Analysis Report for: 1731586B511E5F87715331A74C8DE818.exe.c.js

Overall Functionality

The provided C code is highly obfuscated, likely through the use of a decompiler (Hex-Rays). It's impossible to definitively determine the overall functionality without significantly more context or deobfuscation. However, the code exhibits characteristics strongly suggestive of malware, particularly a packer or dropper. The sheer number of functions, many with unclear purposes and many marked as `weak`, the extensive use of bitwise operations, and the presence of potentially malicious actions (e.g., `__debugbreak`, `JUMPOUT` to seemingly random addresses) all point to deliberate obfuscation to hinder reverse engineering. The code likely unpacks or drops other malicious payloads, performs actions based on specific conditions, and interacts with the system in ways indicative of malware. The many `__usercall` functions suggest compiler-specific calling conventions and further obfuscation.

Function Summaries

Due to the obfuscation, providing precise function summaries is challenging. Many functions lack clear names and their operations are obscured. I can provide a few examples, though many will be incomplete or speculative.

* `sub_40146C()`: Appears to initialize global variables with addresses of other locations within the code. Returns a pointer.
* `sub_4014CC(int a1)`: Calls `sub_40146C()` and conditionally calls `sub_40263E()`, possibly modifying the returned pointer based on the input. It also executes `fnclex` (clear floating-point exception flags). Returns a pointer.
* `sub_4014EB()`: A thunk, possibly selecting between two different functions based on a boolean input.
* `sub_4014F5()`: Appears to involve floating-point calculations and conditional jumps based on the value of `dword_419E64`. Calls `sub_401592`.
* `sub_40152D()`: Decompilation error; analysis failed.
* `sub_401534()`: Calls `sub_40153C()`.
* `sub_40153C()`: Conditional jump, likely to different code paths based on a boolean input. Calls `sub_401592`.
* `sub_401592()`: Extremely complex function with many floating-point operations, conditional jumps, and calls to other functions. Likely performs some core calculation or manipulation. This function is crucial to understanding the malware's behavior.
* `sub_401C48()`: Checks if a double-precision floating-point number is non-negative. Returns a boolean.
* `sub_405092()`: Terminates the program (via `__noreturn`), likely after some actions are completed. This would be a potential end point of the main execution.
* `sub_4027A9()`: Extremely long and complex function using SIMD instructions (`__m128d`). This suggests sophisticated mathematical operations, which could be used for encryption, decryption, or other complex computations related to obfuscation or payload execution.
* Many other functions are similarly complex and difficult to analyze without significant deobfuscation efforts.

Control Flow

The control flow is heavily obfuscated, with frequent `JUMPOUT` statements to seemingly random addresses. This makes it nearly impossible to follow the program's execution path without dynamic analysis or significant static analysis with deobfuscation techniques. Many functions contain nested loops and conditional branches that are hard to trace due to the confusing variable names and lack of meaningful comments.

The main execution likely starts with `start()` which appears to be a thunk to `sub_404BCA()`. Following the execution flow from there would require deep analysis and deobfuscation.

Data Structures

The code utilizes several arrays (e.g., `dword_41610C`, `dword_416700`), which might serve as lookup tables or store encoded data. The large array `dword_41610C` is particularly interesting, holding many different values. Other data is stored globally in various `dword_` and `xmmword_` variables; some are initialized, others appear to be used for intermediate calculations or as flags. The meaning and purpose of these data structures remain largely unknown without deobfuscation.

Malware Family Suggestion

Based on the observed characteristics, this code is highly suggestive of a **packer/dropper** type of malware. Packers obfuscate malicious code to evade detection, while droppers deploy additional payloads. The complex mathematical functions (especially `sub_4027A9`) and extensive use of obfuscation techniques like `JUMPOUT` and meaningless function names are common in advanced malware samples. The code's structure and behavior are consistent with packers that use decryption routines and perform system interactions that are characteristic of malware. Further analysis would be needed to determine if there are additional characteristics to suggest a specific malware family or variant beyond the packer/dropper classification. Dynamic analysis would be crucial to uncovering the true functionality and behaviors of this malware.