Analysis Report for: 038440C2BEA4C2A8B47E60550732293C.exe.c

Overall Functionality

This C code is a kernel-mode driver for the Windows operating system. Its primary purpose appears to be to create a device and symbolic link (PROCEXP152), suggesting potential for interacting with system processes. The driver extensively utilizes undocumented and/or internal Windows APIs, exhibiting behavior consistent with rootkit or other malicious activity. Its many functions obfuscate the exact purpose but strongly suggest malicious intent. The extensive use of memory allocation and deallocation, coupled with string manipulation and system information gathering, points to a sophisticated approach to hiding its actions.

Function Summaries

The code contains a large number of functions (65), many of which are obfuscated by the decompiler. Here's a summary of some of the key functions:

* `DriverEntry`: The entry point of the driver. It calls `sub_180001000` to initialize the driver.

* `sub_180001000`: This function appears to be the main initialization routine. It creates a device object and a symbolic link, potentially preparing the driver to operate within the system. It checks the Windows build number, suggesting compatibility issues or different behavior depending on the OS version.

* `sub_1800013B0`: This function duplicates a handle from one process to another. It checks whether the target process is protected (using `PsIsProtectedProcess`), suggesting an attempt to escalate privileges or bypass security measures.

* `sub_180001500`: This function seems to perform operations related to process handles. The use of `KeStackAttachProcess` suggests it is manipulating objects in the context of a specific process.

* `sub_180001610`: A dispatcher function that handles various requests. It performs actions based on a request code (`v10`), involving process manipulation, handle duplication, and querying system information.

* `sub_180001D60`: An IRP (I/O Request Packet) handler. It likely handles device I/O requests and forwards the requests to `sub_180001610`.

* `sub_180002040`, `sub_180002480`: These functions appear to gather information about processes, possibly for exfiltration or other malicious purposes. They use string manipulation and handle operations.

* `sub_180002730`: Checks if a process is protected.

* `sub_180002780`: Manipulates memory regions within a thread's context. This is highly suspicious and could be used to inject code or alter process behavior.

* `sub_180002960`: Unloads the device driver, releasing the device and symbolic link.

* `sub_18000708C`, `sub_180007124`: These functions seem to handle device creation and possibly security descriptor manipulation.

* `sub_1800072B4`: Sets the security descriptor on a device object. This could be used to hide the device from normal access.

* `sub_180008530`, `sub_1800085D0`: Functions that create registry keys.

* `sub_180008648`, `sub_1800086B8`: Open registry keys.

* `sub_18000870C`, `sub_180008A2C`: Query registry key values.

* `sub_18000878C`, `sub_180008A80`: Query registry key values (more detailed information).

* `sub_180008ADC`: Set registry key values.

* `sub_180008B44`, `sub_180008B88`: Set registry key values (more specific).

* `sub_180008BE8`: Allocate memory and zero it out.

Many other functions are present, but their exact purpose is unclear due to obfuscation and the use of internal Windows APIs.

Control Flow

The control flow is complex, but some key aspects include:

* `sub_180001000`: The main control flow is sequential, creating the device and symbolic link. There are checks for OS build number that affect subsequent behavior. Error handling is minimal, and failure might not be handled correctly.

* `sub_180001610`: This function uses a `switch` statement to dispatch different operations based on an input code. Each case handles a distinct request (e.g., querying system information, modifying process handles).

* `sub_180001D60`: The IRP handler checks the `MajorFunction` code to determine what type of operation to perform.

Data Structures

Several Windows data structures are used, including:

* `IRP` (I/O Request Packet): Used for device communication.
* `DEVICE_OBJECT`: Represents the created device.
* `UNICODE_STRING`: Used for Unicode strings.
* `SECURITY_DESCRIPTOR`: Used for security access control.
* `ACL` (Access Control List): Part of the security descriptor.
* `OBJECT_NAME_INFORMATION`: Contains the name of an object.
* `PEPROCESS`: Represents a process object.

The code also uses several custom data structures, whose exact definitions are not available without the original source code.

Malware Family Suggestion

Based on the functionality of the provided code, it strongly resembles a **rootkit** and possibly a **backdoor**. The creation of a hidden device and symbolic link, combined with the ability to manipulate processes and access system information, is classic rootkit behavior. The capability to handle I/O requests suggests a backdoor allowing remote command and control. The obfuscation techniques employed significantly increase the difficulty of reverse engineering and analysis, making it even more suspicious. The ability to operate on other processes without being easily detected is another indicator. More sophisticated static and dynamic analysis would be needed to definitively classify it and identify specific malicious actions.