Analysis Report for: 7728B54CFE1E9CDBE09F9D3A75429F95.exe.c

Overall Functionality

This C code appears to be part of a malicious program, likely a downloader or dropper. The code heavily utilizes Windows API functions related to resource loading (`FindResourceA`, `LoadResource`, `LockResource`), process/thread management (`CreateThread`, `WaitForSingleObject`, `SetEvent`, `ExitProcess`), file system access (`GetModuleFileNameA`, `GetFileAttributesA`, `SHGetFolderPathA`), and memory management (`HeapAlloc`, `HeapFree`, `VirtualAlloc`). Its core functionality involves loading resources from a module (possibly itself), performing checks based on system time and file paths, and ultimately attempting to download and execute additional code from a remote server. The extensive use of obfuscation techniques, including function renaming (`sub_xxxx`), makes precise determination difficult, but the overall pattern strongly suggests malicious intent.

Function Summaries

Due to the length and complexity of the code, I will summarize a selection of the most crucial functions that reveal the malware's behavior:

* **`WinMain`**: The entry point of the program. It orchestrates the entire malicious operation. It performs checks (date/time, file existence), loads a library ("WinInet.dll"), and uses functions from that library to connect to a remote server (`38.47.239.143`), download a payload, and execute it.

* **`sub_402170`**: This function checks for the presence of specific files in the `%AppData%\Local\Temp` directory and on the Desktop. It also attempts to create a shortcut if the conditions are met. This is a reconnaissance step, likely to identify the victim's operating system and configuration.

* **`sub_402600`**: This function loads the "WinInet.dll" library dynamically and retrieves function addresses for networking operations, such as `InternetOpenA`, `InternetConnectA`, `InternetReadFile`, etc. This is preparation for network communication and the download of a secondary payload.

* **`sub_401060`, `sub_401110`, `sub_401160`, `sub_4011B0`**: These functions work together to load embedded resources from the executable using the Windows API. This suggests the program contains encoded data that is loaded and later decoded/executed.

* **`sub_4012F0`**: This function appears to handle file path manipulation, potentially to construct the download location for the secondary payload.

* **`sub_40301F` - sub_403350`**: These functions seem to manage internal data structures, likely related to handling the downloaded payload. The functions heavily utilize critical sections for thread synchronization.

* **`sub_401030`**: This function throws a C++ exception, likely used for error handling or to obfuscate code flow.

Control Flow (Significant Functions)

* **`WinMain`**: The control flow is complex, involving multiple conditional checks and function calls. The main path involves:
1. Checking for specific files and file paths.
2. Loading the WinInet library.
3. Attempting to retrieve data from a remote server.
4. Decrypting the downloaded data.
5. Allocating memory using VirtualAlloc and executing the decrypted data.

* **`sub_402170`**: This function iteratively checks for the existence of files at various locations, primarily in the `%AppData%\Local\Temp` directory and the desktop. If files are absent, it might attempt to create a shortcut ('Telegram.lnk').

Data Structures

The code uses several data structures, but their precise definition is obscured by the decompilation and obfuscation. The functions `sub_40301F` - `sub_403350` and the functions operating on `_DWORD*` seem to manage custom data structures which likely represent internal program state. The functions manipulating arrays of `_DWORD` likely hold the decrypted payload or configuration details.

Malware Family Suggestion

Based on the analysis, this code strongly suggests a **downloader/dropper** type of malware. It's designed to retrieve and execute further malicious code from a remote location. The date/time check indicates a possible attempt to limit its execution window, and the file system checks suggest some form of reconnaissance or persistence mechanism. The obfuscation level is also consistent with sophisticated malware families often seen in advanced persistent threats (APTs). It's difficult to pinpoint a specific malware family without further investigation and analysis (e.g., analyzing the decrypted payload).

Further Analysis

To gain a more comprehensive understanding, further analysis is required:

* **Dynamic analysis:** Run the code in a sandboxed environment to observe its behavior in real-time. This would reveal network connections, file system modifications, and process creation.
* **Static analysis (deeper):** Use more advanced static analysis tools to deobfuscate the code further and obtain clearer function names and data

```
structure definitions.
* **Payload analysis:** Once the secondary payload is downloaded, the code needs to be reverse engineered to know the exact malware.


This analysis provides a high-level overview of the code's malicious characteristics. Due to the code obfuscation, a complete and precise
understanding requires more in-depth reverse engineering. This code should **not** be executed on a production system.
```