Analysis Report for: D842C75E60687D74D30FE7F9A9469FB6.exe.c

Overall Functionality

This C code, likely generated from a disassembled malware sample, exhibits characteristics consistent with a sophisticated, highly obfuscated piece of malware. It performs a series of operations that appear to involve:

1. **Module Enumeration and Search:** Functions `sub_10001190` and `sub_100012E0` iterate through loaded modules (likely using `InLoadOrderModuleList` and `InMemoryOrderModuleList`), searching for specific modules based on name comparisons. The comparison uses custom string manipulation functions (`sub_100030A0`, `sub_10003160`).

2. **Data Extraction and Processing:** Several functions handle data manipulation, including string operations, bitwise operations, and custom encryption/decryption routines (`sub_100027E0`, `sub_10002F10`, `sub_10002900`). This suggests data is being extracted from these located modules or elsewhere.

3. **Function Pointer Manipulation:** The code extensively uses function pointers (`dword_10061148`, `dword_10061150`, `dword_10061154`, etc.), which are initialized and called dynamically. This dynamic behavior is typical of malware employing polymorphism or anti-analysis techniques. The initialization of these pointers happens in `sub_100017F0`.

4. **System Calls (Likely Obfuscated):** `sub_10004482` contains a `sysenter` instruction, indicating a direct system call. This call is likely obfuscated, making analysis difficult.

5. **Initialization Routine:** `sub_10002DE0` appears to gather system information (OS version, build number, etc.). This information could be used for environment checks or creating unique identifiers.

6. **Custom Cryptography:** The code employs functions with strong indicators of custom cryptographic routines (`sub_100027E0`, `sub_10002F10`).

7. **Memory Allocation and Manipulation:** Function `sub_10002000` possibly allocates and initializes memory.

Function Summaries

Due to the obfuscation, precise function summaries are challenging. However, based on analysis, here's an attempt:

Function Name	Purpose	Parameters	Return Value
`sub_10001000`	Possibly an obfuscated function, likely used by other functions.	Integers and a character	Void
`sub_10001190`	Searches for a specific module in the process's loaded module list.	Integer (likely a string), `LIST_ENTRY` pointer	`LIST_ENTRY` pointer (module found) or NULL
`sub_100012E0`	Similar to `sub_10001190` but with wide character string input.	Wide character string, `LIST_ENTRY` pointer	`LIST_ENTRY` pointer (module found) or NULL
`sub_100013F0`	Core function; extracts and processes data based on module and string.	Integer, pointer to DWORD array, integer, char string	Pointer to data or NULL
`sub_10001670`	Calls `sub_100013F0` based on input parameters.	Integer, Integer	Integer
`sub_100016B0`	Possibly handles string manipulation and calls an external function.	Pointer to a string	Integer (result from external function call)
`sub_100017F0`	Initializes several global function pointers.	None	Void
`sub_100019F0`	Returns a function pointer initialized by `sub_100017F0`.	None	Function pointer
`sub_10001A10`	Calls a function pointer and performs additional data processing.	None	Integer
`sub_10001BC0`	Performs checks and possibly calls system functions.	Pointer to DWORD, pointer to Integer	Boolean (success/failure)
`sub_10001D50`	Complex function; performs checks and calls `sub_10001BC0`.	Pointer to DWORD	Boolean (success/failure)
`sub_10001EC0`	Iterates and processes data based on function pointers.	Pointer to DWORD	Integer
`sub_10002000`	Allocates and initializes memory; potentially decrypts data.	Pointer to character, unsigned int	Pointer to unsigned int
`sub_10002560`	Extracts data and processes based on string.	Pointer to DWORD, Pointer to character	Integer
`sub_100025B0`	Returns an integer, likely a handle or address.	None	Integer
`sub_10002620`	Possibly performs a check based on input.	Unsigned long long	Integer
`sub_10002720`	Performs a check based on input and calls `sub_10001000`.	Integer, Integer, Long long	Boolean (success/failure)
`sub_100027E0`	Custom encryption/decryption routine.	Integers and unsigned int	Unsigned char
`sub_10002900`	Custom operation on a byte array.	Pointer to byte array, Integer	Character
`sub_10002940`	Returns the exception list.	None	Pointer to `EXCEPTION_REGISTRATION_RECORD`
`sub_10002950`	Performs a check and copies data.	Integers, pointer to DWORD	Integer
`sub_10002980`	Exception handler function	Pointer to EXCEPTION_RECORD	Integer
`sub_10002AC0`	Initializes Exception Handlers	None	Integer
`sub_10002DD0`	Sets a global variable (dword_1006116C)	Integer	Integer
`sub_10002DE0`	Gathers system information and sets global variables.	None	Integer
`sub_10002F10`	Custom encryption/decryption routine.	Integers	Unsigned char
`DllEntryPoint`	DLL entry point.	HINSTANCE, DWORD, LPVOID	BOOL
`Kixewlinaglekubupaha`	Main function; orchestrates other functions.	Integers	Unsigned int
`sub_10003050`	Compares two byte arrays (case-insensitive).	Pointer to byte arrays, Integer	Integer (difference)
`sub_100030A0`	Case insensitive wide string comparison	Integer, pointer to unsigned short	Integer (difference)
`sub_10003100`	Copies a char array.	Integer, pointer to char, Integer	Integer

`sub_10003130`	Fills memory with a specified value.	Pointer to char, unsigned char, unsigned int	Pointer to char
`sub_10003160`	Copies a wide character array.	Pointer to short, Integer, unsigned int	Pointer to short
`sub_100031B0`	Performs complex operations, and appears to check for certain conditions	Pointer to Integer	Boolean
`sub_10003820`	Performs a series of operations on a byte array (potentially encryption).	None	Unsigned int
`sub_10003900`	Performs a series of operations on a byte array (potentially encryption).	None	Unsigned int
`Cudihupokoweti`	Performs a series of operations on a byte array (potentially encryption).	None	Unsigned int
`Kiqecahmnumatepnoswer`	Performs a series of operations on a byte array (potentially encryption).	None	Unsigned int
`Jagoxiseyag31`	Performs a series of operations on a byte array (potentially encryption).	None	Unsigned int
`Wirwwnnmu`	Performs a series of operations on a byte array (potentially encryption).	None	Unsigned int
`Levutorwiboba`	Performs a series of operations on a byte array (potentially encryption).	None	Unsigned int
`Juhorexidiwi`	Performs a series of operations on a byte array (potentially encryption).	None	Unsigned int
`Coxllwurexurlotijerwoy`	Performs a series of operations on a byte array (potentially encryption).	None	Unsigned int
`sub_10004040`	Initializes a DWORD array with data and performs custom encryption.	Pointer to DWORD	Pointer to DWORD
`sub_100042F0`	Main execution routine, calls numerous other functions.	Integers	Integer
`sub_10004460`	Simple loop, followed by a call to `sub_10004482`	Integers	Integer
`sub_10004482`	Performs a `sysenter` instruction (likely an obfuscated system call).	None	Void
`sub_10004490`	Calls a provided function with three integer parameters.	Function pointer, Integers	Integer (result from the called function)

Control Flow

The control flow is highly complex due to heavy obfuscation, especially within functions like `sub_100013F0`, `sub_10001EC0`, `sub_10002000`, and `sub_100031B0`. These functions feature nested loops, intricate conditional statements, and heavy use of function pointers, making a detailed analysis extremely challenging without dynamic analysis tools. The primary control flow is driven by function pointer calls and the results of string/data comparisons and environment checks.

Data Structures

The code uses standard C data structures like arrays and pointers extensively. The most notable data structure is `struct _LIST_ENTRY`, used for traversing linked lists of loaded modules. Custom structures are also heavily used but are obscured by names and heavy use of offsets.

Malware Family Suggestion

Given the extensive use of obfuscation techniques (heavy use of function pointers, custom encryption/decryption, complex control flow, and unclear names), coupled with the clear module enumeration and data extraction, this code strongly suggests a **sophisticated, polymorphic malware sample**. The functions that seem to perform extensive bitwise and arithmetic operations on data blocks point toward encryption or complex obfuscation. The `sysenter` call indicates direct system calls, which might be used for various malicious purposes (privilege escalation, data exfiltration, etc.). Further analysis with dynamic analysis tools is needed for a more precise malware family identification. The current obfuscation level makes static analysis for this purpose alone insufficient.