Analysis Report for: 3BF27599EB7AA99EECD127864966652F.exe.c

Overall Functionality

The provided C code appears to be a decompiled version of a malware sample. It's highly obfuscated, making precise determination of its functionality difficult without access to the original binary and potentially additional context. The code uses a large number of function pointers, indirect calls, and numerous arrays of unknown contents (many labeled `dword_XXXXXXX`, `byte_XXXXXXX`, `word_XXXXXXX`), indicative of anti-analysis techniques. The code seems to involve a main loop (`sub_401020`) that performs some sort of cyclical operation, potentially involving data manipulation, network communication (implied by error handling of network-related error codes in `sub_401150`), and cryptographic operations (suggested by the use of `__rdtsc` for generating a seemingly random number). The code exhibits characteristics consistent with a downloader or dropper, possibly delivering and executing additional payloads. The extensive use of weak symbols further increases the obfuscation.

Function Summaries

* **`sub_401000()`**: Calls a function pointed to by `dword_425300`, likely performing some initialization or cleanup function. Returns the result of that function call.
* **`sub_401010()`**: Calls a function pointed to by `dword_4252E4`, similar in purpose to `sub_401000`. Returns the result of that function call.
* **`sub_401020()`**: The main loop of the malware. It appears to repeatedly execute a series of functions, potentially performing actions such as data processing, network communication, or payload execution. It handles various error codes, particularly those related to network operations, as seen within the function `sub_401150`. Never returns (declared as `__noreturn`).
* **`start()`**: Calls `dword_4252D0(1)` and then `sub_401020()`, effectively starting the main malware execution loop. Never returns (declared as `__noreturn`).
* **`sub_401150(unsigned int **a1)`**: Handles error codes, particularly those related to network or system errors. It maps specific error codes (like -1073741819, -1073741795, -1073741676, -1073741674, -1073741677) to actions, some involving calls to further obfuscated functions (`dword_41D9B0`). Returns an integer indicating success or failure.
* **`sub_401290()`**: Returns a function pointer, possibly dynamically selecting a function based on some condition. The selection involves calls to functions pointed to by `dword_41DD10` and `dword_41DD20`.
* **`sub_4012F0()`**: Iterates through a list of function pointers stored in memory (starting at `dword_41E004`) and executes each one sequentially. Terminates when a null pointer is encountered.
* **`sub_401320()`**: Initializes a global variable (`dword_41E000`) and calls `sub_401290()`. It then potentially executes a series of functions from another function pointer array. Subsequently calls `sub_401000()`.
* **`sub_4013A0(char a1, _DWORD *a2)`**: A complex function containing numerous calls to other functions, many through function pointers. It seems to perform data processing and manipulation, potentially involving some form of cryptographic operation or data encoding/decoding. Involves loops and conditional branches and possibly network communication.
* **`sub_4016E4(int a1, int *a2, int a3)`**: Appears to interact with a data structure, potentially modifying it depending on flags and values. Includes some conditional logic based on byte checks within the data structure.
* **`sub_401BF8(int a1)`**: Simple function that likely serves as a jump table or dispatcher. It contains an out-of-bounds jump, indicating more obfuscation.
* **`sub_401C0A(int a1, int a2, _DWORD *a3)`**: Another very complex function with numerous function pointer calls and conditional logic. It manipulates data structures, and potentially handles network-related operations or data synchronization.
* **`sub_401DA2(int a1, int a2, ...)`**: Uses variable arguments. It likely passes data to another function through a function pointer call.
* **`sub_401E02(int a1)`**: Initializes and manages some global data structures. Contains conditional logic based on the input parameter.
* **`sub_401EB4()`**: Checks global state (`dword_423040`) and calls a function through a function pointer. Potentially retrieves a value for return based on state.

Control Flow

Most functions exhibit complex control flow, heavily relying on function pointers and indirect calls. The main loop (`sub_401020`) is an infinite loop, driven by the continuous execution of several functions. Detailed analysis of control flow would require significantly more time and potentially dynamic analysis of the original binary. Many of the functions (particularly `sub_4013A0` and `sub_401C0A`) have deeply nested loops and conditional statements that are difficult to completely unravel from the decompiled code alone.

Data Structures

The code uses many large arrays (e.g., `dword_4032B8`, `dword_403400`, `dword_41CD18`, etc.) of unknown data types and purposes. These are likely used to store function pointers, configuration data, or other critical information for the malware's operation. The meaning of the data stored in these arrays cannot be determined without more context.

Malware Family Suggestion

Based on its functionality (cyclical operation, network communication, and extensive obfuscation), this code most strongly suggests a **downloader/dropper** type of malware. It likely downloads and executes additional payloads. The intricate network error handling, combined with the data manipulation and potentially cryptographic aspects, hints towards a sophisticated piece of malware that attempts to evade detection. More advanced static and dynamic analysis would be needed to reach a more precise family classification. The use of anti-analysis techniques like extensive function pointer usage, indirect calls, and weak symbols further supports this classification. It is not possible to say for certain which specific malware family it belongs to based solely on the decompiled code.