Analysis Report for: 7728B54CFE1E9CDBE09F9D3A75429F95.exe.c

Overall Functionality

This C code is highly obfuscated, likely to hinder reverse engineering efforts. It appears to be a malicious program, possibly a downloader or installer for other malware. The code interacts with the Windows API extensively, performing operations such as loading external libraries ("WinInet.dll", "USER32.DLL"), retrieving resource data from the executable, manipulating strings, and allocating memory. A key characteristic is its reliance on a significant number of small, interconnected functions with unclear purposes. The presence of exception handling and obfuscation strongly suggests malicious intent. The final stages of `WinMain` involve network communication, retrieving data, and potentially executing it.

Function Summaries

Due to the obfuscation, precise summaries are challenging. However, here's a high-level interpretation of several functions based on their names, parameters, and API calls:

* `sub_401030`: Throws an exception, likely used for error handling or program termination.
* `sub_401060`: Extracts a specific string from a resource embedded within a module.
* `sub_401110`, `sub_401160`, `sub_4011B0`: Functions involved in locating resources (strings) within loaded modules. This suggests the main payload or configuration data is stored within resources.
* `sub_4012F0`: This appears to be a critical function, possibly responsible for extracting the main code's execution path. It uses `GetModuleFileNameA` and manipulates paths.
* `sub_401230` to `sub_402130`: These are primarily helper functions, potentially manipulating strings, memory, or data structures.
* `sub_4018A0`: Measures the length of a null-terminated string (`strnlen`).
* `WinMain`: The main entry point of the program. It orchestrates resource extraction, network communication, and code execution.
* `sub_402170`: Performs checks related to file attributes and paths, possibly determining the execution environment.
* `sub_402600`: Loads "WinInet.dll" and retrieves function pointers for several network-related functions (e.g., `InternetOpenA`, `InternetConnectA`, `InternetReadFile`).
* Functions starting with `sub_4030`: These manage memory allocation and deallocation using the Windows heap functions.
* Functions involving `_decode_pointer` and `_encode_pointer`: Obfuscation techniques to hide function addresses.

Control Flow

* **`WinMain`**: This function has a complex control flow. It first calls `sub_402170` for environment checks. It then performs a time-based check before calling `sub_402600` to load network functions. The core functionality involves network communication using the functions loaded in `sub_402600`. The retrieved data is then processed and potentially executed.
* **`sub_4012F0`**: This function retrieves the executable's file path using `GetModuleFileNameA`. Based on the path, it either extracts data from embedded resources or sets a default path/data using `byte_40CAB2`.

Data Structures

The code uses several structures, many inferred:

* Standard C data types (int, char, etc.).
* `_RTL_CRITICAL_SECTION`: A Windows structure for thread synchronization, suggesting that the malware handles multiple threads.
* Custom structures implied through function parameters and pointer arithmetic. These are likely used to manage memory allocations, resource data, and possibly network connections. Their exact layout is unclear due to obfuscation.
* Array `byte_40CAB2`: Appears to hold default string data, possibly a fallback path or configuration if resource extraction fails.

Malware Family Suggestion

Given its use of resource extraction, network communication, and obfuscation techniques, this code strongly suggests a **downloader/installer** type of malware. The network communication functions imply that the malware downloads additional components or instructions from a remote server. The checks on file paths and the date/time suggest an attempt to evade detection and ensure execution at a specific time or in a particular environment. The level of obfuscation is typical of advanced malware that aims to avoid analysis. It's highly likely that the final stage of the malware (the downloaded/decoded payload) would perform malicious actions such as data exfiltration, system compromise, or further malware installation.