Analysis Report for: 1EE28B617044F4A3D0BBFA8BC8B2F899.exe.c

Overall Functionality

This C code appears to be a DLL (Dynamic Link Library) for a VBA (Visual Basic for Applications) application, likely acting as a backend for some functionality within a larger program. The code heavily relies on a set of VBA-related functions (e.g., `__vbaStrCat`, `__vbaNew`, `__vbaObjSet`, etc.), suggesting interaction with VBA objects and strings. The DLL exports standard COM (Component Object Model) entry points (`DllEntryPoint`, `DllUnregisterServer`, `DllGetClassObject`, `DllRegisterServer`, `DllCanUnloadNow`) indicating its design for in-process COM server usage. The numerous `sub_xxxx` functions seem to handle specific tasks within the VBA environment, potentially related to data manipulation or interaction with external resources. The presence of warnings about "positive sp value" suggests potential issues in the decompilation process, which may affect the accuracy of the analysis, particularly regarding stack frame usage.

Function Summaries

* **`DllEntryPoint(HINSTANCE hinstDLL, DWORD fdwReason, LPVOID lpReserved)`:** This is the standard DLL entry point. It appears to simply delegate its functionality to another function (`UserDllMain`), which is not directly defined in this code snippet.

* **`sub_11001BFF(int a1)` and `sub_11001C0C(int a1)`:** These functions act as wrappers, subtracting 103 from their input (`a1`) before calling `sub_11003B90` and `sub_11003E00`, respectively. The purpose of subtracting 103 is unclear without more context.

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

* **`DllGetClassObject(const IID *const rclsid, const IID *const riid, LPVOID *ppv)`:** Standard COM function to obtain a class factory for a specific class within the DLL.

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

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

* **`sub_11003B90(int a1)`:** This function performs complex operations involving VBA functions, likely manipulating VBA objects and strings. It creates a new VBA object, sets its properties (possibly from the input `a1`), performs string concatenation and manipulation, and handles potential errors using `__vbaOnError`.

* **`sub_11003E00(int a1)`:** Similar to `sub_11003B90`, this function uses VBA functions to work with objects and strings. It casts objects, sets object properties, and likely performs some initialization or setup tasks.

* **`sub_11004A5E(int a1, int a2)`:** This function calls a function pointer located at offset 8 in a memory address pointed to by the content of `a1 + 8`. It returns a value from memory related to `a1`. This behavior suggests a polymorphic method invocation or virtual function call.

* **`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 they might be placeholders, incomplete functions, or functions whose primary purpose isn't in performing actions but perhaps creating a specific effect in a larger context.

* **`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 the stack to a location pointed by the first parameter `a1`. They appear to perform straightforward data copying and are likely helpers within a broader context.

Control Flow

The control flow of `sub_11003B90` and `sub_11003E00` is linear, primarily involving calls to VBA functions. `sub_11003B90` contains a conditional statement based on a string comparison, determining whether a particular string is assigned a value. The control flow of the other functions is relatively simple; most of them consist of a single operation before returning.

Data Structures

The code uses several arrays (e.g., `dword_11001F80`, `dword_11002014`). The purpose of these arrays is unclear without further context. Their usage within `sub_11003E00` suggests they might represent object IDs or other internal data used by the VBA interface. The code also uses some structures whose specifics are not fully revealed due to the decompilation process and the absence of structure definitions.

Malware Family Suggestion

Based on the available information, this code doesn't strongly suggest a specific malware family. However, several aspects raise suspicion:

* **Obfuscation:** The use of seemingly arbitrary function names (`sub_1100XXX`), unclear number manipulations (subtracting 103), and heavy reliance on external VBA functions could indicate an attempt to obfuscate the code's true purpose.

* **DLL Structure:** The DLL structure with COM entry points suggests an attempt to integrate into a legitimate application, hiding malicious actions.

* **Data Manipulation:** Functions such as `sub_11003B90` and `sub_11003E00` perform extensive string manipulation and object interaction which could be used to steal data, modify system settings or alter application behavior.

Without further context and dynamic analysis, it's impossible to definitively classify this code as malicious. However, its characteristics raise significant suspicion, and further investigation is warranted. The code could be part of a dropper, information stealer, backdoor, or other types of malware depending on the context in which it is used. Reverse engineering and dynamic analysis would be necessary to fully assess its malicious potential.