Analysis Report for: DA1EBC4B312E168C187A9EB2E1DA2161.cs

Overall Functionality

This C# code implements a console application that acts as a wrapper for a PowerShell script. The application (`MainApp`) loads a PowerShell script from an embedded resource ("Inventario.ps1"). It then parses command-line arguments, passing them as parameters or arguments to the PowerShell script. The application provides basic error handling, including catching unhandled exceptions. It allows for extracting the embedded PowerShell script to a file, running the script with help information, and waiting for the script's completion before exiting.

Function Summaries

* **`MainApp.Main(string[] args)`:** The main entry point of the application. It initializes various objects, handles command-line arguments, sets up PowerShell execution, manages console events (like Ctrl+C), and handles errors. It returns an exit code.

* **`MainApp.CurrentDomain_UnhandledException(object sender, UnhandledExceptionEventArgs e)`:** An event handler for unhandled exceptions in the application domain. It throws a new exception wrapping the original unhandled exception.

* **`MainModule.MainModule(MainAppInterface app, MainModuleUI ui)`:** Constructor for the `MainModule` class, which implements a custom PowerShell host. It takes references to the main application and UI objects.

* **`MainModule.PrivateData`:** Getter for the PrivateData property of the PSHost, used to provide console color controls to the PowerShell script. Returns a PSObject representing the console colors

* **`MainModule.CurrentCulture`, `MainModule.CurrentUICulture`, `MainModule.InstanceId`, `MainModule.Name`, `MainModule.UI`, `MainModule.Version`:** Getters for standard PSHost properties.

* **`MainModule.EnterNestedPrompt`, `MainModule.ExitNestedPrompt`, `MainModule.NotifyBeginApplication`, `MainModule.NotifyEndApplication`:** Overridden methods from the `PSHost` interface that are not implemented in this application.

* **`MainModule.SetShouldExit(int exitCode)`:** Sets the `ShouldExit` flag and `ExitCode` in the main application, signaling the application to exit with a given code.

* **`MainModule.ConsoleColorProxy`:** A nested class providing a proxy to control console colors from the PowerShell script through the `MainModuleUI`

* **`MainModuleRawUI.ReadConsoleOutput`, `MainModuleRawUI.WriteConsoleOutput`, `MainModuleRawUI.ScrollConsoleScreenBuffer`, `MainModuleRawUI.GetStdHandle`:** These are `DllImport` calls used for low-level console manipulation. They're not user-defined functions but wrappers around Windows API functions.

* **`MainModuleRawUI.BackgroundColor`, `MainModuleRawUI.BufferSize`, `MainModuleRawUI.CursorPosition`, `MainModuleRawUI.CursorSize`, `MainModuleRawUI.ForegroundColor`, `MainModuleRawUI.KeyAvailable`, `MainModuleRawUI.MaxPhysicalWindowSize`, `MainModuleRawUI.MaxWindowSize`, `MainModuleRawUI.WindowPosition`, `MainModuleRawUI.WindowSize`, `MainModuleRawUI.WindowTitle`:** Getters and setters for properties of the `PSHostRawUserInterface`, mostly forwarding calls to the standard `Console` class.

* **`MainModuleRawUI.FlushInputBuffer`:** Clears the console's input buffer.

* **`MainModuleRawUI.GetBufferContents(Rectangle rectangle)`:** Retrieves buffer contents from a specified rectangular area.

* **`MainModuleRawUI.ReadKey(ReadKeyOptions options)`:** Reads a key from the console with specified options.

* **`MainModuleRawUI.ScrollBufferContents(Rectangle source, Coordinates destination, Rectangle clip, BufferCell fill)`:** Scrolls buffer contents.

* **`MainModuleRawUI.SetBufferContents(Rectangle rectangle, BufferCell fill)`:** Sets buffer contents for a given rectangle to a specified `BufferCell`.

* **`MainModuleRawUI.SetBufferContents(Coordinates origin, BufferCell[,] contents)`:** Sets buffer contents from an array.

* **`MainModuleUI.MainModuleUI()`:** Constructor for the `MainModuleUI` class, which implements a custom PowerShell host user interface. It initializes a `MainModuleRawUI` object.

* **`MainModuleUI.Prompt(string caption, string message, Collection descriptions)`:** Presents a prompt to the user, getting input and returning a dictionary of key-value pairs. Handles various input types including arrays and `SecureString`.

* **`MainModuleUI.PromptForChoice(string caption, string message, Collection choices, int defaultChoice)`:** Prompts the user for a choice from a list of options.

* **`MainModuleUI.PromptForCredential(string caption, string message, string userName, string targetName, PSCredentialTypes allowedCredentialTypes, PSCredentialUIOptions options)`:** Prompts the user for credentials.

* **`MainModuleUI.PromptForCredential(string caption, string message, string userName, string targetName)`:** Another overload of `PromptForCredential`

* **`MainModuleUI.RawUI`:** Getter for the `RawUI` property.

* **`MainModuleUI.ReadLine()`:** Reads a line of text from the console.

* **`MainModuleUI.ReadLineAsSecureString()`:** Reads a line of text from the console as a `SecureString`.

* **`MainModuleUI.Write(ConsoleColor foregroundColor, ConsoleColor backgroundColor, string value)`:** Writes text to the console with specified colors.

* **`MainModuleUI.Write(string value)`:** Writes text to the console.

* **`MainModuleUI.WriteDebugLine(string message)`:** Writes a debug message to the console.

* **`MainModuleUI.WriteErrorLine(string value)`:** Writes an error message to the console.

* **`MainModuleUI.WriteLine()`:** Writes a new line to the console.

* **`MainModuleUI.WriteLine(ConsoleColor foregroundColor, ConsoleColor backgroundColor, string value)`:** Writes a line of text to the console with specified colors.

* **`MainModuleUI.WriteLine(string value)`:** Writes a line of text to the console.

* **`MainModuleUI.WriteProgress(long sourceId, ProgressRecord record)`:** Writes progress information to the console. Not Implemented

* **`MainModuleUI.WriteVerboseLine(string message)`:** Writes a verbose message to the console.

* **`MainModuleUI.WriteWarningLine(string message)`:** Writes a warning message to the console.

Control Flow

The `MainApp.Main` function follows this general control flow:

1. **Initialization:** Creates instances of `MainApp`, `MainModuleUI`, and `MainModule`.
2. **Argument Parsing:** Iterates through command-line arguments (`args`), handling options like `-wait`, `-extract`, `-end`, `-?`, `-detailed`, `-examples`, `-full`, and `-debug`. A regular expression is used to parse parameters of the form `-param:value`.
3. **Script Loading:** Loads the embedded PowerShell script.
4. **Extraction or Execution:** If `-extract` is specified, the script is written to a file and the application exits. Otherwise, the PowerShell script is added to a `PowerShell` object.
5. **Parameter/Argument Passing:** Command-line arguments are passed as parameters or arguments to the PowerShell script. The way parameters are added allows for both named parameters (e.g., -param:value) and positional parameters.
6. **PowerShell Execution:** The `PowerShell` object's `BeginInvoke` method asynchronously executes the script.
7. **Event Handling:** The application waits for either the `ShouldExit` flag to be set (by the PowerShell script through the host), or for the asynchronous execution to complete.
8. **Error Handling:** Checks the invocation state of the `PowerShell` object and handles any errors.
9. **Exit:** The application exits with the appropriate exit code.

Data Structures

* **`MainApp`:** Holds the `shouldExit` flag (boolean) and `exitCode` (integer) to control application termination.

* **`MainModule`:** Manages the interaction between the C# application and the PowerShell Runspace. It includes properties to provide culture information and a unique instance ID. It utilizes a nested class `ConsoleColorProxy` to pass UI data to the PowerShell.

* **`MainModuleUI`:** Provides a custom UI for the PowerShell host, handling input/output and colors. It has properties to control the foreground and background colors for different log levels (error, warning, debug, verbose).

* **`MainModuleRawUI`:** A low level UI implementation that directly interacts with the console, relying on the Windows API to manipulate the console buffer.

* **`PSDataCollection`, `PSDataCollection`:** Used for input and output data streams in PowerShell.

* **`Dictionary`:** Used in the `MainModuleUI.Prompt` method to store the results of a user prompt as a key-value pair.

* **`SortedList`:** Used in the `MainModuleUI.PromptForChoice` method to map choice labels to their indices efficiently.

Malware Family Suggestion

While this code itself is not inherently malicious, its structure could be easily adapted for malicious purposes. The key features that raise concerns are:

* **Embedded PowerShell Script:** The ability to embed a PowerShell script allows for a high degree of obfuscation. A malicious actor could embed a script that performs harmful actions, such as downloading malware, exfiltrating data, or manipulating system settings.

* **Argument Parsing and Parameter Passing:** The flexible argument parsing enables a malicious actor to deliver command-line arguments that control the embedded script's actions. These parameters could include sensitive data or malicious commands.

* **Asynchronous Execution:** The use of asynchronous execution makes the script harder to detect and trace by traditional security tools.

Because of this capability, a likely malware family suggestion is **PowerShell-based malware**. Specifically, it could be a loader for more complex malware delivered later, a tool for post-exploitation activities, or even a standalone tool performing actions such as credential theft or data exfiltration. The functionality is sufficiently generic that it could be used in a variety of attacks.