Analysis Report for: 7939C0DF93A584E709B01F69C0F22B91.exe.c

Overall Functionality

This C code appears to be a driver installer, specifically designed to install drivers on Windows systems. It interacts with the operating system through various Windows APIs, notably those related to device management (SetupAPI), registry manipulation (RegCreateKeyExA, RegSetValueExA), and file I/O (CreateFileA, ReadFile, WriteFile). The installer communicates with a named pipe (`\\\\.\\pipe\\libwdi-installer`), suggesting a client-server architecture where this code acts as the client. It handles different error conditions and logs events to a log file (setupapi.dev.log, setupapi.log, or setupact.log). A significant portion of the code deals with error handling and logging. The installer also manipulates system restore points. The presence of extensive error handling, along with functions for reading and parsing data, indicates an attempt at robustness. However, the obfuscation through function names and heavy reliance on external libraries makes definitive assessment difficult.

Function Summaries

The code contains a large number of functions. Here's a summary of some key functions:

Function Name	Purpose	Parameters	Return Value
`main`	The main entry point of the program; orchestrates the driver installation.	argc, argv, envp	Exit code (0 for success, non-zero for failure)
`sub_1400010B4`	Converts a multi-byte string to a wide-character string.	Multi-byte string	Wide-character string (NULL on failure)
`sub_140001144`	Copies an OEM INF file.	INF file path, other parameters	Number of bytes written (0 on failure)
`sub_14000124C`	Writes formatted output to a log file.	Format string, variable arguments	None
`sub_140001314`	Loads and initializes functions from Cfgmgr32.dll and Msvcrt.dll.	None	1 on success, 0 on failure
`sub_140001448`	Writes a single byte to the named pipe.	Byte to write	BOOL (TRUE on success, FALSE on failure)
`sub_140001478`	Reads data from the named pipe asynchronously.	Byte to send, buffer, buffer size	Number of bytes read (0xFFFFFFFF on failure)
`sub_1400015D0`	Reads device ID, hardware ID, or user SID from the named pipe.	Request ID (1-3)	String (NULL on failure)
`sub_1400016B8`	Re-enumerates a device node.	Device ID	0 on success, 0xFFFFFFFF on failure
`sub_14000174C`	Flags removed USB devices for reinstallation.	Driver name (can be NULL)	None
`StartAddress`	Thread function to read and process log file entries.	Pointer to unnamed pipe	Does not return
`sub_140001EDC`	Formats a Windows error code into a human-readable string.	Error code	Error message string
`sub_140002014`	Handles and logs error codes.	Error code, optional parameter	Error code (or 0 for success)
`sub_140002264`	Manages system restore point creation settings in the registry.	1 to disable restore points, 0 to restore them	1 on success, 0 on failure

Control Flow (Significant Functions)

* **`main`**: The `main` function is the entry point. It initializes the named pipe, loads necessary DLLs, parses command-line arguments (handling both UTF-16 and ANSI), gets the full path of the INF file, and then performs the driver installation process. It uses `UpdateDriverForPlugAndPlayDevicesW` for the core installation. Error handling involves checking return values of API calls and using `sub_140002014` for error code interpretation and logging. A separate thread (`StartAddress`) is created to monitor and process log file entries.

* **`StartAddress`**: This thread function continuously monitors a log file (setupapi.dev.log, setupapi.log, or setupact.log). It reads new lines from the file and processes them using `sub_140001924`. If it encounters `` in the file, it does nothing. Otherwise, it converts the new lines into a format to send through the pipe and sends them. It handles errors such as file read failures and memory allocation failures.

Data Structures

The code uses several important data structures, most of them are implied from function arguments:

* **Named Pipe**: The code uses a named pipe (`\\\\.\\pipe\\libwdi-installer`) for communication.
* **Windows Structures**: various Windows API structures such as `_OVERLAPPED`, `_OSVERSIONINFOW`, `_SP_DEVINFO_DATA`, `_exception` are likely used implicitly by several functions

Malware Family Suggestion

Given the functionality of installing drivers and interacting with the system registry and system restore points, this code bears some characteristics of a rootkit or backdoor. Rootkits often install drivers to maintain persistence and hide malicious activity. The communication with a named pipe suggests a backdoor component waiting for instructions from a remote server. The extensive error handling and logging could help to avoid detection and to hide the malicious activity. However, without additional analysis such as examining the INF file that this program will install and further reverse-engineering of the obfuscated functions (especially unknown_libname_* functions), it is impossible to definitively classify it as malware. The possibility of it being legitimate software is low. A further, more in-depth analysis is needed to make a conclusive statement about whether this is malware.