Analysis Report for: C9CE3D159FFA0A48063AE06DCF98CD33.exe.c

Overall Functionality

This C code, likely decompiled from a binary, appears to be a complex program with functionalities related to data management, potentially within a simulation or database application. The sheer number of functions (328) and the presence of many seemingly trivial functions (e.g., those named `sub_1400xxxxx`) suggest a heavily obfuscated or automatically generated codebase. A core part of the functionality deals with handling and manipulating linked lists. There's extensive use of `malloc` and `free`, indicating dynamic memory allocation, which is susceptible to memory corruption vulnerabilities if not handled carefully. The code also includes substantial error handling and reporting mechanisms, especially concerning runtime checks and stack corruption. The inclusion of Windows API functions like `GetModuleHandleW`, `GetProcAddress`, `LoadLibraryExW`, `RaiseException` and others suggests it's a Windows application. The presence of `_security_cookie` and security-related functions hints at some attempts to protect against exploits. However, many of these security mechanisms are weak and likely easily bypassed. The code also appears to contain extensive logging and information-gathering aspects, which are suspicious. The functions related to CPUID strongly indicate some form of system information gathering. The `sub_140016290` function appears to attempt to extract data from a PDB file and suggests that the code may utilize debugging information during runtime, often used in Anti-Reverse Engineering measures. However, this is not an absolute indication that this is not a malicious application.

Function Summaries

Due to the large number of functions and the lack of clear comments within the decompiled code, providing summaries for all 328 functions is impractical. However, a sample of functions is analysed below:

* **`sub_1400118B0()`**: This function calls `sub_1400113DE` (likely an obfuscated function) and returns a pointer to a global variable `unk_14001F938`. Its exact purpose is unclear without further context.

* **`sub_1400118F0(FILE *a1, const char *a2, __crt_locale_pointers *a3, va_list a4)`**: This function uses a dynamically obtained function pointer (`_stdio_common_vfprintf`) to perform formatted output to a file. It appears to be a wrapper around the standard C library function.

* **`sub_140011980(__int64 a1, __int64 a2)`**: This function adds a new node containing `a2` to the end of a linked list pointed to by `a1`. Error handling is implemented for memory allocation failures.

* **`sub_140011A50(__int64 a1)`**: This function counts the number of nodes in a linked list.

* **`sub_140011AD0(__int64 a1, unsigned __int64 a2)`**: This function iterates `a2` elements through a linked list starting at `a1`, returning the address of the `a2`-th element or NULL if the list is shorter.

* **`sub_140011B70(__int64 a1)`**: This function creates a new two-element linked list node containing the pointer `a1`.

* **`sub_140011C00(_QWORD *a1, unsigned int (__fastcall *a2)(_QWORD))`**: Iterates a linked list, calling a function (`a2`) on each node. This is used to process each element of the linked list.

* **`sub_140011CD0(void ***a1, unsigned __int64 a2)`**: This function deletes a node from a linked list. `a2` specifies the node to delete (0 for the first node).

* **`TopLevelExceptionFilter(struct _EXCEPTION_POINTERS *ExceptionInfo)`**: A top-level exception handler. This is a standard Windows exception handling mechanism that is used to catch exceptions that are not otherwise handled.

* **`sub_140013380()`**: This function appears to be the program's main entry point or initialization function. It initializes various components, registers callbacks, and handles potential errors.

Control Flow

The control flow in many functions is straightforward, mainly involving iterating linked lists or handling conditional logic for error situations. More complex functions involve a series of calls to internal functions (the `sub_1400xxxx` functions). For example, `sub_140016290` demonstrates a nested control flow, extracting data from PDB files through a series of function calls, returning a boolean indicating success or failure.

Data Structures

The primary data structure is a doubly linked list. The nodes in the list are dynamically allocated using `malloc` and likely contain various data members depending on the application's needs. The code also utilizes `_onexit_table_t` for registered exit handlers, which is a standard C runtime structure. Further data structures are difficult to discern due to heavy obfuscation.

Malware Family Suggestion

Given the characteristics of the code – heavy obfuscation, dynamic memory allocation, extensive error handling (potentially masking malicious behavior), system information gathering (CPU features), potential backdoor functionality through the multiple functions and the apparent use of PDB manipulation which often serves as an anti-reverse engineering technique, the possibility of this code being malicious should be treated as very high. It is likely a highly obfuscated backdoor/RAT (Remote Access Trojan) or information stealer. More analysis would be needed to determine its precise actions and targets, but the red flags are significant. The extensive use of linked lists suggests a modular architecture enabling easy

extension of functionalities. The potential PDB file parsing could be used to gather information about the host's software configuration or to dynamically load other malicious components. Further investigation with tools like a sandbox is necessary to definitively classify its malicious capabilities. Analysis should focus on the network traffic generated by this application when run in a controlled environment.