Analysis Report for: 00588E364CF77C0528CF17F84CA91937.cs

Overall Functionality

The provided code consists of several C# files from different projects within a larger solution. The core functionality revolves around a Dynamics NAV codeunit (`Codeunit7050260`) that manages change logging. This codeunit responds to events related to table logging, specifically determining whether a table should always be logged. It interacts with other parts of the Dynamics NAV system through events and function calls. The other files appear to be related to configuration, assembly information, and possibly obfuscation (indicated by the presence of a `DotfuscatorAttribute`). There is no direct malicious code, however the overall structure and function point toward a potential misuse in a malicious campaign.

Function Summaries

* **`Codeunit7050260(ITreeObject parent)`:** Constructor for the codeunit. Initializes the base class with the parent object and the codeunit ID (7050260).

* **`ObjectName`:** Getter property that returns the name of the codeunit ("GsiChangeLogManagement").

* **`OnInvoke(int memberId, object[] args)`:** This method handles function calls to the codeunit. It checks the `memberId` to determine which function is being called and processes the arguments accordingly. If `memberId` is 7050000, it calls `OnAfterIsAlwaysLoggedTable`; otherwise, it throws an error.

* **`__Construct(ITreeObject parent)`:** A static factory method for creating instances of `Codeunit7050260`.

* **`OnClear()`:** An override for a cleanup method; it's currently empty.

* **`OnAfterIsAlwaysLoggedTable(int tableID, ByRef alwaysLogTable)`:** This is the core event handler. It determines whether a given table (`tableID`) should always be logged (`alwaysLogTable`). It uses an inner scope (`OnAfterIsAlwaysLoggedTable_Scope`) to manage its execution.

* **`IndirectPermissionList`:** Getter property returning an array of indirect permissions required by the codeunit.

* **`Codeunit7050260()`:** Static constructor that initializes the `indirectPermissionList`.

* **`OnAfterIsAlwaysLoggedTable_Scope.OnRun()`:** Contains the core logic to decide whether to set `alwaysLogTable` to `true`. It checks conditions related to a `changeLogSetup` record and the `tableID`.

* **`OnAfterIsAlwaysLoggedTable_Scope(Codeunit7050260 βparent, int tableID, ByRef alwaysLogTable)`:** Constructor for the inner scope class. Initializes member variables.

* **`OnAfterIsAlwaysLoggedTable_Scope.RawScopeId`:** Getter/setter for the scope ID (appears to be obfuscated using Greek letters).

* **`OnAfterIsAlwaysLoggedTable_Scope.EventScope`:** Getter/setter for the event scope (also obfuscated).

* **`DotfuscatorAttribute(string a, int c)`:** Constructor for a custom attribute, likely used for code obfuscation. It takes two parameters that are not clearly defined, but their values are probably meaningful to the code obfuscator.

* **`DotfuscatorAttribute.A`:** Getter property for the first parameter of `DotfuscatorAttribute`.

* **`DotfuscatorAttribute.C`:** Getter property for the second parameter of `DotfuscatorAttribute`.

* **`CarregarConfiguracaoRegrasAcesso()`:** This function in `ConfigRegrasAcesso` seems to load access rule configuration. The code uses seemingly random numbers which might indicate that the function might either be empty or obfuscated.

Control Flow

* **`OnInvoke`**: This function performs a simple `if-else` check based on `memberId`. If it matches a specific ID (7050000), it calls another function; otherwise it generates an error.

* **`OnAfterIsAlwaysLoggedTable`**: This function creates and runs an instance of `OnAfterIsAlwaysLoggedTable_Scope`.

* **`OnAfterIsAlwaysLoggedTable_Scope.OnRun`**: This function has a complex conditional statement involving multiple checks using bitwise AND (`&`). Each check uses `base.CStmtHit` (which is not defined but probably a profiling/tracing mechanism), indicating possible debugging or logging features. If all conditions are true, it sets `alwaysLogTable.Value` to `true`.

Data Structures

* **`uint[] indirectPermissionList`:** An array of unsigned integers representing indirect permissions.

* **`OnAfterIsAlwaysLoggedTable_Scope`:** A nested class acting as a scope for the event handler. It manages the execution context and holds relevant data (e.g., `tableID`, `alwaysLogTable`, `changeLogSetup`).

* **`NavRecordHandle changeLogSetup`:** A handle to a record (likely in the Dynamics NAV database) that contains change log setup information.

* **`ProcDLL` and `EnumAlias` (within `ConfigRegrasAcesso`):** These appear to be custom classes for managing procedure and enumeration information within an access configuration setting.

Malware Family Suggestion

The code itself is not inherently malicious. It is a custom codeunit for a business application (Dynamics NAV). However, the obfuscation (evident in the `DotfuscatorAttribute` and unusual use of Greek letters in nested class variable names) and the fact that it manipulates logging behavior raise some concerns. A malicious actor could potentially modify this codeunit to:

* **Exfiltrate sensitive data:** By subtly altering the logging behavior, the modified codeunit could log and transmit confidential data outside the intended system boundaries.

* **Manipulate audit trails:** By changing what is logged, an attacker could cover their tracks and make it harder to detect malicious activities.

* **Establish persistence:** The codeunit might be a part of a larger infection chain where the attacker would ensure the malicious code is always executed during specific processes.

Therefore, while the code isn't directly malicious, its structure and obfuscation make it suspicious and point to a potential for misuse in a **backdoor** or **data exfiltration** malware campaign. The specific malware family would depend on the full context of the larger application and how this codeunit is integrated and used. This is a typical technique employed in advanced persistent threats (APTs).