Analysis Report for: B7305B1CA4E336AAB30997085F9E8072.cs

Overall Functionality

The C# code in `Program.cs` acts as a loader for a secondary assembly. It searches for a file with a `.tma` extension in the application's directory, loads its contents as a byte array, decrypts the array using a simple XOR cipher, loads the decrypted bytes as an assembly, and then executes a specific method within that loaded assembly. The target method is identified based on criteria involving the name "args" and optionally "tart". The code includes error handling and conditional logic based on the execution environment (checking for "rssimulator" and "lector" substrings in paths and the assembly name). The delay of 2 seconds before reading the file and the year check before invoking the method are obfuscation techniques.

Function Summaries

* **`Main(string[] args)`**: This is the entry point of the program. It takes command-line arguments (`args`) as input. It doesn't return a value (void). Its primary function is to load and execute the secondary assembly.

Control Flow

The `Main` function follows this control flow:

1. **File Location and Loading:** It constructs the path to the `.tma` file. It waits for 2 seconds before attempting to read the file. It checks if the application directory contains "rssimulator" substring. If it does, it reads the `.tma` file into a byte array.

2. **Decryption:** If the application name does not contain "lector" substring, it decrypts the byte array using a simple XOR cipher with a rotating key.

3. **Assembly Loading:** The decrypted byte array is loaded as an assembly using `Assembly.Load()`.

4. **Method Selection:** It searches the loaded assembly for methods that match specific criteria: they must be static, non-public, have one parameter named "args". If multiple methods match, it prioritizes methods whose declaring type's name contains "tart", otherwise it picks the last one.

5. **Method Invocation:** If a matching method is found, and the current year is after 1900, it invokes the method with the command-line arguments.

6. **Error Handling:** A `try-catch` block handles exceptions that may occur during file I/O, assembly loading, or method invocation. Error messages are printed to the console.

Data Structures

* **`string text`:** Stores the path to the `.tma` file.
* **`FileInfo fileInfo`:** Represents the `.tma` file's metadata.
* **`byte[] array`:** Stores the contents of the `.tma` file (and after decryption the assembly code).
* **`Assembly assembly`:** Represents the loaded assembly.
* **`IEnumerable enumerable`:** A collection of `MethodInfo` objects representing potential methods to invoke in the loaded assembly.
* **`MethodInfo methodInfo`:** Represents the selected method to invoke.
* **`object[] array3`:** Contains the command-line arguments passed to the invoked method.

Malware Family Suggestion

Given its functionality of loading and executing a secondary assembly from a custom file format (`.tma`) after a simple decryption, this code is strongly indicative of a **dropper** or a **downloader**. The obfuscation techniques (XOR encryption, sleep timer, and conditional checks) are common in malware to hinder analysis and detection. The "rssimulator" and "lector" checks suggest the malware may attempt to avoid execution in certain analysis environments (sandboxes or virtual machines). The selection of a method in a loaded assembly using name matching indicates a modular design, allowing the malware authors to easily swap out different malicious payloads by changing the assembly contents. The code does not contain any direct malicious actions, its goal is to execute another assembly and that assembly can contain arbitrary code. Therefore, the analysis concludes that this code is a sophisticated dropper with a capability to load and execute a secondary payload.