Analysis Report for: 1731586B511E5F87715331A74C8DE818.exe.c

Overall Functionality

The provided C code is heavily obfuscated, likely by a decompiler (Hex-Rays) from disassembled machine code. It's impossible to definitively determine the overall functionality without access to the original assembly or more context. However, the code exhibits characteristics strongly suggestive of malware. Numerous functions are marked as `weak`, indicating potential polymorphism or dynamic code generation. The extensive use of indirect function calls, complex arithmetic, and bitwise operations points to an attempt to hinder reverse engineering. The presence of functions related to exception handling and potentially memory manipulation (though obscured) further strengthens this suspicion. The code appears to perform complex mathematical operations, possibly for encryption/decryption or obfuscation purposes. Its modular structure, with many small functions, is common in malware to make analysis more difficult.

Function Summaries

Due to the obfuscation, many functions lack clear semantic meaning. The following summarizes those functions where some interpretation can be attempted. Many others are impossible to interpret confidently without the original binary and more comprehensive reverse-engineering efforts.

* `sub_40146C()`: Appears to initialize global variables with addresses of other functions and data structures. Likely part of setting up the malware's environment.
* `sub_4014CC(int a1)`: Calls `sub_40146C()` and conditionally calls another function based on parameter `a1`.
* `sub_4014EB()`: A thunk (a simple wrapper function) that conditionally executes a function based on a flag.
* `sub_4014F5()`: Performs some FPU operations and calls `sub_401592()`. Potentially numerical computation related to cryptography or obfuscation.
* `sub_401592()`: A very large and complex function with heavy reliance on FPU instructions. Likely involved in some sort of numerical computation that is crucial to the malware's functionality. Possibly an encryption/decryption routine, given the level of complexity.
* `sub_401C48(double *a1)`: Checks if a double-precision floating-point number is non-negative.
* `sub_401C64()`, `sub_401CA6()`: Functions that appear to manipulate data, potentially copying or modifying data structures based on a flag.
* `sub_401E6B()`: A complex function that processes data, potentially string manipulation or data transformation. It handles error conditions and calls `sub_4014EB`.
* `sub_401F2F()`: A wrapper function calling `sub_401E68`.
* `sub_401F52()`: A large and complex function. The significant flow control suggests a substantial data processing step, possibly related to data encoding/decoding.
* `sub_40263E()`: Appears to perform an indirect function call, casting a pointer to a function. This is very suspicious.
* `sub_4027A9()`: An extremely complex function using SIMD instructions (`__m128d`). This strongly points towards highly optimized, intensive, numerical computation. Its sheer size and complexity imply a major role in the malware's core functionality, possibly encryption or data manipulation.
* `sub_403B6A()`, `sub_403B9C`, `sub_403BD8`: These functions seem to handle data structures, possibly arrays or linked lists, performing some operation based on input.
* `sub_404981(int a1)`: Simple assignment of `a1` to a global variable.
* `sub_405092()`: Contains a `__noreturn` call, suggesting a termination or crucial function call.
* `sub_406996()`: Likely executes an indirect function call through `nullsub_3`, which indicates more dynamic behavior.
* `sub_4075D1()`: Uses SIMD instructions (`__m128i`) and suggests more numerical processing, potentially dealing with arrays of data.
* `sub_407C82(double a1)`: Performs classification based on the value of a double-precision floating-point number. Looks like a conditional branch based on floating-point comparisons.
* `sub_408716()`: Performs integer division with overflow checks.
* `sub_40BBB2()`: Another large and complex function. Looks like it manipulates numerical data in a highly structured way, possibly for some form of encoding or signal processing.
* `sub_40CD2F()`: Contains indirect function calls and potentially dynamic code generation.
* `start()`: The entry point of the program, calls `sub_404BCA`.

Control Flow

Analyzing the control flow of every function is impractical given the complexity and obfuscation. However, the following highlights aspects seen in several key functions:

* **`sub_401592()`**: This function features deeply nested `if` statements and `goto`s, creating a very tangled control flow. The use of `__FYL2X__` (a floating-point instruction) and other FPU operations suggests extensive numerical computation.
* **`sub_401F52()`**: This function also uses significant branching with `if` statements and `goto`s, making it difficult to trace. It contains nested loops and potentially makes indirect function calls, increasing the obfuscation.
* **`sub_4027A9()`**: This function employs many SIMD instructions, possibly for parallel processing of numerical data. The control flow is incredibly complex due to nested conditional statements and loops.
* **`sub_40350E()`, `sub_403573()`**: These functions use `fxam` for FPU status checks and `xlat` (translate byte) suggesting interaction with data based on FPU results and potentially look-up tables.

Data Structures

The code uses various data structures, many of which are only hinted at due to obfuscation. Several arrays and potential custom structures are suggested but cannot be completely determined without the original assembly. The function parameters and local variables of functions like `sub_401F52()` and `sub_4027A9()` might indicate the use of structures for storing large amounts of data required for their complex computations. The presence of `__m128d` and `__m128i` indicates the use of SIMD registers and vectors for handling either doubles and integers respectively.

Malware Family Suggestion

Given the characteristics of the code (heavy obfuscation, complex numerical computations, potentially dynamic code generation, and indirect function calls), it's highly likely this is a sample of a sophisticated piece of malware. The specific family is impossible to identify without further analysis and knowledge of the underlying binary. However, the complexity and use of SIMD instructions lean towards more advanced malware families, possibly a type of polymorphic virus or a downloader/dropper, as opposed to simpler malware types. The use of Windows API calls (`NtCurrentTeb`, etc.) confirms the target operating system. Further analysis would be needed to determine specifics such as botnet communication, payload delivery, and persistence mechanisms.