Analysis Report for: 1EE28B617044F4A3D0BBFA8BC8B2F899.exe.c

Overall Functionality

This C code appears to be a DLL (Dynamic Link Library) designed to interact with a VBA (Visual Basic for Applications) environment. It likely serves as a bridge between a native C/C++ application and VBA code within a host application like Microsoft Office. The functions heavily rely on VBA-specific functions (like `__vbaStrCat`, `__vbaObjSet`, etc.), suggesting it's designed to handle string manipulation, object creation and management, and potentially data transfer between the two environments. The presence of `DllEntryPoint`, `DllRegisterServer`, `DllUnregisterServer`, and `DllCanUnloadNow` confirms its role as a standard Windows DLL. However, the obfuscation (use of seemingly random function names like `sub_11001BFF`) and the presence of numerous seemingly trivial functions that just return 0 make reverse engineering and understanding its precise functionality difficult without further context.

Function Summaries

* **`DllEntryPoint(HINSTANCE hinstDLL, DWORD fdwReason, LPVOID lpReserved)`:** The standard DLL entry point. It simply forwards the call to another function (`UserDllMain`), which is not defined in this snippet. This function's actual functionality remains unknown.

* **`sub_11001BFF(int a1)`:** This function likely performs some action in a VBA context. It subtracts 103 from its input and passes the result to `sub_11003B90`.

* **`sub_11001C0C(int a1)`:** Similar to `sub_11001BFF`, this function modifies its input and passes it to `sub_11003E00`.

* **`DllUnregisterServer()`:** Standard DLL function to unregister the DLL from the system registry.

* **`DllGetClassObject(const IID *const rclsid, const IID *const riid, LPVOID *ppv)`:** Standard DLL function used by COM (Component Object Model) to obtain a class factory for a given class ID.

* **`DllRegisterServer()`:** Standard DLL function to register the DLL in the system registry.

* **`DllCanUnloadNow()`:** Standard DLL function to check if the DLL can be unloaded from memory.

* **`sub_11003B90(int a1)`:** This function appears to be the core of some VBA interaction, performing actions like object creation (`__vbaNew`), setting object properties (`__vbaObjSet`, `__vbaObjSetAddref`), string manipulation (`__vbaStrCat`, `__vbaStrMove`, `__vbaStrCopy`, `__vbaStrCmp`), and error handling (`__vbaOnError`). It also calls `__vbaExitProc` before returning.

* **`sub_11003E00(int a1)`:** This function also seems crucial for VBA interaction, it also involves similar VBA functions to `sub_11003B90` (object creation, manipulation, and string handling), suggesting two distinct but related operations within the VBA environment.

* **`sub_11004A5E(int a1, int a2)`:** This function's purpose is unclear due to the decompilation artifacts (`could not find valid save-restore pair`). It seems to dereference pointers, which suggests manipulation of some data structure.

* **`sub_110050C9(int a1, int a2)`, `sub_11005AC3(int a1)`, `sub_110085E3(int a1)`, `sub_11008EEB(int a1, int a2)`:** These functions simply return 0, indicating either stubs or remnants from earlier code versions.

* **`sub_1100594C(int a1, int a2, int a3, int a4, int a5)`, `sub_11006212(int a1, int a2, int a3)`, `sub_1100670A(int a1, int a2, int a3)`, `sub_11006D90(int a1, int a2, int a3)`, `sub_11007320(int a1, int a2, int a3)`, `sub_110076DC(int a1, int a2, int a3)`, `sub_11008132(int a1, int a2, int a3)`:** These functions copy data from one memory location to another, likely manipulating internal data structures. The decompilation notes suggest potential issues with stack frame analysis.

Control Flow

The control flow is generally straightforward within the main functions (`sub_11003B90` and `sub_11003E00`). They follow a linear sequence of calls to VBA functions. Conditional statements are minimal (like the `if` statement in `sub_11003B90` checking a string comparison). The other functions are primarily simple assignments or function calls that lead to the return of a value.

Data Structures

The code utilizes several arrays (`dword_11001F80`, `dword_11002014`). Their exact purpose is unclear without further context, but they are likely used to hold data relevant to VBA objects or other runtime information. The numerous warnings about positive SP values suggest that the decompiler had trouble accurately reconstructing the stack frames; this could indicate complex or unconventional data structure usage.

Malware Family Suggestion

Based solely on the provided code, it's impossible to definitively classify it as a specific malware family. The code is highly obfuscated, and the function names and many of the functions themselves lack semantic meaning. While it interacts with VBA, this doesn't automatically indicate malicious intent, as many legitimate applications utilize VBA.

However, several aspects raise suspicion:

* **Obfuscation:** The use of meaningless function names and the overall complexity make analysis difficult, a common characteristic of malware.

* **VBA Interaction:** Malicious macros often use VBA to perform harmful actions. The reliance on VBA-specific functions increases the potential for abuse.

* **Unclear Functionality:** The functions are not self-explanatory, so it's hard to determine their exact purpose, leaving room for malicious behavior to be hidden.

* **Potential Memory Corruption:** The decompiler warnings regarding stack frame analysis might suggest the code uses unsafe memory manipulation techniques. This is a red flag for potential instability, crashes or malicious behavior.

Conclusion:

Without further analysis in a dynamic environment (running the code in a sandbox) and having more context on where this DLL is called, a definitive assessment of its nature (malicious or benign) is impossible. The obfuscation and the use of VBA increase suspicion, necessitating a more thorough investigation. The presence of functions like `DllRegisterServer` and `DllUnregisterServer` suggest the DLL is attempting to integrate itself into the system which is another potentially suspicious behavior.