Analysis Report for: AFCA77888B30A169C7BA30BB0D21C0DD.exe.c

Overall Functionality

This C code is a heavily obfuscated program, likely malware, designed to perform several malicious actions. The code's structure suggests it's a self-extracting installer or a dropper, unpacking and executing further malicious payload. It appears to install itself persistently by modifying the registry (adding a RunOnce key), and potentially deleting files or directories. The extensive use of Windows API calls related to file manipulation, registry access, and process creation points toward a multifaceted threat. The numerous weak functions suggest the malware may use different techniques to evade detection based on the system environment.

Function Summaries

The code contains numerous functions, many with names like `sub_14000XXX`, indicating they were likely generated by a decompiler. Here's a summary of some key functions:

* `sub_140001008`: Copies a null-terminated string from one location to another, handling potential buffer overflows. Returns an error code.
* `sub_1400010BC`: Uses `vsnprintf` to format a string using variable arguments, similar to `printf`. Handles potential buffer overflows. Returns an error code.
* `sub_140001130`: Checks if the current user is a member of the Administrators group using the `CheckTokenMembership` function from `advapi32.dll`. Returns 1 if successful, 0 otherwise.
* `sub_140001258`: Determines user privilege level (Administrator or not). It uses `GetTokenInformation` to check group memberships, potentially to modify its behavior based on the user's privileges. Returns a numerical code representing privilege level.
* `sub_140001470`: Dialog box procedure. Handles different message types, showing error messages or closing dialogs.
* `sub_140001558`: Parses a string, extracting substrings delimited by a specified character. Returns a pointer to the next part of the string.
* `sub_1400015F4`: Parses an installer-like command line argument string. It extracts file paths, section names, and other parameters. It handles the execution of commands (.INF, .BAT). Returns error codes or indicates successful processing.
* `sub_140001D10`: Performs registry manipulation, adding a value to the `RunOnce` registry key. This ensures the malware runs on the next system boot.
* `sub_140002034`: Recursively deletes files and directories.
* `sub_140002EDC`: The main initialization function. Calls various other functions to perform actions like extracting resources, checking the operating system version, checking for the presence of a Mutex (to prevent multiple instances), and finally launching the GUI interface.
* `pfnopen`, `pfnread`, `pfnwrite`, `pfnfdid`, `pfnseek`, `pfnalloc`, `pfnfdin`: These functions seem to implement a custom file I/O system, possibly for extracting embedded resources. The names suggest they are emulating the standard file handling functions.
* `sub_140004838`: Creates a new process using `CreateProcessA`.
* `sub_140004B70`: Opens a folder browser dialog box using `SHBrowseForFolder` and `SHGetPathFromIDList`.
* `sub_1400051F8`: Loads a resource from the executable. Returns the size of the loaded resource.
* `sub_1400053B8`: Deletes a file. Checks for specific files to remove.
* `StartAddress`: Thread function that likely performs the main installation/execution actions.
* `TopLevelExceptionFilter`: Custom exception handler.

Control Flow

The control flow is complex and highly intertwined. Key functions like `sub_140002EDC` and `StartAddress` manage the main execution flow, involving many nested loops and conditional branches:

* `sub_140002EDC`: This function checks various system parameters and resources before proceeding. It demonstrates a complex decision-making process, performing different actions (e.g., installing via registry, showing GUI, extracting other payloads) based on the system configuration and user permissions.
* `StartAddress`: This is a thread procedure. It uses `FDICopy` (along with the custom file I/O functions) to extract a cabinet file, indicating that additional malware components might be embedded within the main executable.

Data Structures

* `struct _EXCEPTION_POINTERS`: This standard Windows structure is used for exception handling.
* `struct _CONTEXT`: Another standard Windows structure used to save the CPU context.
* `struct _STARTUPINFOA`: Standard Windows structure for process creation.
* `FDICABINETINFO`: This structure holds data related to the cabinet file.
* `ERF`: Custom error reporting structure likely used to communicate between functions.

Malware Family Suggestion

The combination of self-extraction, registry modification for persistence, file deletion, process creation, and dialog box interaction makes this sample malware look like an **installer-based malware** or **dropper**. The complex unpacking, custom file system, and error handling strongly suggest an attempt at obfuscation and anti-analysis techniques. It could potentially be categorized as a **downloader** if it fetches additional payloads from remote servers (this aspect is not evident in this snippet, but is consistent with the observed functionality).

Further Analysis

The code requires extensive reverse engineering to completely understand its behaviour. A deeper analysis should focus on:

* **Network communication:** Check for any network connections the malware attempts to establish.
* **Strings:** Analyze the string constants, particularly file paths, registry keys, and URLs.
* **External function calls:** Examine API calls to understand the actions the malware is performing.
* **Dynamic analysis:** Running the code in a sandbox environment will provide insights into the malware's behavior and effects on the system.

Without dynamic analysis and a thorough examination of any embedded files, it is impossible to say with certainty what all payloads this malware eventually executes. The code provided only shows the unpacking and install stage.