Analysis Report for: D328EAB763B65B499561C73AEB372571.exe.c

Overall Functionality

This C code implements a network proxy or a tunneling tool, likely acting as a middleman between two network connections. It appears designed to create a persistent connection between a client and a server, potentially for malicious purposes like data exfiltration or command-and-control communication. The code uses Windows sockets and handles multiple connection modes (listen, SOCKS5, slave). The extensive use of error handling and low-level file operations suggests an attempt to be stealthy and robust. The decompilation artifacts (comments, naming conventions) indicate the code may have been extracted from a malware sample.

Function Summaries

* **`main(int argc, const char **argv, const char **envp)`**: The entry point of the program. Parses command-line arguments to determine the operating mode (listen, SOCKS5 client, or slave) and initiates the corresponding network operations. Returns 0 on success.

* **`sub_4012F0(void *this)`**: Displays usage instructions for the program, likely called when incorrect arguments are provided. Returns the result of a logging function.

* **`sub_401340(int hostshort, int a2)`**: Listens on two specified ports, accepts connections on each, and then creates a thread (`StartAddress`) to handle the bidirectional data transfer between the two connections. Returns a result indicating success or failure of socket operations.

* **`sub_4015C0(SOCKET a1, SOCKET *a2)`**: Handles SOCKS5 protocol communication. Receives a request, processes authentication (if present), and establishes a connection to the target host/port. The established socket is written to the *a2 parameter. Returns 1 on success, -1 on error, and 0 for unsupported requests.

* **`sub_401A10(char *a1, int a2)`**: Connects to a specified host and port. Then acts as a middleman between client(s) and this connection, creating a thread for each connection using `StartAddress`. Returns a pointer, probably to memory allocated for threads.

* **`sub_401CF0(char *a1, int a2, char *a3, u_short a4)`**: Establishes two connections: one to a host/port specified by the first set of parameters and another to a second host/port specified by the second set of parameters. It then creates a thread (`StartAddress`) to handle the data relay between these two connections. Returns result of socket operations.

* **`StartAddress(SOCKET *lpThreadParameter)`**: This is the thread function. It performs the bidirectional data relay between two sockets passed as parameters. It continuously monitors both sockets using `select`, reading and writing data to maintain the tunnel. Returns the result of logging function.

* **`sub_402490(void)`**: (Weak) Likely performs some cleanup or finalization tasks. The code inside is not available.

* **`sub_402500(void)`**: (Weak) Likely creates a new socket. The code inside is not available.

* **`sub_402520(SOCKET s, u_short hostshort)`**: Binds a socket to a given address and port and starts listening for incoming connections. Returns 1 on success, 0 on error.

* **`sub_402691()`**: Returns a pointer to a global variable, possibly related to logging.

* **`sub_40276D()`**: Decodes a pointer, likely for obfuscation.

* **`sub_402D1C()`**: Calls a weak function `flsall`, potentially for thread local storage cleanup.

* **`sub_402E5E(char *String)`**: Converts a string to a long integer.

* **`sub_403049(unsigned __int8 *a1, int a2)`**: A logging function that writes a string to a file. Uses file locking mechanisms for thread safety. Returns result of the low-level write operation.

* **`sub_403E70(void *a1)`**: Stores a pointer in a global variable.

* **`sub_404BB4(int FileHandle)`**: Flushes the buffers of a file handle.

* **`sub_404C9A(int a1, const void *a2, DWORD nNumberOfBytesToWrite)`**: Writes data to a file handle.

* **`sub_404D7C(int a1, const void *a2, DWORD nNumberOfBytesToWrite)`**: Low-level file writing function, handles console and file output differently.

* **`sub_405953(FILE *a1, unsigned __int8 *a2, struct localeinfo_struct *a3, int *a4)`**: Complex formatting and writing function. Handles different format specifiers, potentially for log message creation.

* **`sub_407066(LPTOP_LEVEL_EXCEPTION_FILTER lpTopLevelExceptionFilter)`**: Sets the top-level exception filter.

* **`sub_407074(UINT uExitCode)`**: Terminates the current process.

* **`sub_407089(struct _EXCEPTION_POINTERS *ExceptionInfo)`**: Handles exceptions.

* **`sub_40709F(int FileHandle)`**: Closes a file handle.

* **`sub_407233()`**: Calls a series of functions pointed to by a global array.

* **`sub_407253()`**: Calls a series of functions pointed to by a global array.

* **`sub_407795(void *a1)`**: Stores a pointer in a global variable.

* **`sub_4077A2(int a1)`**: Stores an integer in a global variable.

* **`sub_4077AF(int a1)`**: Stores an integer in a global variable.

* **`sub_407F00()`**: Sets an exception filter.

* **`sub_408231(int a1)`**: Stores an integer in a global variable.

* **`sub_408B01()`**: Allocates and initializes file handles. Appears to manage a pool of file handles.

* **`sub_40942B(unsigned __int8 a1, FILE *Stream)`**: Writes a single character to a file.

* **`sub_409DC6(const WCHAR *a1, int a2, int a3)`**: Displays a message box (if a debugger isn't present) or logs a message. Handles different output methods based on whether the application is packaged or running in a debugger.

* **`sub_40AB38()`**: Closes a global handle.

* **`sub_40AC45(_DWORD *a1, int *a2, LPCWSTR lpFileName, int a4, int a5, char a6)`**: Creates a file handle. Handles various file access modes and flags.

* **`sub_40B720(int FileHandle, CHAR *lpWideCharStr, DWORD nNumberOfBytesToRead)`**: Reads data from a file handle. Handles different character encodings and console I/O.

* **`sub_40BEAF(_DWORD *a1)`**: Sets a global variable.

Control Flow

The control flow is complex, reflecting the multi-faceted nature of the program. The `main` function primarily handles argument parsing and delegates tasks to other functions based on the chosen mode. The core logic resides in `sub_401340`, `sub_401A10`, `sub_401CF0`, and `StartAddress`. These functions all use loops and conditionals to manage connections, data transfers, and error handling. The `StartAddress` function contains a main loop that uses `select` to monitor multiple sockets, handling input and output on each.

Data Structures

* **`SOCKET`**: Standard Windows socket type for network communication.
* **`sockaddr`**: Structure for network addresses.
* **`fd_set`**: Structure for managing sets of file descriptors in `select`.
* **`dword_413380`**: An array likely used for managing file handles. It appears to be a pool of handles with status flags.

Malware Family Suggestion

Given the functionality (network proxying/tunneling, extensive error handling, obfuscation hints), this code strongly suggests it's part of a **remote access trojan (RAT)** or a **downloader**. The SOCKS5 support is a common feature in RATs, enabling communication through firewalls. The complex data relaying mechanism in `StartAddress` could conceal C&C communication or data exfiltration. The lack of overt malicious actions (e.g., file deletion, encryption) suggests a downloader or a component of a larger malware family. Further analysis (looking at the undefined functions, inspecting strings within byte arrays, etc.) would be necessary to determine the exact malware family.