Analysis Report for: 66CC4C8E6A0BE81A873BF5B45D3B4C11.exe.c

Overall Functionality

This C code is an obfuscated installer, likely for a piece of malware. It performs several actions characteristic of malicious installers, including:

* **Integrity Check:** It verifies the installer's integrity upon launch. Failure leads to an error message.
* **Privilege Escalation:** It attempts to obtain the `SeShutdownPrivilege`, suggesting an intent to perform actions with elevated system rights.
* **File System Manipulation:** It creates, reads, writes, moves, and deletes files, potentially dropping malicious payloads or modifying system settings.
* **Registry Manipulation:** It interacts with the Windows Registry, potentially adding or removing registry keys and values.
* **Process Creation:** It creates new processes, possibly executing other malicious components.
* **Resource Extraction/Deobfuscation:** The code deobfuscates and extracts resources from itself (likely embedded malware components) based on an embedded configuration.
* **UI Interaction:** The installer displays a dialog box that shows progress for certain steps of the installation.

The heavy use of indirect function calls and the lack of meaningful variable names strongly indicates an attempt to hinder reverse engineering efforts. The large number of functions, many with cryptic names (e.g., `sub_401000`, `sub_40117D`), further complicates analysis.

Function Summaries

Due to the obfuscation, precise function summaries are difficult. However, based on the analysis so far, here's a high-level summary:

Function Name	Purpose	Parameters	Return Value
`sub_401000`	Window procedure for the main application window. Handles painting and window messages.	`HWND hWnd`, `UINT Msg`, `WPARAM wParam`, `LPARAM lParam`	`LRESULT`
`sub_40117D`	Manipulates an internal data structure likely related to installer stages.	`int a1`	`int*`
`sub_4011EF`	Processes a list of internal flags; possible stage management.	`unsigned int a1`, `int a2`	`unsigned int`
`sub_401299`	Modifies an internal state based on a bitmask.	`unsigned int a1`	`char`
`sub_4012E2`	Checks flags in an internal data structure.	`int a1`	`unsigned int`
`sub_40136D`	Accesses data from an array, potentially deobfuscating strings.	`int a1`	`int`
`sub_401389`	Executes installer stages and provides progress updates.	`int a1`, `HWND hWnd`	`int`
`sub_40140B`	Executes a sequence of installer operations.	`int a1`	`int`
`sub_401423`	String manipulation or output.	`int a1`	`LPWSTR`
`sub_401434`	Main installer logic dispatcher. Handles numerous installer operations.	`int lpFileName`	`int`
`sub_402C1F`	Accesses data from an array, likely part of configuration data.	`int a1`	`int`
`sub_402C41`	Retrieves strings from an embedded data structure.	`int a1`	`WCHAR*`
`sub_402C81`	Opens or creates a registry key.	`HKEY phkResult`	`unsigned int`
`sub_402CB9`	Likely modifies an index value for accessing configuration data.	`int a1`	`int`
`sub_402CD1`	Opens or creates a registry key.	`int a1`, `LPCWSTR lpSubKey`, `HKEY phkResult`	`unsigned int`
`sub_402CFF`	Deletes registry keys.	`int a1`, `LPCWSTR lpSubKey`, `int a3`	`int`
`sub_402D44`	Recursively deletes registry keys.	`HKEY a1`, `LPCWSTR lpSubKey`, `int a3`	`int`
`DialogFunc`	Dialog procedure for the installer progress dialog box.	`HWND hWnd`, `UINT a2`, `WPARAM a3`, `LPARAM a4`	`INT_PTR`
`sub_402E79`	Creates or destroys the installer's progress dialog.	`int a1`	`HWND`
`sub_402EDD`	Deobfuscates and extracts the installer's configuration data.	`int Buffer`	`wchar_t*`
`sub_403116`	Writes data to a file.	`int nDenominator`, `HANDLE hFile`, `LPVOID a3`, `signed int nNumberOfBytesToWrite`	`int`
`sub_403331`	Reads data from a file.	`LPVOID lpBuffer`, `DWORD nNumberOfBytesToRead`	`BOOL`
`sub_403347`	Sets the file pointer for a file handle.	`LONG lDistanceToMove`	`DWORD`
`sub_40335E`	Creates a temporary file.	None	`LPWSTR`
`start`	Main entry point of the installer.	None	None
...	...	...	...

Control Flow (Examples for significant functions)

* **`sub_401434` (Main Installer Logic):** This function acts as a large switch statement, dispatching control to other functions based on the value of a four-byte integer (`*(_DWORD *)v191`). Each case represents a different operation, such as file system actions, registry operations, or process creation. Error handling is minimal, often increasing a global error counter (`v195`).

* **`sub_402EDD` (Deobfuscation):** This function reads data from the installer's executable file, performs a series of checks (likely to verify version and integrity), and then proceeds to deobfuscate embedded configuration data based on the results of these checks. The deobfuscation involves bitwise operations and array lookups. Failure leads to error reporting.

* **`sub_403116` (File Writing):** This function writes data to a file, handling both buffered and unbuffered writes. It incorporates progress reporting mechanisms using `GetTickCount()` and `MulDiv()` for percentage calculation. It also deals with potential errors during write operations.

Data Structures

Several important data structures are used:

* **Embedded Configuration Data:** Large arrays and structures within the code hold the installer's configuration. This data is likely deobfuscated during the execution of `sub_402EDD`. The configuration dictates which operations are performed and their parameters.

* **Internal State Structures:** Several arrays (e.g., arrays starting at `dword_47AF28`) appear to maintain the installer's internal state, flags, and execution flow.

* **Standard Windows Structures:** The code makes extensive use of standard Windows API structures like `MSGBOXPARAMSW`, `SHELLEXECUTEINFOW`, `LOGFONTW`, etc., for interacting with the operating system.

Malware Family Suggestion

Based on the observed functionalities (integrity check, privilege escalation, file system manipulation, registry manipulation, process creation, and sophisticated obfuscation), this code strongly suggests a **malicious installer** and potentially belongs to a family of **installers for advanced persistent threats (APTs)** or other sophisticated malware. The detailed obfuscation efforts indicate the authors' desire to conceal the true nature of the payload. Further analysis would be necessary to identify the specific malware family. The use of COM objects also hints at a more sophisticated, possibly nation-state sponsored, type of malware.

Further Analysis Recommendations:

To fully understand this malware, further analysis would require:

* **Deobfuscation:** The heavy obfuscation needs to be systematically removed to reveal the true logic and the nature of the payload(s).
* **Static Analysis:** A deep dive into the data structures and the exact way in which the "installer operations" are being orchestrated.
* **Dynamic Analysis:** Executing the code in a sandboxed environment to observe its runtime behavior and identify dropped files or registry modifications.
* **Payload Identification:** Determining the type and functionality of the final malicious payload(s) that this installer deploys.

Due to the complexity of the code, using advanced reverse engineering tools and techniques are advised. This should be done in a safe and controlled environment to prevent potential harm.