Sent filename: test2.txt
[ClamAV Agent] Received file: test2.txt
|██████████████████████████████████████████████████| 100.00%
[Client] File test2.txt uploaded successfully to server as test2.txt
Press any key to continue . . .
```

Figure 12: Uploading multiple files with `mput` (all scanned before upload).



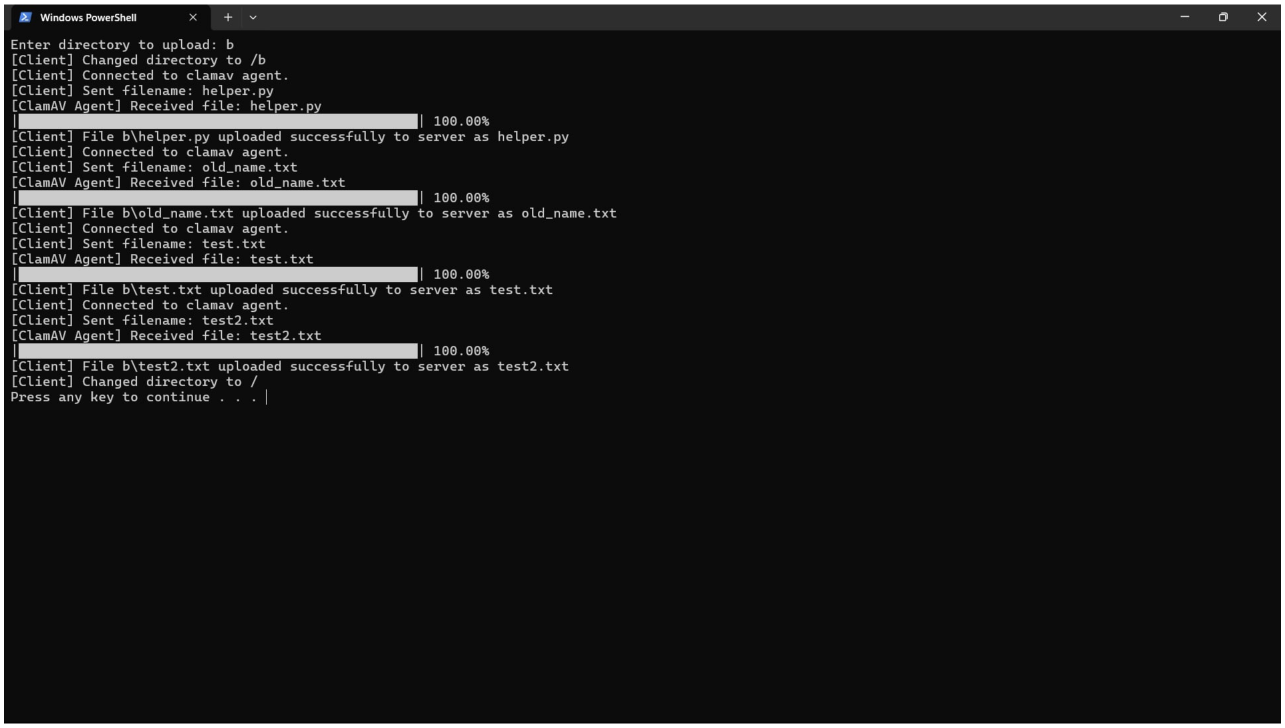
```
Enter files to download (space-separated): new.txt test2.txt
Do you want to download these files? (y/n): y
|██████████████████████████████████████████████████| 100.00%
[Client] Connected to clamav agent.
[Client] Sent filename: new.txt
[ClamAV Agent] Received file: new.txt
[Client] File new.txt downloaded successfully as downloads_from_server\new.txt
|██████████████████████████████████████████████████| 100.00%
[Client] Connected to clamav agent.
[Client] Sent filename: test2.txt
[ClamAV Agent] Received file: test2.txt
[Client] File test2.txt downloaded successfully as downloads_from_server\test2.txt
Press any key to continue . . .
```

Figure 13: Downloading multiple files with `mget`.



```
Enter directory to download: b
[Client] Changed directory to /b
|██████████████████████████████████████████████████| 100.00%
[Client] Connected to clamav agent.
[Client] Sent filename: helper.py
[ClamAV Agent] Received file: helper.py
[Client] File helper.py downloaded successfully as downloads_from_server\helper.py
|██████████████████████████████████████████████████| 100.00%
[Client] Connected to clamav agent.
[Client] Sent filename: old_name.txt
[ClamAV Agent] Received file: old_name.txt
[Client] File old_name.txt downloaded successfully as downloads_from_server\old_name.txt
|██████████████████████████████████████████████████| 100.00%
[Client] Connected to clamav agent.
[Client] Sent filename: test.txt
[ClamAV Agent] Received file: test.txt
[Client] File test.txt downloaded successfully as downloads_from_server\test.txt
|██████████████████████████████████████████████████| 100.00%
[Client] Connected to clamav agent.
[Client] Sent filename: test2.txt
[ClamAV Agent] Received file: test2.txt
[Client] File test2.txt downloaded successfully as downloads_from_server\test2.txt
[Client] Changed directory to /
Press any key to continue . . .
```

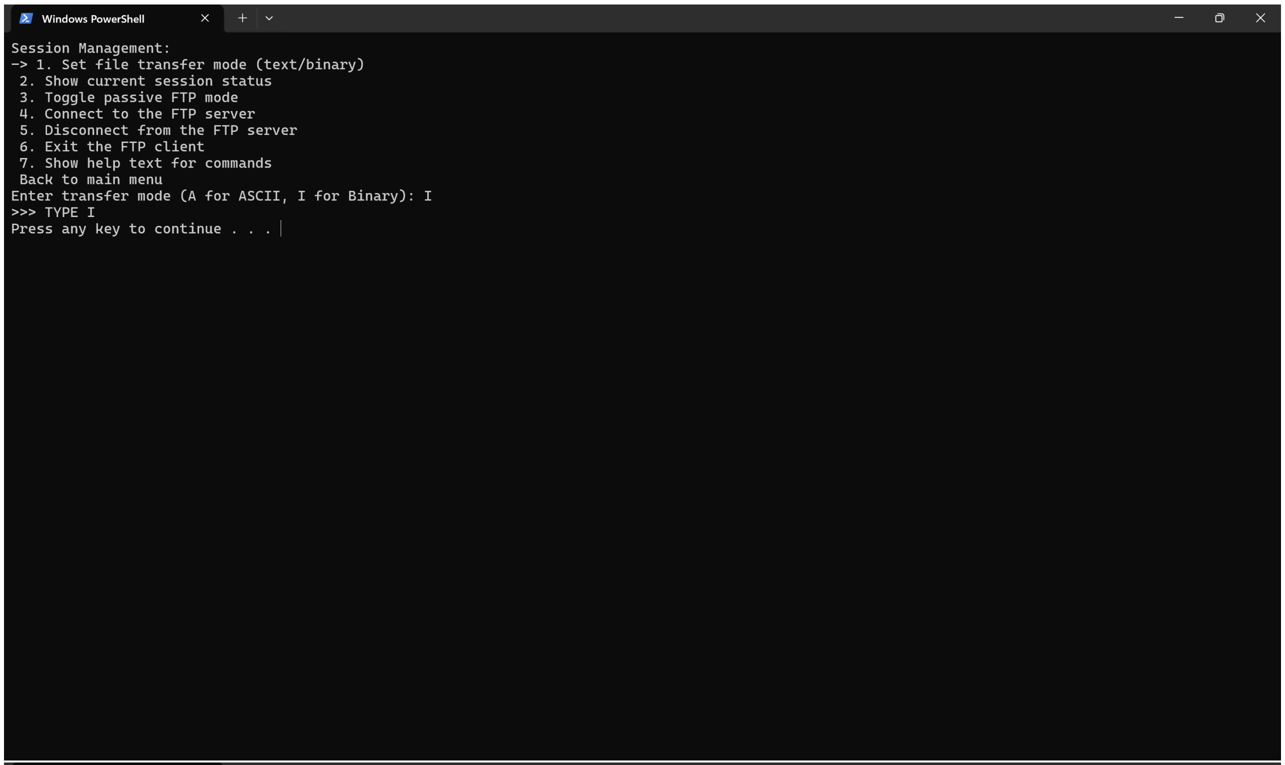
Figure 14: Downloading folder using prompt.



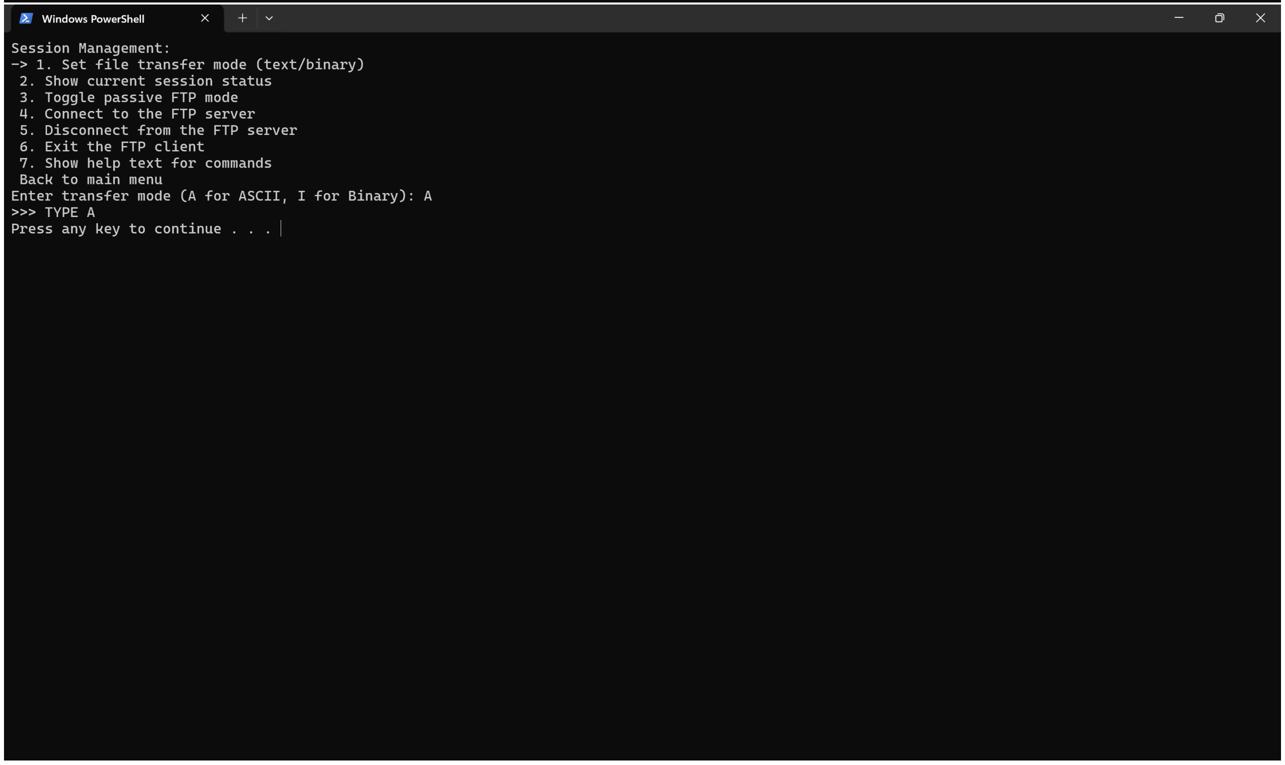
```
Enter directory to upload: b
[Client] Changed directory to /b
[Client] Connected to clamav agent.
[Client] Sent filename: helper.py
[ClamAV Agent] Received file: helper.py
|██████████████████████████████████████████████████| 100.00%
[Client] File b\helper.py uploaded successfully to server as helper.py
[Client] Connected to clamav agent.
[Client] Sent filename: old_name.txt
[ClamAV Agent] Received file: old_name.txt
|██████████████████████████████████████████████████| 100.00%
[Client] File b\old_name.txt uploaded successfully to server as old_name.txt
[Client] Connected to clamav agent.
[Client] Sent filename: test.txt
[ClamAV Agent] Received file: test.txt
|██████████████████████████████████████████████████| 100.00%
[Client] File b\test.txt uploaded successfully to server as test.txt
[Client] Connected to clamav agent.
[Client] Sent filename: test2.txt
[ClamAV Agent] Received file: test2.txt
|██████████████████████████████████████████████████| 100.00%
[Client] File b\test2.txt uploaded successfully to server as test2.txt
[Client] Changed directory to /
Press any key to continue . . .
```

Figure 15: Uploading folder using prompt (all scanned before upload).

3. Session Management

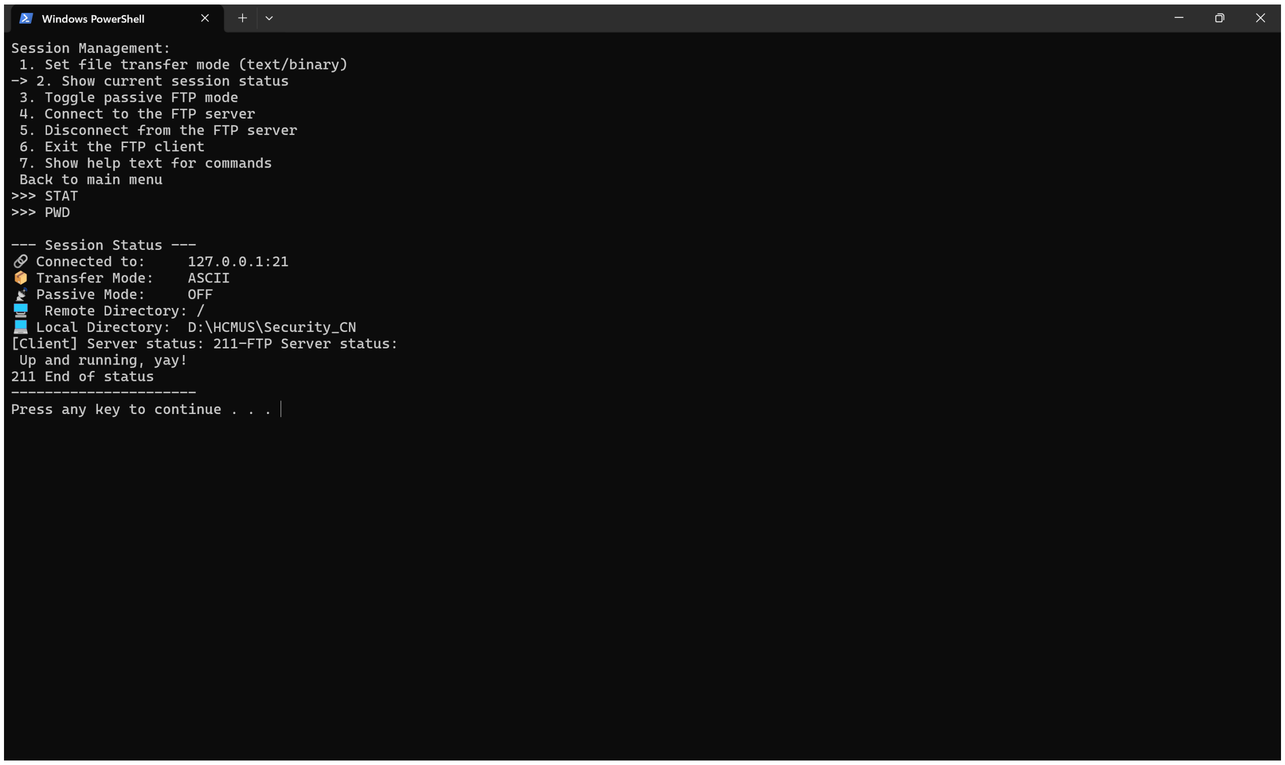


```
Session Management:
-> 1. Set file transfer mode (text/binary)
 2. Show current session status
 3. Toggle passive FTP mode
 4. Connect to the FTP server
 5. Disconnect from the FTP server
 6. Exit the FTP client
 7. Show help text for commands
 Back to main menu
Enter transfer mode (A for ASCII, I for Binary): I
>>> TYPE I
Press any key to continue . . .
```



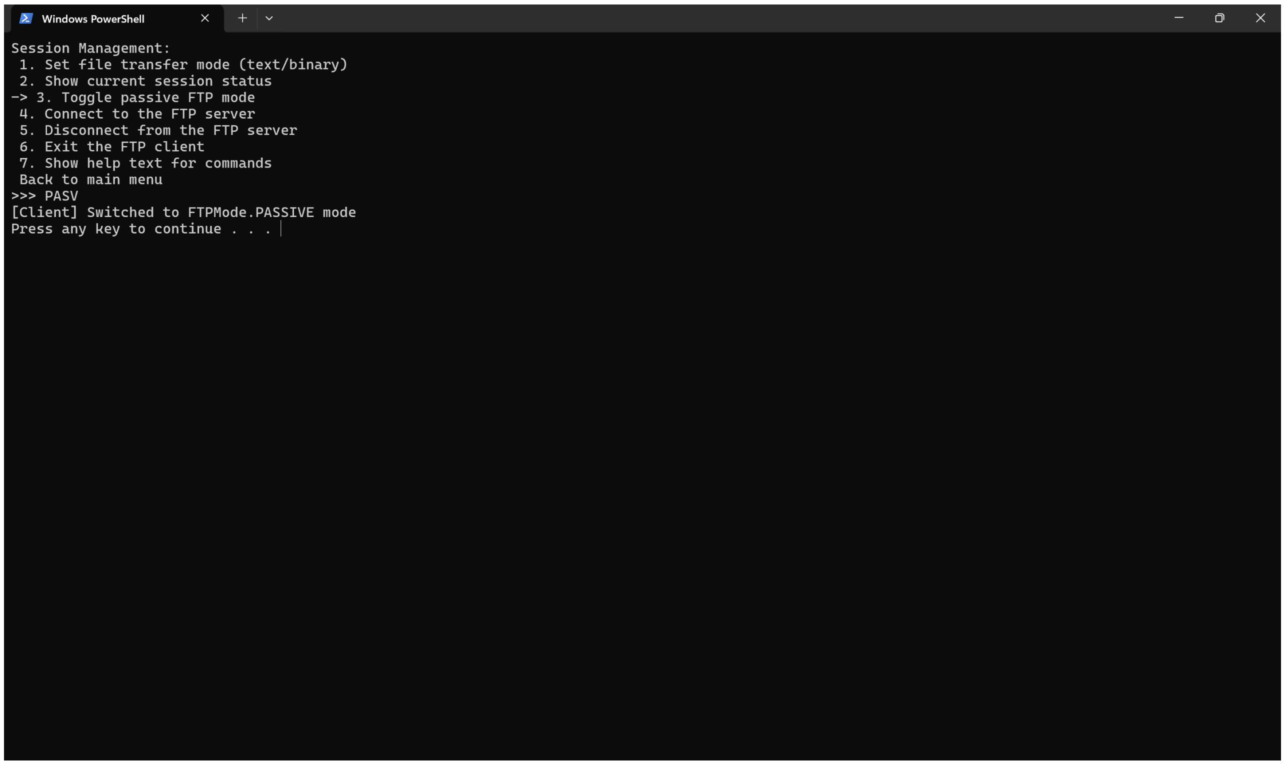
```
Session Management:
-> 1. Set file transfer mode (text/binary)
 2. Show current session status
 3. Toggle passive FTP mode
 4. Connect to the FTP server
 5. Disconnect from the FTP server
 6. Exit the FTP client
 7. Show help text for commands
 Back to main menu
Enter transfer mode (A for ASCII, I for Binary): A
>>> TYPE A
Press any key to continue . . .
```

Figure 16: Switching between ASCII and Binary transfer mode using `ascii` / `binary`.



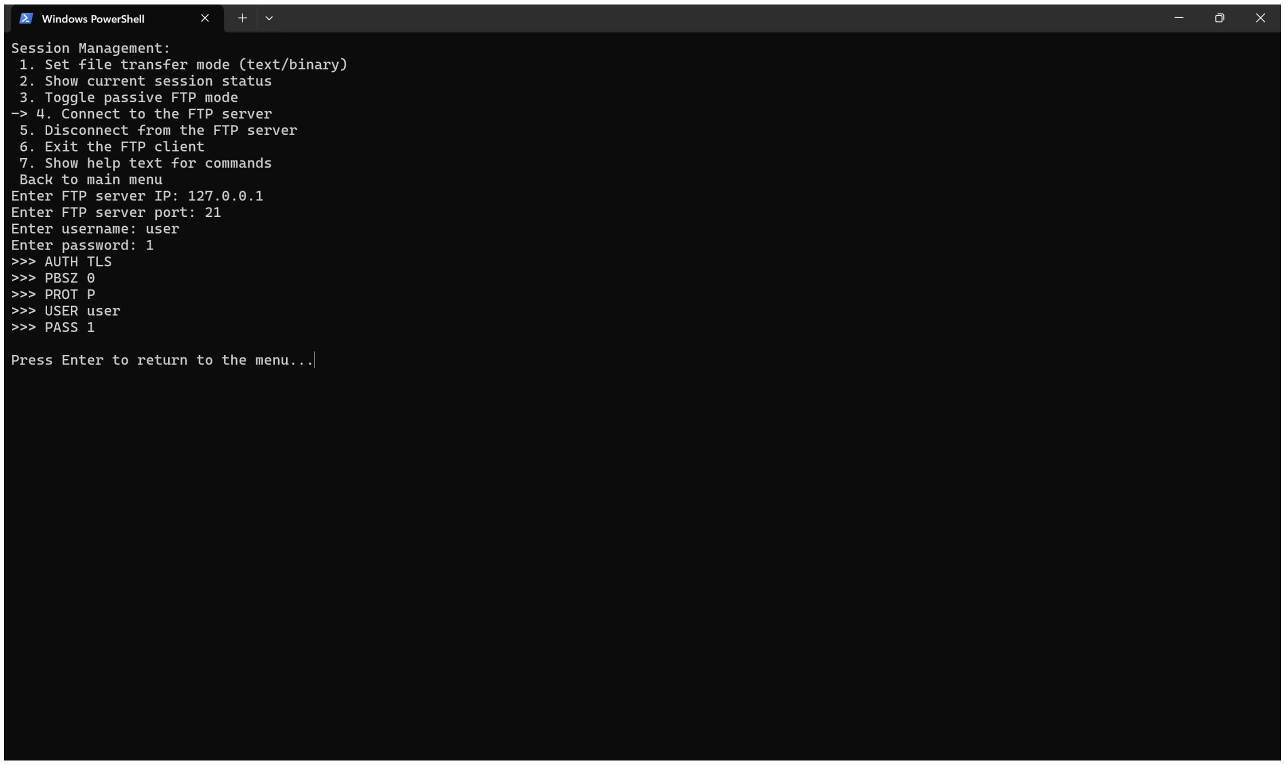
```
Session Management:
 1. Set file transfer mode (text/binary)
-> 2. Show current session status
 3. Toggle passive FTP mode
 4. Connect to the FTP server
 5. Disconnect from the FTP server
 6. Exit the FTP client
 7. Show help text for commands
 Back to main menu
>>> STAT
>>> PWD

--- Session Status ---
🔗 Connected to:     127.0.0.1:21
📦 Transfer Mode:    ASCII
📡 Passive Mode:     OFF
💻  Remote Directory: /
💻 Local Directory:  D:\HCMUS\Security_CN
[Client] Server status: 211-FTP Server status:
 Up and running, yay!
211 End of status
----------------------
Press any key to continue . . .
```

Figure 17: Displaying the current FTP session status with `status`.



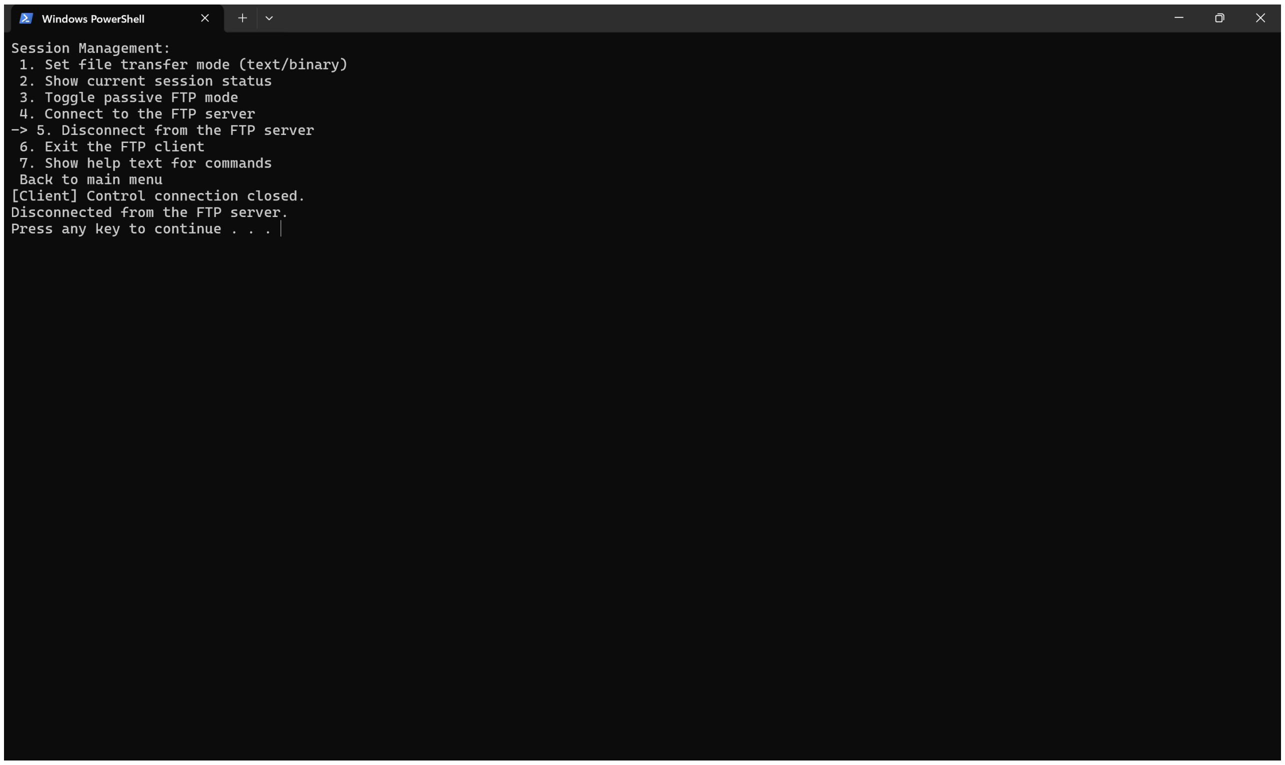
```
Session Management:
 1. Set file transfer mode (text/binary)
 2. Show current session status
-> 3. Toggle passive FTP mode
 4. Connect to the FTP server
 5. Disconnect from the FTP server
 6. Exit the FTP client
 7. Show help text for commands
 Back to main menu
>>> PASV
[Client] Switched to FTPMode.PASSIVE mode
Press any key to continue . . .
```

Figure 18: Toggling passive FTP mode with `passive`.



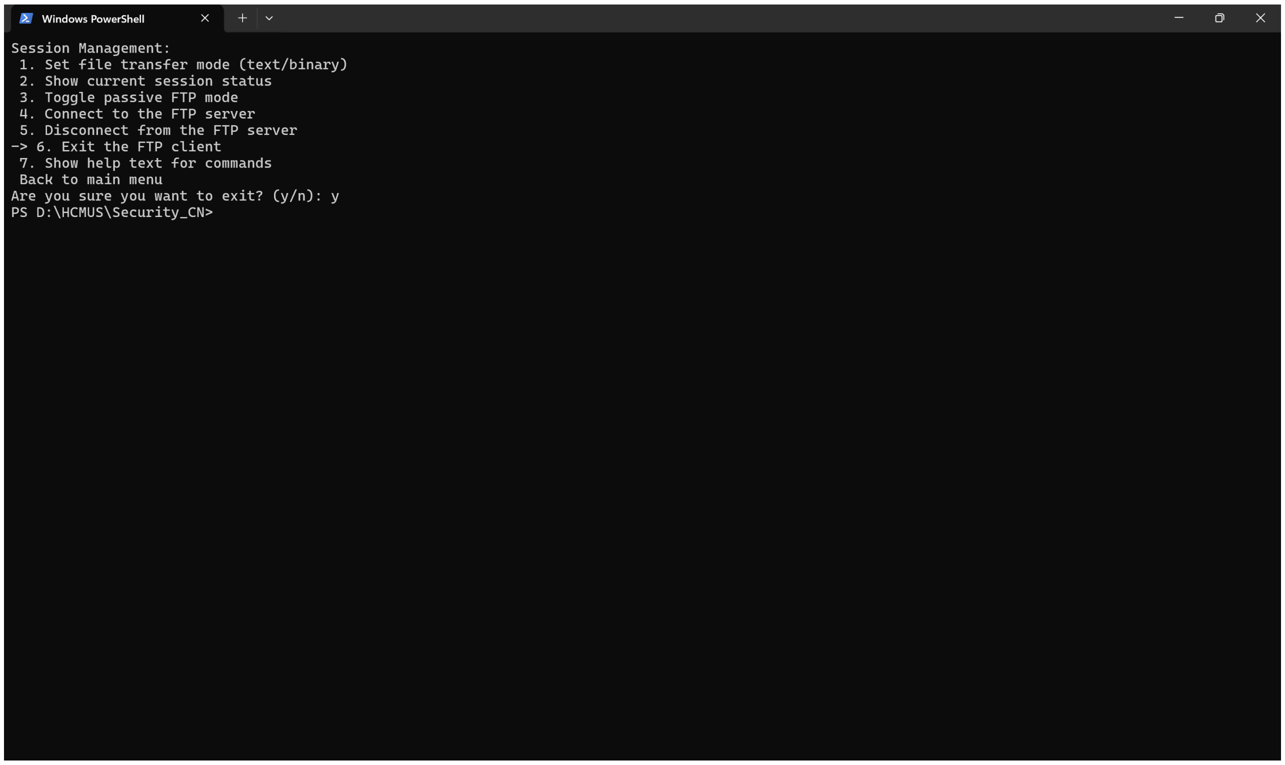
```
Session Management:
 1. Set file transfer mode (text/binary)
 2. Show current session status
 3. Toggle passive FTP mode
-> 4. Connect to the FTP server
 5. Disconnect from the FTP server
 6. Exit the FTP client
 7. Show help text for commands
 Back to main menu
Enter FTP server IP: 127.0.0.1
Enter FTP server port: 21
Enter username: user
Enter password: 1
>>> AUTH TLS
>>> PBSZ 0
>>> PROT P
>>> USER user
>>> PASS 1

Press Enter to return to the menu...
```

Figure 19: Connecting from the FTP server with `open`.



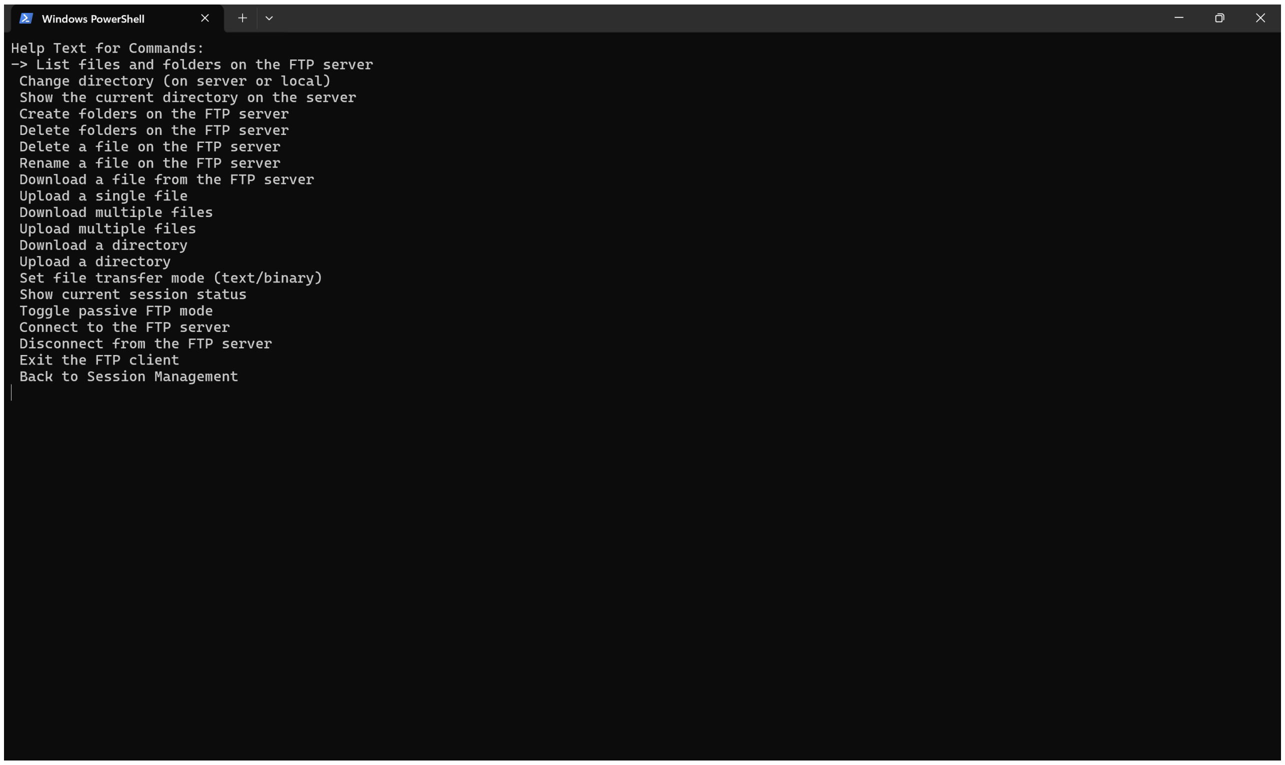
```
Session Management:
 1. Set file transfer mode (text/binary)
 2. Show current session status
 3. Toggle passive FTP mode
 4. Connect to the FTP server
-> 5. Disconnect from the FTP server
 6. Exit the FTP client
 7. Show help text for commands
 Back to main menu
[Client] Control connection closed.
Disconnected from the FTP server.
Press any key to continue . . .
```

Figure 20: Disconnecting from the FTP server with `close`.



```
Session Management:
 1. Set file transfer mode (text/binary)
 2. Show current session status
 3. Toggle passive FTP mode
 4. Connect to the FTP server
 5. Disconnect from the FTP server
-> 6. Exit the FTP client
 7. Show help text for commands
 Back to main menu
Are you sure you want to exit? (y/n): y
PS D:\HCMUS\Security_CN>
```

Figure 21: Exiting the FTP client using `quit` / `bye`.



```
Help Text for Commands:
-> List files and folders on the FTP server
 Change directory (on server or local)
 Show the current directory on the server
 Create folders on the FTP server
 Delete folders on the FTP server
 Delete a file on the FTP server
 Rename a file on the FTP server
 Download a file from the FTP server
 Upload a single file
 Download multiple files
 Upload multiple files
 Download a directory
 Upload a directory
 Set file transfer mode (text/binary)
 Show current session status
 Toggle passive FTP mode
 Connect to the FTP server
 Disconnect from the FTP server
 Exit the FTP client
 Back to Session Management
```

Figure 22: Displaying the help menu with `help` / `?`.

4 Problems Encountered and Solutions

During the development and testing of the secure FTP client-server system with ClamAV integration, several technical issues were encountered. The following subsections describe each problem in detail and the corresponding solution implemented.

4.1 Authentication and TLS Encryption Failures

Problem: Initial connections to the FTP server sometimes failed during authentication, particularly when TLS encryption was enabled. The issue occurred during the `AUTH TLS` negotiation phase or when incorrect credentials were provided, resulting in connection termination. This is handled in the `create_control_socket()` function in `connection.py`, where multiple server responses (e.g., `234`, `331`, `230`) must be validated in sequence.

Solution: Implemented SSL/TLS encryption using Python's `ssl` module. The client explicitly sends `AUTH TLS` and upgrades the control socket to an encrypted channel if `ftpconfig.use_ssl` is enabled. All subsequent FTP commands and data channel connections are secured, preventing credentials and file contents from being transmitted in plain text. Error handling ensures clear messages when authentication fails.

4.2 Passive Mode Data Channel Failures due to Firewall

Problem: In passive mode (implemented in `create_data_socket_passive()` in `connection.py`), the FTP server opens a dynamic port for each data transfer. In some environments, firewall rules blocked these ephemeral ports, leading to failed file transfers or directory listings.

Solution: Configured the FTP server (FileZilla) to use a fixed passive port range and opened these ports in the firewall configuration. This allowed the client to reliably establish passive mode data connections without random port failures.

4.3 File Corruption Due to Incorrect Transfer Mode

Problem: Binary files (e.g., images, executables) became corrupted when transferred in ASCII mode, as line-ending conversions altered the raw file content. The `transfer_ascii_binary_mode()` function in `command.py` allows switching between ASCII and binary mode, but incorrect mode selection caused this issue.

Solution: Updated the client to default to binary mode (`TYPE I`) for all non-text files. Before starting a transfer, the file type is detected, and the appropriate mode command is sent to the server, ensuring file integrity.

4.4 Antivirus Scanning Delays for Large Files

Problem: The `scan_for_virus()` function in `connection.py` sends files to the ClamAV agent for scanning before uploads and after downloads. Large files caused significant scanning delays, sometimes triggering the client's socket timeout (`ftpconfig.timeout`).

Solution: Increased the socket timeout to accommodate longer scan durations. Additionally, optimized ClamAV scanning by excluding temporary files from repeated scans and improving the file transfer chunk size for faster delivery to the ClamAV agent.

4.5 Socket Timeout and Connection Resets During Transfers

Problem: Long-running uploads or downloads occasionally failed due to socket timeouts or unexpected connection resets. This occurred in both active and passive modes, as seen in `create_data_socket_active()` and `create_data_socket_passiv`

Solution: Extended the `ftpconfig.timeout` value and added better exception handling to close sockets cleanly when errors occur. Implemented keep-alive mechanisms (e.g., sending `NOOP` commands during long transfers) to prevent idle disconnections.

5 Summary of How Each Requirement Was Fulfilled

The following table summarizes each project requirement and explains how it was implemented in the developed system. All implementations are based on the provided source files (`ftp_client.py`, `command.py`, `connection.py`, `clamav.py`, and `ftp_config.py`).

1. **Implement FTP Client Functionality** Implemented in `ftp_client.py` and `command.py`. The client supports standard FTP commands for:
   - File and directory operations (`ls`, `cd`, `pwd`, `mkdir`, `rmdir`, `delete`, `rename`).
   - Upload and download (`put`, `mput`, `get`, `mget`, directory transfers).
   - Session management (`open`, `close`, `status`, `ascii`, `binary`, `passive`, `quit`).

   All FTP commands are sent through the control channel, and data transfer operations open temporary data channels in active or passive mode.

2. **Support for Active and Passive FTP Modes** Handled in `connection.py` through the functions `create_data_socket_a` and `create_data_socket_passive()`. The client can toggle between modes using the `transfer_passive_mode()` function in `command.py`. Passive mode parses the server's `227` response to establish the data channel.

3. **Secure Transmission using TLS/SSL** Implemented in `connection.py` within `create_control_socket()`. If `ftpconfig.use_ssl` is enabled, the client sends `AUTH TLS` and upgrades the socket using Python's `ssl` module. Both control and data channels can be encrypted, ensuring confidentiality of credentials and file contents.

4. **Virus Scanning for Uploads and Downloads** Implemented in `connection.py` (`scan_for_virus()`) and `clamav.py`. Before uploads (`put`, `mput`) and after downloads (`get`, `mget`), the client sends the file to the ClamAV Agent over a dedicated TCP socket for scanning. If the result is `INFECTED`, the transfer is aborted and the file is deleted.

5. **Configurable Parameters** Centralized in `ftp_config.py` via the `FTPConfig` dataclass. Parameters such as FTP host, port, ClamAV server address, buffer size, timeout, transfer mode, and SSL usage are easily adjustable.

6. **User Interaction and Menu Navigation** Provided in `ftp_client.py` through a console-based menu system with keyboard navigation. The menu groups commands into three categories: File/Directory Operations, Upload/Download, and Session Management. Interactive prompts and confirmations ensure a user-friendly experience.

7. **Error Handling and Status Reporting** Implemented throughout `command.py` and `connection.py`. Each operation validates server responses (e.g., `226` for successful transfers) and prints detailed error messages on failure. The `status()` function retrieves server status, and timeouts or socket errors are handled gracefully.

8. **Support for Recursive Directory Transfer** Implemented in `command.py` via `directory_put()` and `directory_get()`. These functions recursively upload or download entire directory structures, creating or navigating into remote directories as needed.