Analysis Report for: 18A8AA40871288B811FC55920F0A1741.cs

Overall Functionality

The provided code consists of several C# files related to a .NET assembly named "FI.SqlResUtilConfig.SrvGVCaptLancBloq". The primary functionality appears to be configuration management for access rules, possibly within a database or application related to financial transactions (suggested by names like "LECCAFINANCEIRA" and the presence of modules related to data access). The `DotfuscatorAttribute` suggests the assembly has undergone obfuscation to protect its intellectual property. The `ConfigRegrasAcesso` class loads and likely applies configuration settings. The other files (`AssemblyInfo.cs`) contain metadata about the assemblies.

Function Summaries

* **`DotfuscatorAttribute(string a, int c)`:** This constructor initializes a custom attribute with two parameters: a string (`a`) and an integer (`c`). The purpose of these parameters within the obfuscation process is unclear without knowing the details of the Dotfuscator tool's configuration.

* **`DotfuscatorAttribute.A` (getter):** Returns the string value (`a`) passed to the constructor.

* **`DotfuscatorAttribute.C` (getter):** Returns the integer value (`c`) passed to the constructor.

* **`ConfigRegrasAcesso.CarregarConfiguracaoRegrasAcesso()`:** This function loads and returns a `ConfiguracaoRegrasAcesso` object. The internal logic involves some seemingly meaningless arithmetic operations on short integers (likely obfuscation). The function then returns a `ConfiguracaoRegrasAcesso` object containing a list of `ProcDLL` objects, each associated with an array of `EnumAlias` values. This suggests configuration data is being hardcoded into the assembly.

Control Flow

* **`DotfuscatorAttribute`:** The constructor directly assigns the input parameters to private fields. The getter methods simply return the values of these private fields. No significant control flow exists.

* **`ConfigRegrasAcesso.CarregarConfiguracaoRegrasAcesso()`:** The function's core logic is obfuscated. The switch statement is empty, and the `if` statements are also non-functional, implying they serve no purpose other than code obfuscation. The function ultimately returns a hardcoded `ConfiguracaoRegrasAcesso` object.

Data Structures

* **`DotfuscatorAttribute`:** A simple class containing two private fields: a string (`a`) and an integer (`c`).

* **`ConfiguracaoRegrasAcesso`:** This appears to be a class (the exact structure is not fully shown) that encapsulates access rule configurations.

* **`ProcDLL`:** A class likely representing a procedure or DLL with an ID and an array of `EnumAlias` objects (presumably representing some type of permission or access level).

* **`EnumAlias`:** This is likely an enumeration type; the actual definition isn't provided.

Malware Family Suggestion

Based solely on the provided code snippets, there is **no indication of malicious behavior**. The code is obfuscated, which is common in commercial software to protect intellectual property, not necessarily a hallmark of malware. The presence of obfuscation makes reverse engineering more difficult, but it doesn't automatically classify the software as malicious. The hardcoded configuration data is a potential security concern (e.g., sensitive information might be embedded if this were a different application), but not necessarily indicative of malware. More context (the functionality of `EnumAlias`, the external data accessed or modified, etc.) is needed for a proper security assessment.

Further Analysis Needed

A complete analysis requires:

* **Understanding `EnumAlias`:** The definition and use of this type is crucial in determining the meaning of the configuration data.
* **Examining `ProcDLL`:** More details about its purpose are needed.
* **Reviewing External Dependencies:** The code interacts with external libraries (`FI.Dados`, `FI.Modulos.DML`) which are not provided. Analyzing these libraries is essential.
* **Investigating Obfuscation Techniques:** A deeper analysis of the obfuscation techniques used might reveal clues about the intentions behind the code.

Without more information, it's impossible to definitively rule out the possibility of malicious intent, but the provided code alone doesn't provide sufficient evidence to classify it as malware.