Analysis Report for: 00588E364CF77C0528CF17F84CA91937.cs

Overall Functionality

The provided code consists of several C# files, primarily focusing on a Dynamics NAV codeunit (`Codeunit7050260`). This codeunit appears to manage change logging, specifically determining whether a given table should always be logged. It interacts with Dynamics NAV's runtime environment and utilizes event subscriptions. The other files are assembly information files, containing metadata about various projects and potentially obfuscated code in `FI.SqlResUtilConfig` (indicated by the `DotfuscatorAttribute`). There's no evidence of direct malicious activity, however, the obfuscation raises suspicions.

Function Summaries

* **`Codeunit7050260(ITreeObject parent)`**: Constructor for the codeunit. Initializes the base class with the parent object and codeunit ID.
* **`ObjectName`**: Getter property returning the name of the codeunit ("GsiChangeLogManagement").
* **`OnInvoke(int memberId, object[] args)`**: Handles method invocations on the codeunit. It specifically checks for `memberId == 7050000` (corresponding to `OnAfterIsAlwaysLoggedTable`) and calls the appropriate method. Otherwise it throws an error.
* **`__Construct(ITreeObject parent)`**: Static factory method for creating instances of `Codeunit7050260`.
* **`OnClear()`**: Override of a base class method; empty in this case.
* **`OnAfterIsAlwaysLoggedTable(int tableID, ByRef alwaysLogTable)`**: This is the core logic. It's an event subscriber that is triggered after a table's "always log" status is being determined. It modifies the `alwaysLogTable` parameter based on some internal logic.
* **`IndirectPermissionList`**: Getter property returning an array of indirect permission IDs.
* **`Codeunit7050260()`**: Static constructor initializing the `indirectPermissionList`.
* **`OnAfterIsAlwaysLoggedTable_Scope.Run()`**: The core logic inside a separate scope class. Checks certain conditions involving a `changeLogSetup` record and `tableID`, setting `alwaysLogTable` to `true` if conditions are met.
* **`OnAfterIsAlwaysLoggedTable_Scope.RawScopeId`**: Getter and setter for the raw scope ID.
* **`OnAfterIsAlwaysLoggedTable_Scope.EventScope`**: Getter and setter for the event scope.
* **`OnAfterIsAlwaysLoggedTable_Scope(Codeunit7050260 βparent, int tableID, ByRef alwaysLogTable)`**: Constructor for the `OnAfterIsAlwaysLoggedTable_Scope` class.
* **`OnAfterIsAlwaysLoggedTable_Scope.OnRun()`**: Contains the main logic for determining if a table should always be logged.

Control Flow

* **`OnInvoke`**: A simple `if-else` statement checks the `memberId`. If it's 7050000, it validates the number of arguments and calls `OnAfterIsAlwaysLoggedTable`. Otherwise, it throws an error.
* **`OnAfterIsAlwaysLoggedTable`**: Creates a scope object and calls its `Run` method.
* **`OnAfterIsAlwaysLoggedTable_Scope.OnRun`**: This function contains a complex conditional statement using bitwise AND operations (`&`) to check several conditions. Each condition involves `base.CStmtHit`, indicating statement hit counting for debugging, and operations on the `changeLogSetup` record handle. If all conditions are true, `alwaysLogTable.Value` is set to `true`. The specific meaning of the magic numbers used (e.g., 409565, 34048, 2000000073) is unclear without further context (likely internal Dynamics NAV IDs).

Data Structures

* **`uint[] indirectPermissionList`**: An array of unsigned integers representing permission IDs.
* **`OnAfterIsAlwaysLoggedTable_Scope`**: A nested class representing a scope for the event handler. It encapsulates the `tableID`, `alwaysLogTable`, and a `NavRecordHandle` (`changeLogSetup`).
* **`NavRecordHandle`**: A Dynamics NAV handle to a specific database record (likely of type 402 in this context, but its meaning is unknown without Dynamics NAV schema knowledge).
* **`ConfiguracaoRegrasAcesso`**: A class used in `FI.SqlResUtilConfig.SrvGVCaptLancBloq` to represent configuration of access rules. Its members aren't fully visible, but it includes a list of `ProcDLL` objects.
* **`ProcDLL`**: A class in `FI.SqlResUtilConfig.SrvGVCaptLancBloq` that seems to represent a DLL procedure with associated enum aliases.

Malware Family Suggestion

The provided code itself doesn't exhibit malicious behavior. It's a codeunit designed to manage change logging in Dynamics NAV. However, the presence of extensive obfuscation in the `FI.SqlResUtilConfig` project, using seemingly random numbers and the `DotfuscatorAttribute` (a common tool for protecting code, but also used by malware authors), raises a red flag. This obfuscation makes reverse engineering and security analysis significantly harder.

Without deobfuscating the code in `FI.SqlResUtilConfig`, it's impossible to definitively state whether it's malicious. However, the obfuscation strongly suggests the *possibility* of a backdoor, a keylogger, or another type of malicious code designed to remain undetected. This would potentially fall under the broader category of **trojans** or **information stealers**. The suspicious code has clear potential for malicious usage. Further analysis is needed to determine its true nature.