Secure File Transfer Protocol (SFTP) Project Documentation

Pau Santana

Project Name: Secure File Transfer Protocol (SFTP)

Date: 30/7/2023

Table of Contents

Introduction

The Secure File Transfer Protocol (SFTP) project aims to provide a secure and reliable file transfer mechanism between a client and a server. The primary purpose is to ensure data integrity and confidentiality during file transfer and manipulation.

Features

- Secure file transfer between client and server
- Client-side and server-side implementations
- File upload, download, listing, deletion, and renaming functionalities
- Data integrity verification using MD5 hash

Architecture/Design

The SFTP project follows a client-server architecture. The server listens for incoming client connections and handles file transfer and actions. Both client and server use socket-based communication for data exchange. Encryption and decryption techniques from the `cryptography` library are used to ensure data confidentiality. MD5 hashing is employed for data integrity verification.

Implementation

Client

```
# client/client.py

import os
import socket

# Function to establish a connection with the server


def connect_to_server():
client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
# Replace with the actual server address and port
server_address = ('localhost', 12345)
client_socket.connect(server_address)
return client_socket

# Function to send an action to the server


def send_action(client_socket, action):
try:
client_socket.send(action.encode())
except Exception as e:
print(f"Error while sending action: {str(e)}")

# Function to receive a response from the server


def receive_response(client_socket):
try:
```

```
response = client_socket.recv(1024).decode()
return response
except Exception as e:
print(f"Error while receiving response: {str(e)}")

# Function to upload a file to the server


def upload_file(client_socket, filename):
try:
if not os.path.isfile(filename):
print(f"The file '{filename}' does not exist.")
return

send_action(client_socket, "UPLOAD")
client_socket.send(filename.encode())
with open(filename, "rb") as file:
while True:
data = file.read(1024)
if not data:
break
client_socket.send(data)
print(f"File '{filename}' uploaded successfully.")

except Exception as e:
print(f"Error while uploading file: {str(e)}")

# Function to download a file from the server


def download_file(client_socket, filename):
try:
send_action(client_socket, "DOWNLOAD")
client_socket.send(filename.encode())
response = receive_response(client_socket)

if response == "ERROR":
print(f"File '{filename}' not found on the server.")
return

with open(filename, "wb") as file:
while True:
data = client_socket.recv(1024)
if not data:
break
file.write(data)
print(f"File '{filename}' downloaded successfully.")

except Exception as e:
print(f"Error while downloading file: {str(e)}")

# ... (other client-side functions)


if __name__ == "__main__":
client_socket = connect_to_server()

while True:
print("Available Actions:")
print("1. Upload a file")
print("2. Download a file")
print("3. List files on the server")
print("4. Delete a file on the server")
print("5. Rename a file on the server")
print("0. Exit")

choice = input("Enter action number: ")

if choice == "1":
filename = input("Enter the name of the file to upload: ")
upload_file(client_socket, filename)
elif choice == "2":
filename = input("Enter the name of the file to download: ")
download_file(client_socket, filename)
elif choice == "3":
# Implement the function to list files on the server (if available)
pass
elif choice == "4":
# Implement the function to delete a file on the server (if available)
pass
elif choice == "5":
# Implement the function to rename a file on the server (if available)
pass
```

```
elif choice == "0":
client_socket.close()
break
else:
print("Invalid choice. Please try again.")
```

Server

```
import socket
import os
import threading
import logging
import ssl

from network import send_file, receive_file, send_action, receive_action


def send_file(client_socket, filename):
# Function to send file to the client
pass


def receive_file(client_socket, filename):
# Function to receive file from the client
pass


def send_action(client_socket, action):
# Function to send action type to the client
pass


def receive_action(client_socket):
# Function to receive action type from the client
pass


def authenticate_user(username, password):
# Your authentication logic here
if username == "user123" and password == "password123":
return True
else:
return False

# Establish Secure Connection


def create_secure_connection():
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)

# Bind and listen for incoming connections
# Replace with actual server address and port
server_address = ('localhost', 12345)
server_socket.bind(server_address)
server_socket.listen(5)

# Accept incoming connections
client_socket, client_address = server_socket.accept()

secure_client_socket = ssl.wrap_socket(client_socket, server_side=True)

return client_socket

# File Upload and Download


def handle_file_upload(client_socket):
# Receive the filename and file contents from the client
filename = client_socket.recv(1024).decode()
file_contents = client_socket.recv(1024)

# Save the received file on the server
with open(filename, "wb") as file:
file.write(file_contents)


def handle_file_download(client_socket, filename):
```

```
# Read the file from the server
with open(filename, "rb") as file:
file_contents = file.read()

# Send the file contents to the client
client_socket.send(file_contents)

# Directory Listing


def get_directory_listing():
# Get the list of files and directories in the server's current directory
file_list = os.listdir()

# Convert the list to a string for transmission to the client
listing_str = "\n".join(file_list)

return listing_str

# File Deletion and Management


def delete_file(filename):
# Delete the specified file from the server
os.remove(filename)


def rename_file(old_name, new_name):
# Rename the file on the server
os.rename(old_name, new_name)

# Error Handling


def handle_error(error_message):
# Log the error on the server-side
logging.error("ERROR: %s", error_message)

# Inform the client about the error
client_socket.send(error_message.encode())


# Logging
logging.basicConfig(filename='server.log', level=logging.INFO)

# Concurrency and Multi-Client Support


def handle_client(client_socket):
try:
# Receive the action type from the client
action = receive_action(client_socket)

if action == "UPLOAD":
# Receive the filename from the client
filename = client_socket.recv(1024).decode()
# Handle file upload
receive_file(client_socket, filename)
elif action == "DOWNLOAD":
# Receive the filename from the client
filename = client_socket.recv(1024).decode()
# Handle file download
send_file(client_socket, filename)
elif action == "LIST":
# Handle directory listing
listing_str = get_directory_listing()
client_socket.send(listing_str.encode())
elif action == "DELETE":
# Receive the filename from the client
filename = client_socket.recv(1024).decode()
# Handle file deletion
delete_file(filename)
elif action == "RENAME":
# Receive the old and new filenames from the client
old_name = client_socket.recv(1024).decode()
new_name = client_socket.recv(1024).decode()
# Handle file renaming
rename_file(old_name, new_name)
else:
# Handle unknown action
handle_error("Unknown action.")

except Exception as e:
        # Handle any exceptions that occur during client communication
        handle_error(str(e))

        finally:
        # Close the client socket
        client_socket.close()


        # Configuration and Settings
        server_address = 'localhost'
        server_port = 12345

        if __name__ == "__main__":
        server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)

        # Bind and listen for incoming connections
        # Replace with actual server address and port
        server_address = ('localhost', 12345)
        server_socket.bind(server_address)
        server_socket.listen(5)

        # Main server loop to handle multiple clients
        while True:
        client_socket, client_address = server_socket.accept()

        # Start a new thread to handle the client
        client_thread = threading.Thread(
        target=handle_client, args=(client_socket,))
        client_thread.start()
```

Usage

To use the SFTP client, run the `client.py` file in the `client/` folder. The client will connect to the server and display a menu of available actions. Follow the on-screen instructions to perform various file actions.

To use the SFTP server, run the `server.py` file in the `server/` folder. The server will start listening for incoming client connections and handle file transfers and actions accordingly.

Security Considerations

The SFTP project implements encryption techniques to secure data transmission between the client and server. Data integrity is ensured by verifying the MD5 hash of the transferred files. However, potential vulnerabilities may exist, such as man-in-the-middle attacks or weak encryption key management. Future enhancements could address these concerns.

Challenges and Learnings

During the development of the SFTP project, some challenges were encountered, such as handling large file transfers efficiently and managing encryption keys securely. These challenges led to valuable learnings in networking, encryption, and data verification techniques.

Future Improvements

Some potential future improvements for the SFTP project include:

- Implementing secure key exchange protocols
- Supporting concurrent file transfers
- Enhancing error handling and logging

References

- Python Documentation: https://docs.python.org/
- `cryptography` Library: https://cryptography.io/
- MD5 Hash Algorithm: https://en.wikipedia.org/wiki/MD5