Analysis Report for: F0DA118AB49F7E0138B7BDE65AF94889.exe.c

Overall Functionality

This C code appears to be part of a malware program designed to write a registry key and handle exceptions. The main function calls `sub_401000`, which attempts to create a registry key under `HKEY_LOCAL_MACHINE\Software\PaloAlto` with the value name "PanCar" and a DWORD value of 1. If it fails to create or write to the key, it logs the error to files "C:\\KeyOpenFailed.txt" and "C:\\KeyValueFailed.txt". The code also utilizes various Windows API functions obtained dynamically at runtime from `USER32.DLL`, suggesting an attempt to interact with the user interface or system processes. Several functions seem to deal with error handling and pointer encoding/decoding, potentially obfuscating the malware's actions.

Function Summaries

* **`sub_401000()`**: Attempts to create a registry key (`HKEY_LOCAL_MACHINE\Software\PaloAlto\PanCar`) and set its value to 1. It logs errors to files if the key creation or value setting fails. Returns an `LSTATUS` value indicating success or failure.

* **`main()`**: The entry point of the program. It simply calls `sub_401000()` and exits.

* **`sub_401937()`**: Calls the `flsall` function (likely related to fiber local storage). Its purpose in this context is unclear without more information about `flsall`.

* **`sub_401940()`**: Assigns the input pointer `a1` to the global variable `Ptr` and returns `a1`. Appears to be a simple pointer assignment function.

* **`sub_401B65()`**: Returns a pointer to the global variable `off_40D180`. The purpose of this variable is unknown.

* **`sub_4025B8()`**: A function that appears to write data to a buffer (potentially a file or memory). It handles potential errors during the write operation. The condition `(*(_BYTE *)(a3 + 12) & 0x40) == 0 || *(_DWORD *)(a3 + 8)` suggests it might be checking buffer properties before writing.

* **`sub_403211()`**: Sets the unhandled exception filter to `__CxxUnhandledExceptionFilter`, indicating a desire to handle C++ exceptions.

* **`sub_403E6F()`**: Returns a pointer to the `dword_40BAF0` array. The purpose of this array is unknown.

* **`sub_403E95()`**: An empty function.

* **`sub_4043EA()`**: Sets the global variable `dword_40EB48` to 0.

* **`sub_4071C4()`**: Decodes a pointer using the `DecodePointer` function (likely a custom function for obfuscation) and returns the result.

* **`sub_407374()`, `sub_407383()`, `sub_407392()`**: These functions appear to assign their integer or pointer arguments to global variables (`dword_40EB10`, `dword_40EB14`, `dword_40EB18`, respectively).

* **`sub_407503()`**: This is a complex function that dynamically loads functions from `USER32.DLL` (`MessageBoxW`, `GetActiveWindow`, `GetLastActivePopup`, `GetUserObjectInformationW`, `GetProcessWindowStation`). It seems to gather information about the current window and user session, and then possibly displays a message box. The exact behavior depends on the values of its arguments and the state of global variables.

* **`sub_407BD5()`**: Checks if a processor feature (likely SSE2 or another instruction set) is present using `IsProcessorFeaturePresent` and stores the result in `dword_40EB44`.

* **`sub_408645()`**: Sets the value of a provided pointer to `dword_40EB3C`. It handles null pointer exceptions and sets the errno value accordingly.

* **`sub_4098F3()`**: Closes the handle `hObject` if it's valid (not -1 or -2).

Control Flow

The crucial control flow lies within `sub_401000()` and `sub_407503()`. `sub_401000()` has a simple conditional branch based on the success or failure of registry operations. `sub_407503()` contains a complex control flow based on the loading of functions from `USER32.DLL` and the results of system calls. It dynamically determines its actions based on the existence and results of these functions.

Data Structures

No complex user-defined data structures are explicitly defined. The code primarily utilizes standard C data types and Windows API structures implicitly. The global variables, especially those of type `PVOID`, likely hold encoded pointers to functions or data.

Malware Family Suggestion

Based on its behavior, this code strongly suggests a **registry-based backdoor or downloader**. The creation of a registry key under a seemingly innocuous name ("PaloAlto") is a common technique for persistence. The dynamic loading of functions from `USER32.DLL` hints at potential attempts to interact with the user interface (e.g., displaying messages, capturing information). The logging of errors to files may indicate a self-diagnostic capability. The obfuscation techniques (pointer encoding) make it harder to analyze. Further analysis would be needed to determine if it's merely a component of a larger malware family or a standalone program. It shows similarities with techniques used by various information-stealing malware or other types of backdoors.