Analysis Report for: 18A8AA40871288B811FC55920F0A1741.cs

Overall Functionality

The provided code consists of several C# files that appear to be part of a larger software project. The core functionality centers around a configuration utility (`FI.SqlResUtilConfig`) possibly related to database access and rule management. It uses attributes for obfuscation and contains configuration data relevant to some process management or access control system. There is also an unrelated assembly information file for `inray.SDK.OPC.Windows.NetApi`, suggesting multiple projects are included in this code snippet.

Function Summaries

* **`DotfuscatorAttribute(string a, int c)`:** This constructor initializes a custom attribute used likely for code obfuscation. `a` and `c` are strings and integers respectively, whose purpose is not explicitly defined but probably represent obfuscation parameters.

* **`DotfuscatorAttribute.A`:** A getter property that returns the string `a` passed to the constructor.

* **`DotfuscatorAttribute.C`:** A getter property that returns the integer `c` passed to the constructor.

* **`ConfigRegrasAcesso.CarregarConfiguracaoRegrasAcesso()`:** This function appears to load configuration settings related to access rules. It returns a `ConfiguracaoRegrasAcesso` object. The internal logic is obfuscated with seemingly meaningless arithmetic operations and a switch statement with no cases. The function returns a hardcoded set of access rules.

Control Flow

* **`DotfuscatorAttribute`:** The constructor directly assigns values to internal fields. The getter methods simply return the values of these fields. No complex control flow is present.

* **`ConfigRegrasAcesso.CarregarConfiguracaoRegrasAcesso()`:** The function contains confusing arithmetic operations on `short` variables (`num`, `num2`, `num3`). These operations do not seem to affect the final result; they're likely obfuscation. The `switch` statement is empty. The `if` statements are also effectively no-ops as they don't contain any code. The function's core logic is within the return statement which returns a `ConfiguracaoRegrasAcesso` object populated with hardcoded `ProcDLL` objects containing arrays of `EnumAlias` values.

Data Structures

* **`DotfuscatorAttribute`:** A simple class with two private fields (`a`: string, `c`: int`) and corresponding getter properties.

* **`ConfiguracaoRegrasAcesso`:** This class (not fully shown) appears to hold access rule configurations. It contains a property (likely a collection or list) named `Procs` that consists of `ProcDLL` objects.

* **`ProcDLL`:** (Inferred) A class likely representing a procedure or DLL with an ID and an array (`EnumAlias[]`) of associated aliases or permissions.

* **`EnumAlias`:** (Inferred) This represents an enumeration that likely defines specific access levels or functionalities.

Malware Family Suggestion

Based on the code's characteristics, it's difficult to definitively label it as belonging to a specific malware family. However, some aspects raise concerns:

* **Obfuscation:** The extensive use of seemingly meaningless arithmetic operations and an empty switch statement in `CarregarConfiguracaoRegrasAcesso` strongly suggests deliberate obfuscation to hide the true functionality. This is a common technique employed by malware to evade detection.

* **Hardcoded Configuration:** The hardcoded nature of the access rules in `CarregarConfiguracaoRegrasAcesso` is suspicious. It could be a sign of malicious behavior. It might be used to provide access control to specific processes or data in a way that is harder to reverse engineer.

* **`DotfuscatorAttribute`:** The custom attribute is a valid obfuscation technique, however it's usage with seemingly cryptic parameters raises suspicion if the purpose and values of these aren't well understood within the development team.

Therefore, while not definitively malware, the code exhibits characteristics indicative of obfuscation techniques often used in malware to hide its malicious behavior. Further investigation of the functionality of `ProcDLL` and `EnumAlias` and the larger system within which this code operates would be necessary to determine if this code is part of a larger malicious program. A static analysis alone isn't enough to reach a firm conclusion. The code is strongly suspicious and requires further context and analysis.