Analysis Report for: 787728CB67656EF6C39ACF618E8356E5.exe.c

Overall Functionality

This C code appears to be malicious, designed to inject and execute code into another process (likely "explorer.exe"). It uses several Windows API calls related to process creation, memory allocation, and thread injection. The code also involves cryptography (using the BCrypt API), suggesting an attempt to obfuscate its payload. The program creates a directory, copies itself to that directory, and then creates a shortcut.

Function Summaries

* **`sub_140001000`**: Fills a memory region with a specified byte value. It takes a byte pointer, a byte value, and a count as input and returns the original byte pointer.

* **`sub_1400010D0`**: Copies a specified number of bytes from one memory location to another. It takes two memory pointers and a byte count. It returns the first input memory pointer.

* **`sub_140001230`**: Compares two strings (byte arrays) case-insensitively and returns the difference between the first differing characters if found, or the difference in length if the strings are prefixes of each other or if one is an empty string, otherwise it return 0

* **`sub_140001490`**: Compares two strings (word arrays) case-insensitively. Returns the difference between the first differing characters if found, or 0 if strings are equal.

* **`sub_140001680`**: This function seems to search for a specific module (likely its own) in the loaded modules list of the current process and then performs a complex operation involving byte-by-byte comparison, possibly code decryption or data transformation using `sub_140001230`. Returns an integer.

* **`sub_140001AA0`**: Finds the process ID of "explorer.exe" using the Tool Help library. Returns the process ID or 0 if not found.

* **`sub_140001C50`**: This function uses the BCrypt API for symmetric key generation and decryption. It decrypts input data (`a1`) using a key derived from input data (`a3`). The decrypted data is then written to a new memory location and copied to another location (`a5`). The length of the decrypted data is stored in `a6`. Returns 0 on success, -1 on failure.

* **`sub_140002310`**: Injects code into a specified process. It allocates memory in the target process, writes data to it, and then creates a thread that executes the injected code. The injected code's entry point is determined by `sub_140001680` and may indicate self-referencing and the need to locate its own code. Returns 1 on success, 0 on failure.

* **`sub_140002570`**: Creates a directory ("MyApp") in the Application Data directory, copies itself ("app.exe") to that directory, and returns the full path to the copied executable.

* **`sub_140002670`**: Creates a shortcut to the copied executable in the MyApp directory. It uses COM to interact with the shell.

* **`start`**: The main function. It orchestrates the entire process: copies itself, creates a shortcut, decrypts its payload, injects the payload into explorer.exe, and then cleans up.

Control Flow

Most functions follow a straightforward control flow. Key control flow aspects:

* **`sub_140001C50`**: Uses nested `if` statements based on the success of cryptographic operations. Failure at any stage results in cleanup and a return value of -1.

* **`sub_140002310`**: The success of the injection depends on `VirtualAllocEx` and `WriteProcessMemory` calls. If either fails, the function returns 0. The thread creation is also checked.

* **`start`**: This function's control flow is sequential, with error checks after each major step. Failure at any point leads to `ExitProcess`.

Data Structures

The code uses standard data types (bytes, words, pointers) and some Windows-specific structures, such as `PROCESSENTRY32`, `BCRYPT_ALG_HANDLE`, `BCRYPT_KEY_HANDLE`, etc. No custom, complex data structures are apparent.

Malware Family Suggestion

Based on its functionality, this code strongly resembles a **dropper** or a **remote access trojan (RAT)**. The dropper functionality is evident in the copying and shortcut creation. The injection of code into explorer.exe is characteristic of a RAT aiming for persistence and potentially broader system compromise. The use of cryptography adds to its stealthiness, making analysis more difficult. The code's actions point to it being a downloader that may acquire additional malware components. Further analysis would be necessary to determine the exact payload and its capabilities. The use of COM and the structure suggests a sophisticated design, making it potentially difficult to remove once executed.