# **Secure FTP Client with Virus Scanning via ClamAVAgent**

## **A. Overview**

In this project, you will simulate a **real-world file transfer scenario** where files are scanned for viruses before being uploaded to a server. You will use **socket programming** to build communication between components and practice using the **FTP protocol** and **ClamAV antivirus engine**.

You will write **two programs** and work with an **FTP Server** setup:

1. A **custom FTP Client** to interact with an FTP Server and a ClamAV scanning service.
2. A **ClamAVAgent**, running on a separate machine, to receive files, scan them using **clamscan**, and return the result.

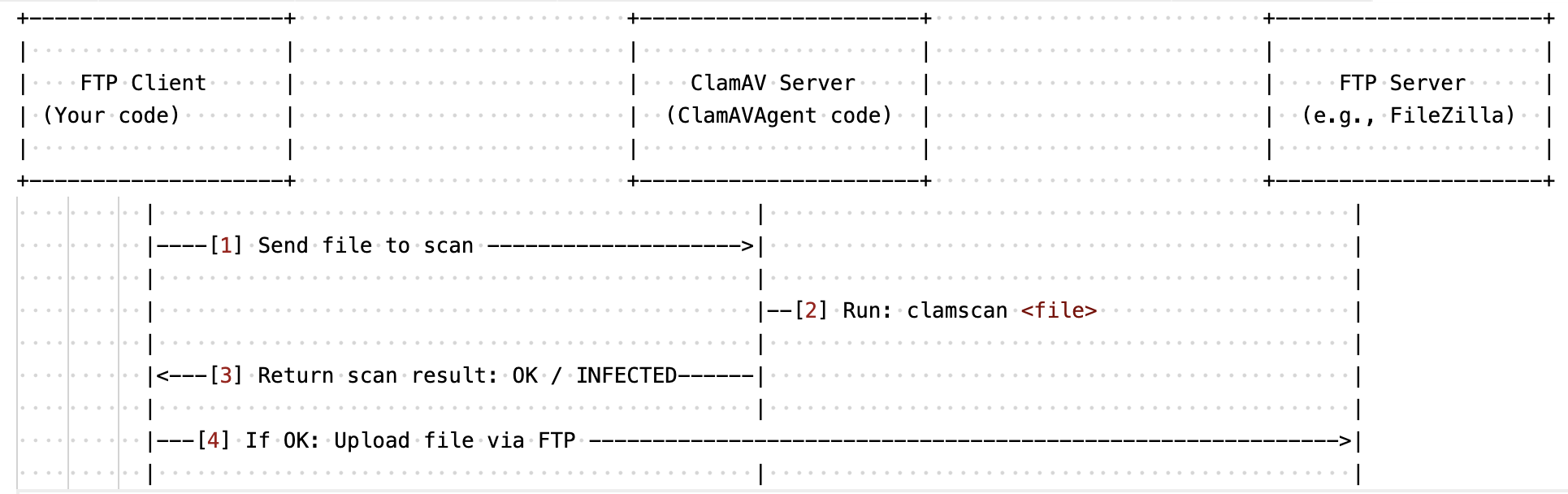
This lab will give you hands-on experience with:

* Client-server communication using sockets
* Protocols (FTP)
* File handling and virus scanning
* Command parsing and user interaction

## **B. What You Will Learn**

* How to implement a simple FTP-like client that interacts with servers
* How to implement a server program to receive and scan files using antivirus tools
* How to integrate socket communication between different machines
* How to parse commands and build a command-line interface
* How to transfer files securely in controlled environments

## **C. System Components & Setup**



You need to **simulate 3 machines** (can be 3 actual machines or 3 terminal windows using different ports/IPs):

### **1. FTP Client (Your Code) [1]**

* Runs your main client application.
* Accepts FTP-like commands (e.g., **ls, put, get**, etc.).
* For every file upload to the FTP server (put, mput), it **first sends the file to the ClamAVAgent** for virus scanning.
  + If the result is OK, then it uploads the file to the FTP Server.
  + If the result is INFECTED, it aborts the upload and shows a warning.

### **2. ClamAV Server (ClamAVAgent – Your Code)**

* Receives files from the FTP Client via a socket. [2]
* Runs virus scanning using (with the ClamAV software, which can be downloaded from <https://www.clamav.net/downloads>):  
   clamscan <file>
* Sends result (OK or INFECTED) back to the client.

### **3. FTP Server (Use any available software)**

* Receives file uploads from the client.
* Can be FileZilla Server, vsftpd, or any tool the student prefers.

## **D. Functional Requirements**

### **FTP Client Commands (MUST support)**

#### **1. File and Directory Operations**

|  |  |
| --- | --- |
| **Command** | **Description** |
| ls | List files and folders on the FTP server |
| cd | Change directory (on server or local) |
| pwd | Show the current directory on the server |
| mkdir, rmdir | Create or delete folders on the FTP server |
| delete | Delete a file on the FTP server |
| rename | Rename a file on the FTP server |

#### **2. Upload and Download**

|  |  |
| --- | --- |
| **Command** | **Description** |
| get, recv | Download a file from the FTP server |
| put | Upload a single file (must be scanned by ClamAVAgent before FTP upload) |
| mput | Upload multiple files (wildcard supported, all must be scanned first) |
| mget | Download multiple files |
| prompt | Toggle confirmation for mget / mput operations |

#### **3. Session Management**

|  |  |
| --- | --- |
| **Command** | **Description** |
| ascii / binary | Set file transfer mode (text/binary) |
| status | Show current session status |
| passive | Toggle passive FTP mode |
| open, close | Connect/disconnect to the FTP server |
| quit, bye | Exit the FTP client |
| help, ? | Show help text for commands |

## **E. Deliverables**

### **1. Source Code**

* **ftp\_client.py** (or ftp\_client.cpp, etc.)
* **clamav\_agent.py** (or other language)
* Comments and **documentation** are required.
* File transfer must use sockets (not system copy).

### **2. README File**

Include the following:

* Instructions to run the programs
* Sample commands and expected outputs
* FTP Server software used and how it was set up
* ClamAV installation and configuration

### **3. Report (PDF or Markdown)**

* Overview of your system design
* Diagrams (architecture)
* Screenshots of a successful session
* Problems encountered and how you solved them
* Summary of how each requirement was fulfilled

## **F. Testing Checklist**

Before submitting, make sure:

* You can list, rename, delete, and navigate files/folders on the FTP server.
* Upload is only allowed if the file is clean.
* ClamAVAgent correctly scans files and returns results.
* Wildcard (mput, mget) operations work.
* All commands behave as expected.
* All communication happens over sockets (not system shell calls except clamscan).

## **G. Grading Rubric (Total 10 Points)**

|  |  |  |
| --- | --- | --- |
| **No.** | **Requirement** | **Points** |
| 1 | Uploads go through virus scanning (ClamAVAgent works) | 2 |
| 2 | File & folder management commands (ls, cd, etc.) | 2 |
| 3 | Upload/download with put, get, mput, mget | 4 |
| 4 | Session control (open, status, quit, etc.) | 1.5 |
| 5 | Report & instructions | 0.5 |

## **H. Bonus Points (Up to 1 Points)**

|  |  |
| --- | --- |
| **Bonus Feature** | **Points** |
| **GUI,** **Progress Bar,** or real-time upload status in the client | +0.5 |
| Supports **recursive upload/download** for folders | +0.5 |

## **I. Group Requirement & Submission Guidelines**

* This exercise must be completed in a group of 3 students.
* Only **one student from the group** needs to submit the assignment.
* **Submit all files in a ZIP file** to the course platform before the deadline.