Analysis Report for: Program.cs

Overall Functionality

This C# code appears to be an obfuscated program loader. It takes a hex-encoded string as a command-line argument (-b), or if not present, performs a complex hashing procedure to generate a decryption key. The core functionality involves loading and executing a secondary assembly ("Flutter") from data embedded within the main executable. This secondary assembly is likely the actual payload of the program. The code heavily relies on cryptographic functions (SHA256, RC4) and custom encoding/decoding schemes. The use of `DllImport("user32.dll")` and inter-process communication suggests potential interaction with the user interface.

Function Summaries

* **`SendMessage(IntPtr hwnd, int msg, IntPtr wParam, IntPtr lParam)`:** A P/Invoke call to the Windows API function `SendMessage`. Sends a message to a window. `hwnd` is the window handle, `msg` is the message ID, `wParam` and `lParam` are message parameters. Returns the result of the message processing.

* **`CalcHashThread(object ctx)`:** A thread function that performs a series of SHA256 hashing operations to generate a decryption key. It uses a seed (`Program.seed`) and a counter. The generated key is sent to the main window via `SendMessage`. The `ctx` parameter is the window handle. It doesn't return a value.

* **`DecodeVarint32(byte[] buffer, int offset, ref uint value)`:** Decodes a variable-length integer (Varint32) from a byte array. `buffer` is the byte array, `offset` is the starting position, and `value` receives the decoded integer. Returns the number of bytes read or 0 if decoding fails.

* **`ConvertHexStringToByteArray(string hexString)`:** Converts a hexadecimal string to a byte array. Takes a hex string as input and returns the corresponding byte array.

* **`Main(string[] Args)`:** The main entry point of the program. Parses command-line arguments, performs decryption (if needed), loads and executes the embedded assembly ("Flutter"), and handles program termination. It doesn't return a value.

Control Flow

* **`CalcHashThread`:** This function contains an infinite loop (`for(;;)`) that continues until a specific SHA256 hash condition is met (three leading zero bytes). This loop is designed to find a specific value that satisfies the hash condition, computationally intensive. The result is then used to create a final hash, which is passed back to the main thread.

* **`DecodeVarint32`:** This function iterates through the byte array, checking the most significant bit of each byte to determine if it's the last byte of the Varint. This efficient way to encode variable-length integers ensures that small integers can be represented using a fewer number of bytes than larger integers.

* **`Main`:** The `Main` function's control flow is complex, involving several conditional statements (`if`, `else if`, etc.) based on the presence of command-line arguments and the successful decoding of embedded data. It checks for a magic number ("KS") at the beginning of the embedded data. If this check passes, it decodes Varints, extracts keys, and attempts to load and execute the secondary assembly. There is an explicit error handling (`try...catch`) block for assembly loading.

Data Structures

* **`byte[] Program.seed`:** A byte array used as a seed for the hashing process. It's initialized from data extracted from `CodeGen.GetBytes()`.

* **`Dictionary dictionary`:** A dictionary that stores byte arrays indexed by integers. It seems to hold different parts of the decoded data from `CodeGen.GetBytes()`.

* **`byte[] Program.RedistData`:** A large byte array containing potentially encrypted or encoded data. Its purpose is unclear without further analysis, but it's passed to the "Flutter" assembly.

Malware Family Suggestion

The characteristics of this code strongly suggest it is a **dropper** or a **program loader** that could be used to deliver a more harmful payload. The obfuscation techniques (complex hashing, RC4 encryption, custom data encoding), the loading of an external assembly, and the command-line argument for injecting code all point towards malicious intent. The "Flutter" assembly is likely a downloader or a component of a more comprehensive malware family. Further analysis of the "Flutter" assembly would be needed to pinpoint a specific malware family. The use of a separate thread for key generation hints at attempts to hide its activity while also utilizing a GUI element to seem legitimate. This is a common obfuscation tactic used by malicious software.